Graph Representation Learning with Random Walk Diffusions

Abdulkadir Celikkanat

université
PARIS-SACLAY

Graph Representation Learning with Random Walk Diffusions

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 580, Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat: Informatique
Unité de recherche: Université Paris-Saclay, CentraleSupélec, Inria, Centre de Vision Numérique, 91190, Gif-sur-Yvette, France
Référent: CentraleSupélec

Thèse présentée et soutenue en visioconférence totale, le 21 avril 2021, par

Abdulkadir ÇELIKKANAT

Composition du jury:

Céline HUDELOT Professeure, CentraleSupélec, Université Paris-Saclay	Présidente & Examinatrice
Thomas BONALD Professeur, Télécom Paris, Institut Polytechnique de Paris	Rapporteur & Examinateur
Philippe CUDRÉ-MAUROUX Professeur, University of Fribourg	Rapporteur & Examinateur
Massih-Reza AMINI Professeur, Université Grenoble Alpes	Examinateur
Yuxiao DONG Chercheur, Facebook AI	Examinateur
Tina ELIASSI-RAD Professeure, Northeastern University	Examinatrice
Apostolos PAPADOPOULOS Professeur associé, Aristotle University of Thessaloniki	Invité
Nikos PARAGIOS Professeur, CentraleSupélec, Université Paris-Saclay	Directeur
Fragkiskos MALLIAROS Maître de conférences, CentraleSupélec, Inria, Université Paris-Saclay	Codirecteur

Thèse de doctorat
NNT: 2021UPASG030

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

TITRE: Apprentissage de représentations sur graphes par diffusions de marche aléatoire

MOTS CLÉS: L'apprentissage de représentations sur graphes, classification des nœuds, prédiction de liens, représentations de nœuds

RÉSUMÉ: L'objectif principal de l'Apprentissage de Représentations sur Graphes est de plonger les nœuds dans un espace vectoriel de petite dimension. Dans cette thèse, nous abordons plusieurs enjeux dans le domaine.

Tout d'abord, nous étudions comment exploiter l'existence de communautés structurelles locales inhérentes aux graphes tout en apprenant les représentations. Nous apprenons des représentations améliorées de la communauté en combinant les informations latentes avec les représentations.

De plus, nous nous concentrons sur l'expressivité des représentations. Nous mettons l'accent sur les distributions de familles exponentielles pour saisir des modèles d'interaction riches. Nous proposons un modèle qui combine les marches aléatoires avec une matrice de factorisation sous forme de noyau.

Dans la dernière partie de la thèse, nous étudions des modèles permettant un bon compromis entre efficacité et précision. Nous proposons un modèle évolutif qui calcule des représentations binaires.

TITLE: Graph representation learning with random walk diffusions

KEYWORDS: Graph representation learning, node classification, link prediction, node representations

ABSTRACT: Graph Representation Learning aims to embed nodes in a low-dimensional space. In this thesis, we tackle various challenging problems arising in the field.

Firstly, we study how to leverage the inherent local community structure of graphs while learning node representations. We learn enhanced community-aware representations by combining the latent information with the embeddings.

Moreover, we concentrate on the expressiveness of node representations. We emphasize exponential family distributions to capture rich interaction patterns. We propose a model that combines random walks with kernelized matrix factorization.

In the last part of the thesis, we study models balancing the trade-off between efficiency and accuracy. We propose a scalable embedding model which computes binary node representations.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

RÉSUMÉ

Les données structurées sous forme de graphes sont omniprésentes dans de multiples domaines d'application comme les télécommunications, la biologie, et les plateformes de réseaux sociaux. L'extraction d'informations pertinentes à partir de graphes est une tâche cruciale pour traiter les problèmes d'apprentissage en analyse de réseaux. L'objectif principal de l'Apprentissage de Représentations sur Graphes (ARG) est de plonger les nœuds du graphe dans un espace vectoriel de petite dimension, tout en préservant les propriétés structurelles du réseau. Dans cette thèse, nous abordons plusieurs enjeux liés à l'ARG. Pour cela, nous développons des algorithmes expressifs et adaptatifs capables de tirer parti de la richesse sémantique structurelle des graphes via des diffusions de marche aléatoires.

Tout d'abord, nous étudions comment exploiter l'existence de communautés structurelles locales inhérentes aux graphes tout en apprenant les représentations. En particulier, nous introduisons les Topical Node Embeddings (TNE), un cadre générique pour améliorer les capacités prédictives des modèles en s'appuyant sur des processus de marche aléatoire. Néanmoins, en apprenant ces représentations, la plupart des modèles existants ignorent les groupes de nœuds densément connectés. Notre modèle attribue une étiquette de communauté latente à chaque nœud avec l'aide de divers modèles statistiques. Nous apprenons ensuite des représentations améliorées de la communauté en combinant les informations latentes avec les représentations. Nous démontrons que les TNE améliorent les capacités prédictives des représentations.

Dans la deuxième partie de la thèse, nous nous concentrons sur l'expressivité des représentations et nous mettons l'accent sur les distributions de familles exponentielles pour saisir des modèles d'interaction riches entre les nœuds dans des séquences de marche aléatoires. Nous introduisons le modèle Exponential Family Graph Embedding (EFGE), qui généralise les techniques d'apprentissage de représentation de graphe à l'aide marche aléatoire, à des familles de distributions exponentielles. Nous étudions trois exemples particuliers de ce modèle qui correspondent à des distributions exponentielles bien connues, nous analysons leurs propriétés et montons leurs liens avec les modèles d'apprentissage non supervisé existants.

Dans la troisième partie, nous étudions l'expressivité de l'ARG en nous concentrant sur les modèles de factorisation matricielle. De nombreuses approches reposant sur la factorisation matricielle apprennent les représentations en décomposant une matrice décrivant les similitudes entre les nœuds. Néanmoins, en raison de la structure complexe des graphes du monde réel, les vecteurs appris préservent difficilement la proximité des nœuds, entraînant une perte de performances dans les tâches en aval. En outre, ces algorithmes nécessitent des ressources de calcul et de mémoire élevées en raison de la construction exacte de la matrice de similarité. Pour répondre à ces défis, nous proposons kernelNE qui combine les marches aléatoires avec une matrice de factorisation sous forme de noyau. Nous examinons une formulation d'apprentissage à plusieurs noyaux qui permet une combinaison linéaire de fonctions de noyau.

Dans la dernière partie de la thèse, nous étudions des modèles permettant un bon compromis entre efficacité et précision. Au fur et à mesure que la taille des réseaux augmente, les modèles classiques sont confrontés à des défis de calcul pour s'adapter à de grands graphes. Malgré les nombreux efforts récents pour concevoir des algorithmes adaptables sur les problèmes d'apprentissage de nœuds, la plupart de ces algorithmes ont une faible précision sur les tâches en aval. Nous proposons NodeSig, un modèle évolutif qui calcule les représentations binaires. NodeSig exploite les probabilités de diffusion de marche aléatoire via une méthode dite de "hashing", pour calculer efficacement les représentations.

ABSTRACT

Graph-structured data is ubiquitous in many application domains such as information technologies, biology, physics and social networks. The problem of extracting meaningful information from graphs is a crucial task for dealing with learning problems in network analysis. As a prominent paradigm, Graph Representation Learning (GRL) aims to embed nodes in a low-dimensional space, preserving the structural properties of the network. In this thesis, we tackle various challenging problems arising in GRL, developing expressive and scalable algorithms capable of leveraging rich structural semantics of real-world graphs via random walk diffusions.

Firstly, we study how to leverage the inherent local community structure of graphs while learning node representations. In particular, we introduce Topical Node Embeddings (TNE), a generic framework to enhance node representations' predictive capabilities relying on random walk-based methods. Most of the existing models are deprived of considering densely connected node group patterns while learning representations. Our model assigns a latent community label to each node with the favor of various statistical models and community detection algorithms. We then learn enhanced community-aware representations by combining the latent information with the embeddings. We demonstrate that TNE improves the predictive capabilities of embeddings in various tasks.

In the second part of the thesis, we concentrate on the expressiveness of node representations and we emphasize exponential family distributions to capture rich interaction patterns between nodes in random walk sequences. We introduce the Exponential Family Graph Embedding (EFGE) model, which generalizes random walk-based graph representation learning techniques to exponential family conditional distributions. We study three particular instances of this model that correspond to well-known exponential family distributions, analyzing their properties and showing their relationship to existing unsupervised learning models.

In the third part, we further study the expressiveness of GRL, focusing on matrix factorization models. Many approaches that rely on matrix factorization learn embeddings by decomposing a matrix designed to signify similarities among nodes. Nevertheless, due to the complex structure of real-world graphs, the learned embedding vectors cannot well preserve node

proximity, causing a performance loss in downstream tasks. Additionally, they require high computational and memory resources because of the exact realization of the similarity matrix. To address such challenges, we propose KernelNE, a model that combines random walks with kernelized matrix factorization. To further improve the model's performance, we examine a multiple kernel learning formulation that allows a linear combination of kernel functions.

In the last part of the thesis, we study models balancing the trade-off between efficiency and accuracy. As the size of networks increases, widely used models face computational challenges to scale to large graphs. While there is a recent effort towards designing algorithms that solely deal with scalability issues in node representation learning, most of them behave poorly in terms of accuracy on downstream tasks. Here, we propose NodeSig, a scalable embedding model which computes binary node representations. NodeSig exploits random walk diffusion probabilities via stable random projection hashing, towards efficiently computing embeddings in the Hamming space.

LIST OF PUBLICATIONS

The thesis covers the following publications and the works under review:

- Abdulkadir Çelikkanat and Fragkiskos D. Malliaros. TNE: A Latent Model for Representation Learning on Networks. In *NeurIPS Relational Representation Learning (NeurIPS-R2L) Workshop*, 2018.
- Abdulkadir Çelikkanat and Fragkiskos D. Malliaros. Kernel Node Embeddings. In *GlobalSIP,*. 2019, pp. 1–5.
- Abdulkadir Çelikkanat and Fragkiskos D. Malliaros. Learning Node Embeddings with Exponential Family Distributions. In *NeurIPS Graph Representation Learning Workshop (NeurIPS-GRL) Workshop*, 2019.
- Abdulkadir Çelikkanat and Fragkiskos D. Malliaros. Exponential Family Graph Embeddings. In *AAAI,*. 2020, pp. 3357–3364.
- Abdulkadir Çelikkanat and Fragkiskos D. Malliaros. Topic-Aware Latent Models for Representation Learning on Networks. *Pattern Recognition Letters* 144 (2021), pp. 89–96.
- Abdulkadir Çelikkanat, Yanning Shen, and Fragkiskos D. Malliaros. Multiple Kernel Representation Learning on Networks. *Manuscript* (2021).
- Abdulkadir Çelikkanat, Apostolos N. Papadopoulos, and Fragkiskos D. Malliaros. NodeSig: Random Walk Diffusion meets Hashing for Scalable Graph Embeddings. *Manuscript* (2021).

The following work has been done during the Ph.D. studies, but it is not included in the dissertation:

- Surabhi Jagtap, Abdulkadir Çelikkanat, Aurélie Pirayre, Frederique Bidard, Laurent Duval and Fragkiskos D. Malliaros. Multiomics Data Integration for Gene Regulatory Network Inference with Exponential Family Embeddings. In *EUSIPCO,*. 2021.

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my advisor, Prof. Fragkiskos Malliaros, for his invaluable guidance and support during my doctoral research. His encouragements always kept me afloat during my course of doctoral research. Apart from his precious help to my research activities, I have learned lots of things from him. His passion, keen interest in research, friendful and helpful attitude inspired me a lot. I'm highly indebted to him for accompanying me during my Ph.D. journey. I would also like to thank Prof. Nikos Paragios for believing me and providing me the opportunity of starting this journey.

I am much obliged to all the jury members of my Ph.D. thesis committee, Prof. Céline Hudelot, Prof. Thomas Bonald, Prof. Philippe Cudré-Mauroux, Prof. Massih-Reza Amini, Dr. Yuxiao Dong, and Prof. Tina Eliassi-Rad for all their insightful comments and invaluable feedback. I would like to give a special thanks to Prof. Apostolos Papadopoulos not only for attending my thesis defense but also for all his suggestions and his valuable time to review the dissertation. I am very fortunate to obtain a chance to work with him during the last two years. I'm very grateful for all of our discussions and his great help with positive energy.

I would like to express my sincerest gratitude to Mrs. Jana Dutrey for her great help and the gentle approach she showed me to deal with all the required procedures and the obstacles I encountered during my stay in France. I am very thankful to Mr. Anthony Guindon for his technical support. Many thanks to all the faculty of the CVN lab, Prof. Jean-Christophe Pesquet, Prof. Hugues Talbot, Prof. Émilie Chouzenoux, Prof. Maria Vakalopoulou, and Prof. Marc Castella for building a wonderful research atmosphere in the lab.

Many thanks to all my collaborators from IFPEN and especially to Surabhi for her outstanding effort, work, and patience. Many thanks to all the other members of CVN that I obtained a chance to share the same atmosphere –

Alexandre, Ana, Andres, Arthur, Enzo, George, Jean-Baptiste, Kavya, Marc, Maria P. Marie-Caroline, Marion, Marvin, Mathieu C. Mathieu Vu, Matthieu T. Maïssa, Mihir, Mouna, Sagar, Ségolène, Tasnim, Théodore, Xiaoyu, Ying-ping, Younes and Yunshi , and all the others that I might have missed. I also would like to thank Maria V. for the gatherings she hosted and for all her kind help. My special thanks to Ségolène and Alexandre for taking their valuable time to review the French translation of the thesis abstract.

I would also like to thank Mrs. Maria Doyet for all her kind support, considering me a member of her family, and become a friend in Palaiseau.

Last but not least, I am very grateful to my family for their endless support and love. I am so thankful to my parents, Ümmühan and Osman, my brother, Burak, as well as my uncles, Abdurrahim and İsmail, and my aunts, Şerife and Münevver. The dissertation came to true with their infinite moral support and encouragement.

"Fair is foul, and foul is fair"
William Shakespeare, Macbeth

Ne içindeyim zamanın,
Ne de büsbütün dışında;
Yekpare, geniş bir anın
Parçalanmaz akışında.
...
Kökü bende bir sarmaşık
Olmuş dünya sezmekteyim,
Mavi, masmavi bir ışık
Ortasında yüzmekteyim.

Ahmet Hamdi Tanpınar

"Je m'en vais enfin de ce monde, où il faut
que le coeur se brise ou se bronze."
Sébastien Nicolas de Chamfort

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

SYMBOL	MEANING
γ	Window size
χ	Chi similarity
$\mathbf{A}$[v]	Representation of context node v
$\mathbf{B}$[v]	Representation of center node v
$\mathcal{C}$	Number of connected components
$\mathcal{C}_\gamma^{(l)}(\mathbf{w})$	Context set of node w_l in the walk $\mathbf{w}$
d	Representation size
$\mathcal{E}$	Edge set
G	Graph
k	Number of negative samples
$\mathcal{K}$	Number of communities
K	Kernel function
$\mathcal{L}$	Walk length
N	Number of walks
$\mathcal{N}(v)$	The set neighbors of node v
$\mathbf{P}$	Right stochastic matrix
$\mathcal{V}$	Vertex set
$\mathbf{w}$	Walk
$\mathbf{W}$	Adjacency matrix
$\mathcal{W}$	The set of walks
X	Embedding space

LIST OF ACRONYMS/ABBREVIATIONS

AUC	Area Under Curve
BFS	Breadth First Search
DFS	Depth First Search
GCN	Graph Convolutional Network
GNN	Graph Neural Network
GRL	Graph Representation Learning
HMM	Hidden Markov Model
IHMM	Infinite Hidden Markov Model
Isomap	Isometric Feature Mapping
JL	Johnson-Lindenstrauss
LDA	Latent Dirichlet Allocation
LLE	Locally Linear Embedding
MCMC	Monte Carlo Markov Chain
MDS	Multidimensional Scaling
NCE	Noise Contrastive Estimation
NLP	Natural Language Processing
NMI	Normalized Mutual Information
PCA	Principal Component Analysis
PPMI	Positive Pointwise Mutual Information
SBM	Stochastic Block Model
SGD	Stochastic Gradient Descent
SVD	Singular Value Decomposition

1

INTRODUCTION

With the advancements in information technologies, graphs have become indispensable mathematical objects to model and represent the interactions among entities. They naturally emerge in numerous disciplines, including physics, chemistry, social sciences, and all related fields. Protein-protein interaction networks, recommender systems, and friendship networks are only a few examples of these different domains [New03]. They constitute ubiquitous elements not only for their ability to model complex structures but they also provide an elegant framework to analyze and study the intricate patterns composed by the individual units of the network [AO04].

Graph analysis is important to gain insights into the network's hidden patterns and perform various predictive tasks by using the extracted information. For instance, one might wish to estimate the protein function in multi-omics data [GBB18], recommend new products to a customer [Lu+15], and find a community of people sharing the same particular hobby in a social network [For10]. Traditional techniques aim to carefully design feature vectors by choosing essential graph measures concerning the task of interest, relying on the network's structural properties, such as clustering coefficient and core number [Mal+20]. These handcrafted feature vectors are further incorporated into machine learning models to carry out the desired tasks.

Although these graph measures provide information about the various properties of a node, they are unable to capture the interactions among nodes sufficiently—a critical point in learning and prediction tasks. For instance, despite the fact that the degree of a node describes well the size of its local neighborhood, it cannot reveal further structural properties related to these neighbors and their interactions. Likewise, while the clustering coefficient and k-core number can provide insights about the connectivity patterns of a node, it is strenuous to properly leverage them towards inferring

relationships among nodes which could be utilized by a machine learning model. Furthermore, features corresponding to node pairs can be extracted through graph similarity-based algorithms, such as the *Jaccard* and *Adamic-Adar* indices [LZ11]. Nevertheless, such features are often limited to specific networks or downstream tasks (e.g. link prediction), and also might require an extensive computational workload [LZ11].

Therefore, classical machine learning techniques on graphs that rely on handcrafted features are not effective in practice, suffering also from high space and computational cost. To this end, the research community have inclined to alternative approaches, leading to the birth of learnable models which mainly aim to automatically extract these features (representations) from the underlying graphs [GF18; Ham20]. The core idea of Graph Representation Learning (GRL) is to learn representations (also known as embeddings) of nodes in a lower-dimensional space, in which the embedding vectors reflect properties of interest. Due to the superior performance of GRL on various practical applications, recently we have witnessed an immense increase in the number of relevant studies [Ham20].

Representation learning is not a new field; it has already been scrutinized in various domains, such as signal processing, natural language processing, and object recognition [BCV13]. Although the evolution of the graph representation learning field [GF18; CZC18] has been highly influenced by studies from diverse domains, developing an effective and efficient algorithm poses plenty of challenges. Unlike other data structures, networks have additional information indicating the relationships between pairs of nodes. Hence, a GRL model should be able to capture various topological patterns and underlying properties of the network. It is already well-known that real-world networks do not emerge from random and irregular forms. In fact, they share interesting and non-trivial characteristics, such as a heavy tail in the degree distribution and community structures [AB02]. Therefore, such *topological properties* should be properly leveraged in the learning procedure [Fen+18; Wan+17]. Although plenty of approaches have been developed in the literature, emphasizing different aspects of networks in the representation learning process, most of them have commonly prioritized the algorithm's *accuracy* in downstream tasks, such as classification and link prediction.

Graph representation learning approaches consider various characteristics of networks in learning the representations. However, these features need to be expressed under a suitable model in order to convey the extracted information in the embeddings effectively. Therefore, the *expressiveness* of the representations is another criterion influencing the quality of the embeddings. On the other hand, the growing size of networks entails the need for *scalable* techniques that can run on networks consisting of millions of nodes and edges. Many previously proposed models are not practically applicable due to the high computational resource demands—thus, algorithms balancing the effectiveness and the scalability have become a prominent direction recently.

To that end, designing effective and efficient algorithms is of great significance for applications that involve learning from graph-structured data. Therefore, we consider the following challenges and points of interest, while developing GRL models:

- How to explicitly incorporate information about the rich structural semantics of real-world graphs in the GRL process.
- How to design models that could properly capture the complex interactions and relationships among nodes, towards improving expressiveness.
- How to design scalable GRL models that can efficiently handle large graphs.

1.1 THESIS STATEMENT AND OVERVIEW OF CONTRIBUTIONS

The dissertation's crux motivation is to study network analysis with graph representation learning approaches. In light of the challenges stated in the previous section, we focus on developing representation learning methods relying on random walks, in an unsupervised manner. Figure 1.1 gives an overview of the contributions of this thesis, which can be summarized as follows:

EXPLICIT INTEGRATION OF COMMUNITY INFORMATION. *How to incorporate the community structure of networks in learning node representations?*

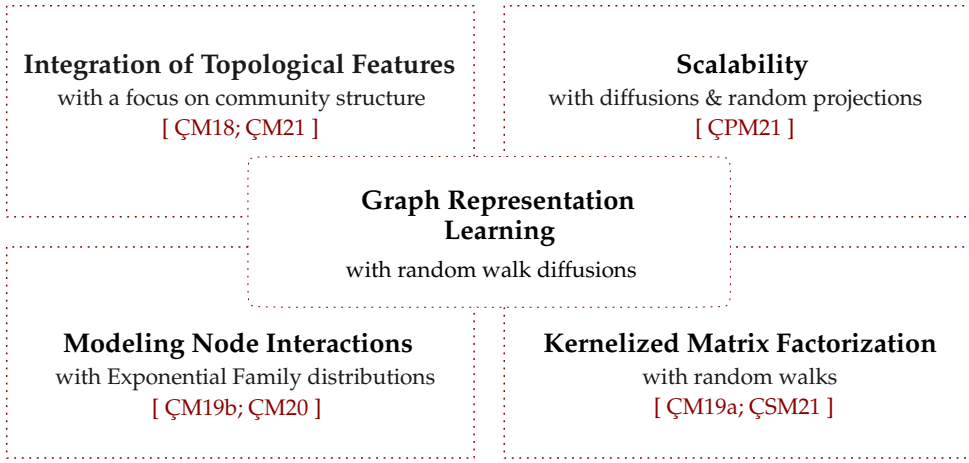


Figure 1.1: Overview of the contributions of the thesis.

Many complex networks comprise groups of nodes having tight connections than the rest of the network. These substructures naturally emerge during the growth of networks. For instance, in social networks, people sharing similar hobbies or interests tend to be closer to each other; similarly, proteins interacting with each other are more likely to share the same cellular functions forming densely connected groups. Although various algorithms have been developed to detect these patterns, they are disregarded in the design of GRL models. We introduce a novel approach, TNE (Topical Node Embeddings) [ÇM18; ÇM21], which aims to integrate the community structure of a network in the embedding learning process. TNE is a general framework that can be applied with various random walk-based methods to learn community-aware embeddings. We examine the behavior of the proposed framework by adapting various latent community detection algorithms. We evaluate its performance in node classification and link prediction tasks (Chapter 3).

MODELING NODE INTERACTIONS WITH EXPONENTIAL FAMILY. *How to model and capture the underlying pattern of node pairs properly within random walks sequences?*

Previously proposed random walk-based approaches, although applying different strategies to generate node sequences, they mainly utilize the *soft-*

max or *sigmoid* functions to model the distribution of nodes within random walks. Nevertheless, this might prohibit to capture richer types of interaction patterns among nodes that co-occur within a random walk. Here, we introduce a family of models, called EFGE (Exponential Family Graph Embeddings) [ÇM19b; ÇM20], which generalizes the conventional approaches with exponential family distributions. EFGE allows employing a wide range of conditional distributions to interpret the complex relationships among nodes; hence, we can learn more expressive embeddings conveying the intricate patterns among nodes in the network. We also show the connection between BigClam [YL13], a widely-used overlapping community detection, and an instance of our proposed model, building a bridge between two different graph mining applications (Chapter 4). Our extensive experimental evaluation deomostrates that leveraging exponential family distributions can further boost the predictive ability of the embeddings on downstream tasks.

A KERNELIZED MATRIX FACTORIZATION FRAMEWORK BASED ON RANDOM WALKS. *How to augment the predictive capacity of random walk-based matrix factorization methods with kernel functions?*

Kernel functions are generally integrated with linear models to map the data points into a higher-dimensional space, so that they are transformed into a structure that can be expressed with linear approaches in the new space. Although there are various graph representation learning approaches relying on matrix factorization, they follow linear techniques to decompose the designed target matrix. We propose a novel model, called KernelNE (Kernel Node Embeddings) [ÇM19a; ÇSM21], which incorporates universal kernels in learning representations. It bypasses the exact realization of the target matrix through random walks, thereby alleviating the optimization step's computation burden. We further enhance the model's capability with MKernelNE, which allows combining multiple kernels. The experimental evaluation demonstrates that the integration of kernels provides to the proposed approach the ability to grasp the underlying patterns of complex data, leading to more expressive node representations (Chapter 5).

SCALABLE NODE EMBEDDINGS. *How to develop an algorithm balancing scalability and efficiency on downstream tasks?*

Real-world networks have been continuously growing with newly added links and nodes—thus, the algorithms dealing with large-scale networks have become essential. Nevertheless, most of the previously proposed approaches are not capable of dealing with large networks due to high computational resource requirements. Recently, random projection-based node embedding algorithms have received significant attention, even though showing lower predictive capabilities on downstream tasks compared to learning-based methods. Here, we propose NodeSig [ÇPM21], a model which targets to balance the trade-off between efficiency and accuracy. It brings together random walk diffusion and hashing techniques relying on α-stable random projections. The design of the proposed methodology provides efficient projections of target node vectors recursively. It computes binary representations in the Hamming space, in which the pairwise distances approximate the chi similarity among the initially designed node vectors (Chapter 6). The experimental evaluation shows that NodeSig achieves good accuracy on downstream tasks compared to recent highly-scalable models, while at the same time being able to run within reasonable time.

1.2 OUTLINE OF THE THESIS

The rest of the dissertation is organized as follows. In Chapter 2, we briefly describe the fundamental elements of graph theory and probability theory. We also summarize the methods developed in the graph representation learning field. Later, we provide the details of the experimental evaluation and a description of the networks used in the experiments. Chapter 3 shows how to enhance node embeddings by means of the community structure of networks with the TNE model. Chapter 4 introduces the EFGE model that leverages exponential family distributions to better capture the underlying interaction patterns of node pairs within the generated node sequences. Chapter 5 presents KernelNE and MKernelNE, which combine kernel functions with matrix factorization. Chapter 6 is devoted to NodeSig, the proposed scalable model that computes binary node embeddings. Finally,

the arguments concluding the dissertation and possible future research directions are stated in Chapter 7.

2

PRELIMINARIES AND OVERVIEW OF RELATED WORK

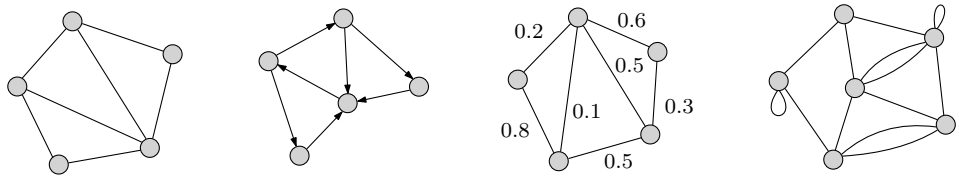
In this chapter, we provide the fundamental concepts and background materials necessary throughout the dissertation. In the beginning, we describe the essential elements related to graph theory and probability theory. Moreover, we provide a general overview concerning graph representation learning methods in a separate section. For more detailed information on the field, the reader might refer to [Ray13; KT05; Tao13; Ham20; HYL17b; Zha+20; GF18; CZC18]. Finally, we discuss the experiments that we perform to evaluate the performance of the algorithms and the network datasets. Throughout the dissertation, we will use the notation $\mathbf{M}$ to denote a matrix, while the term $\mathbf{M}_{(i,j)}$ points out the entry located at the i'th row and j'th column of the matrix. $\mathbf{M}_{(i,:)}$ and $\mathbf{M}[i]$ indicate the i'th row of the matrix.

2.1 BASICS OF GRAPH THEORY

Graphs are the key elements to study objects and the relationships among them. Throughout the dissertation, we use the terms network and graph interchangeably. More formally, a graph is defined as follows:

Definition 2.1. *A graph is an ordered pair $G = (\mathcal{V}, \mathcal{E})$ consisting of non-empty sets $\mathcal{V}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The elements of $\mathcal{V}$ are called the vertices or nodes of G, and those of $\mathcal{E}$ the edges of G.*

The edges of the graph might also indicate direction. A *directed* graph or *digraph* is a graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{E}$ consists of the ordered pairs of nodes. In this case, the pairs (u, v) and (v, u) do not represent the same edge. It might be also allowed to have *multiple edges* or *parallel edges* which are two or more edges joining the same pair of nodes, and the graph containing parallel edges is called *multigraph*.



a) Simple graph b) Directed graph c) Weighted graph d) Multigraph with loops

Figure 2.1: Illustration of different graph types. (a) A simple graph is an unweighted and undirected graph without loops and multi edges; (b) For a directed graph, edges indicate an orientation, and (c) edges can have weights in a weighted graph. (d) In a multigraph, having multiple edges or loops are permitted.

Definition 2.2. *A graph $G = (\mathcal{V}, \mathcal{E})$ is called weighted if there is a weight function $\mathbf{W} : \mathcal{E} \to \mathbb{R}$ assigning each edge to a real number.*

Weighted graphs naturally arise in many applications, which enable the edges to convey additional information. For instance, it can be used to represent the distance or the cost of traveling between cities. The endpoints of an edge can also be connected to the same node, and such an edge (v, v) is called *loop*. In Figure 2.1, we depict various graph types to illustrate the given definitions.

Definition 2.3. *An unweighted, undirected graph containing no loops or multiple edges is called simple graph.*

Throughout the dissertation, we solely consider simple graphs to provide consistency in the experiments unless stated otherwise. Nodes u and v are *adjacent* or *neighbours*, if the pair (u, v) is an element of the edge set $\mathcal{E}$. The *degree* of a node is defined as the number of its neighbours.

Another representation of a graph is the matrix form, with the *adjacency matrix* being the most commonly used.

Definition 2.4. *The adjacency matrix of a simple graph G is the $|\mathcal{V}| \times |\mathcal{V}|$-matrix $\mathbf{M}$ with entries $\mathbf{M}_{(v,u)} = 1$ if $(v, u) \in \mathcal{E}$ and $\mathbf{M}_{(v,u)} = 0$ if $(v, u) \notin \mathcal{E}$.*

For a weighted graph, the entries indicate the corresponding weights of the edges. Note that, the adjacency matrix is always symmetric for undirected graphs according to its diagonal.

Two graphs might have exactly the same structure, and they can have different namings. More formally, we can express it in the following way:

Definition 2.5. *Two graphs $G = (\mathcal{V}_G, \mathcal{E}_G)$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$ are isomorphic if there exits a bijection $f : \mathcal{V}_G \to \mathcal{V}_H$ such that $(u, v) \in \mathcal{E}_G$ if and only if $(f(u), f(v)) \in \mathcal{E}_H$ for all $v, u \in \mathcal{V}_G$.*

A graph might also be a part of the structure of another one or we might be interested to consider only a part of a graph.

Definition 2.6. *A graph $H = (\mathcal{V}_H, \mathcal{E}_H)$ is a subgraph of $G = (\mathcal{V}_G, \mathcal{E}_G)$ if $\mathcal{V}_H \subseteq \mathcal{V}_G$ and $\mathcal{E}_H \subseteq \mathcal{E}_G$.*

One might be interested in visiting a node by starting from one another, but there must be a path from this pair of nodes. More formally, we define it as follows:

Definition 2.7. *Let $\boldsymbol{w} = (v_1, \ldots, v_{\mathcal{L}})$ be sequence where $v_l \in \mathcal{V}$ for all $l \in \{1, \ldots, \mathcal{L}\}$. Then, it is called a walk of length $\mathcal{L}$ from node v_1 to $v_{\mathcal{L}}$ if $(v_l, v_{l+1}) \in \mathcal{E}$ for all $l \in \{1, \ldots, \mathcal{L} - 1\}$. A walk $\boldsymbol{w} = (v_1, \ldots, v_{\mathcal{L}})$ is closed if $v_1 = v_{\mathcal{L}}$. It is a path if the nodes in the walk are distinct, i.e. $v_i \neq v_j$ for all $i \neq j$. It is called cycle if it is closed and $v_i \neq v_j$ for $i \neq j$ except that $v_1 = v_L$.*

It is also possible to have several distinct paths between a pair of nodes. However, It might also be important to find the shortest one since it can provide the best solution for efficiency in many practical applications such as transportation and robotics. Let $G = (\mathcal{V}, \mathcal{E})$ be a graph with a weight function $\mathbf{W} : \mathcal{E} \to \mathbb{R}$. Then, the minimum distance $d_G(v, u, \mathbf{W})$ is defined by

$$\min \left\{ \sum_{l=1}^{\mathcal{L}-1} W[(v_l, v_{l+1})] \;\middle|\; (v_1, \ldots, v_l) \text{ is a walk such that } v := v_1 \text{ and } u := v_l \right\}.$$

If there is no walk from v to u, the distance is defined by $d_G(v, u, \mathbf{W}) := \infty$. There are several algorithms to compute the shortest paths in a graph such as *Dijkstra's* algorithm and *Floyd–Warshall* algorithm [Ray13; KT05]. If the distance $d_G(v, u, \mathbf{W}) < \infty$ for all $u, v \in \mathcal{V}$, then the graph G is called *connected*; otherwise it is *disconnected*.

2.2 BASICS OF PROBABILITY THEORY

In this section, we briefly present the essential elements of probability theory. Let X be a non-empty set and let $P(\mathsf{X})$ be its power set which consists of all subsets of X.

Definition 2.8. *A nonempty collection of subsets $\mathcal{X} \subseteq \mathcal{P}(\mathsf{X})$ is σ-algebra (or σ-field) if the following three conditions hold:*

- $\mathsf{X} \in \mathcal{X}$;
- *if $A \in \mathcal{X}$, then $\mathsf{X} \backslash A \in \mathcal{X}$;*
- *if a sequence of elements $A_1, A_2, \ldots \in \mathcal{X}$, then $\cup_{i=1} A_i \in \mathcal{X}$.*

We follow the conventional approach and the σ-algebra $\mathcal{B}(\mathsf{X})$ for X is used to denote its *Borel* σ-algebra, i.e., the smallest σ-algebra containing all open sets in X [Tao13].

Definition 2.9. *A measurable space $(\mathsf{X}, \mathcal{X})$ is an ordered pair where $\mathcal{X}$ is a σ-algebra of X, and the elements of $\mathcal{X}$ is called measurable sets.*

Definition 2.10. *A function μ from $\mathcal{X}$ to $\widetilde{\mathbb{R}}$ is called measure on $(\mathsf{X}, \mathcal{X})$ if it satisfies*

- $\mu(\emptyset) = 0$;
- *For all $A \in \mathcal{X}$, $\mu(A) \geq 0$;*
- *For any finite or countable collections of pairwise disjoint sets $A_i \in \mathcal{X}$, it satisfies $\mu\left(\cup_{i=1} A_i\right) = \sum_{i=1} \mu(A_i)$.*

We use the notation $\widetilde{\mathbb{R}}$ to denote the extended real number system ($\widetilde{\mathbb{R}} := \mathbb{R} \cup \{-\infty, +\infty\}$). A *probability measure* $\mu : \mathcal{X} \to [0, 1]$ is a measure satisfying $\mu(\mathsf{X}) = 1$. A *probability space* is an ordered pair $(\mathsf{X}, \mathcal{X}, \mu)$ where μ is a probability measure defined on the measurable space $(\mathsf{X}, \mathcal{X})$.

Definition 2.11. *A mapping X from a probability space $(\mathsf{X}, \mathcal{X}, \mu)$ to a measurable space $(\mathsf{Y}, \mathcal{Y})$ is called random variable if X is a measurable function, which means that for every $B \in \mathcal{Y}$ its preimage $X^{-1}(B) = \{a : X(a) \in B\}$ is in $\mathcal{X}$.*

Definition 2.12. *For a random variable $X : \mathsf{X} \to \mathsf{Y}$, the measure ν on $(\mathsf{Y}, \mathcal{Y})$ defined by $\nu(B) := \mu\left(X^{-1}(B)\right) = \mu\{X \in B\}$ for every $B \in \mathcal{Y}$ is called the distribution of the random variable X.*

Definition 2.13. *A discrete-time stochastic process on a countable set Y is a collection of Y-valued random variables $\{X_t : t \geq 0\}$ defined on a probability space $(\mathsf{X}, \mathcal{X}, \mu)$. It is called Markov Chain if it satisfies*

$$\mu\{X_{t+1} = y_{t+1} | X_t, \ldots, X_0\} = \mu\{X_{t+1} = y_{t+1} | X_t\},$$

for all $y_{t+1} \in \mathsf{Y}$ and $t \geq 0$.

Definition 2.14. *A Markov chain $X = \{X_t : t \geq 0\}$ with memory m is a stochastic process satisfying*

$$\mu\{X_{t+1}{=}y_{t+1} | X_t{=}y_t, \ldots, X_0{=}y_0\} = \mu\{X_{t+1}{=}y_{t+1} | X_t{=}y_t, \ldots, X_{t+1-m}{=}y_{t+1-m}\}$$

for $t \geq m$.

Definition 2.15. *A Markov chain $X = \{X_t : t \geq 0\}$ is time-homogeneous if $\mu\left(X_{t+1} = s | X_t = r\right) = \mu\left(X_t = s | X_{t-1} = r\right)$ for all $t \geq 1$ and $s, r \in \mathsf{Y}$.*

The *transition* probabilities can be also represented by a matrix $\mathbf{P}$ where each entry is defined by $\mathbf{P}_{(s,r)} := \mu\left(X_{t+1} = r | X_t = s\right)$. The dissertation mainly considers the methods relying on random walks, and we define a *uniform random walk* as follows:

Definition 2.16. *A uniform random walk of length $\mathcal{L}$ on a graph $G = (\mathcal{V}, \mathcal{E})$ with a root node $v_1 \in \mathcal{V}$ as a time-homogeneous stochastic process with random variables $\{X_l : 1 \leq l \leq \mathcal{L}\}$ such that $X_1 = v_1$ and X_{l+1} is a node chosen uniformly at random from the set of neighbors of X_l for each $l \in \{1, \ldots, \mathcal{L} - 1\}$.*

2.3 GRAPH REPRESENTATION LEARNING: AN OVERVIEW

The history of *graph embeddings* can be dated back to the 1900s, with the initial works in the topological graph theory field studying the embedding of graphs in surfaces [Arc96]. However, the graph representation learning field

has shown impressive evolution and expansion during the past six years as a subfield of machine learning, and recent studies have considered more general types of problems. The intuition behind the approaches relies on finding an embedding vector in a lower-dimensional space such that the desired information in the network can be captured by the pairwise distances in the new space. By following the definition provided in [HYL17b; Ham20], it can be more formally stated in the following way.

Definition 2.17. *Let $G = (\mathcal{V}, \mathcal{E})$ be a graph, $(\mathsf{X}, d_{\mathsf{X}})$ be a metric space and $s_G(\cdot, \cdot)$ be a user-specified function showing the pairwise proximities among nodes, our aim is to find a mapping $\mathbf{E} : \mathcal{V} \to \mathsf{X}$ minimizing the function*

$$\frac{1}{|\mathsf{T}|} \sum_{(v,u)\in\mathsf{T}} \ell\Big(d_{\mathsf{X}}\big(\mathbf{E}[v], \mathbf{E}[u]\big), s_G(v,u)\big)\Big), \tag{2.1}$$

where ℓ is the error or loss function defined by the user to measure the discrepancy between correct and the estimated proximity values, and $\mathsf{T} \subseteq \mathcal{V} \times \mathcal{V}$ is the training set.

By optimizing the above function, we can learn node representations $\mathbf{E}[v] \in \mathsf{X}$ for each node $v \in \mathcal{V}$. An immediate and common choice for the *embedding space* X is the real vector space $\mathbb{R}^d$ for many approaches and we will call d embedding or representation size throughout the thesis. In the remaining part of this section, we provide a brief overview and a categorization of various notable instances in the graph representation learning field.

2.3.1 *Dimensionality Reduction*

The early works of graph embeddings have relied on classical dimensionality reduction techniques. One of the first approaches is Principal Component Analysis (PCA) [JC16], which projects the adjacency matrix of the network or a designed data matrix into a latent space in which the variance of the data is maximized. Linear Discriminant Analysis [Tha+17] is a supervised method contrary to PCA and projects the data on a subspace in which the class-separation is maximized. Multidimensional Scaling (MDS) [KW78] targets to position a set of points by preserving given pairwise Euclidean

distances. Since these methods assume that data lay on a linear subspace, they might fail if the underlying data posses a highly nonlinear pattern. Therefore, alternative ideas have arisen in order to overcome this problem. Isometric Feature Mapping (ISOMAP) [TSL01] extends MDS by integrating with the geodesic distances along the manifold. Locally Linear Embedding (LLE) [RSooa] finds a projection of the data by maintaining distances within local neighborhoods. Lastly, kernel methods or tricks [HSS08] are generally used to map the non-linearly separable data points to a higher-dimensional space. They allow computing the inner products in the new space without exactly needing to know the mapping function so that the linear models can be adapted to capture the intrinsic characteristic of the complex data points.

2.3.2 *Matrix Factorization-Based Approaches*

The approaches relying on matrix factorization learn representations by decomposing a target matrix $\mathbf{T} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ designed based on a proximity measure. The graph factorization algorithm [Ahm+13] is one of the earliest works, which proposes an immediate and natural idea to learn embeddings by minimizing the following objective function:

$$\underset{\mathbf{E}}{\operatorname{argmin}} \frac{1}{2} \sum_{(v,u)\in\mathcal{E}} \left(\mathbf{T}_{(v,u)} - \mathbf{E}[v] \cdot \mathbf{E}[u]^\top\right)^2 + \frac{\lambda}{2} \sum_{v\in\mathcal{V}} \|\mathbf{E}[v]\|^2,$$

where the target matrix $\mathbf{T}$ is chosen as the adjacency matrix of the given graph and λ is the regularization parameter. TADW [Yan+15] combines text features of nodes into network representation learning under the matrix factorization framework as follows:

$$\underset{\mathbf{A},\mathbf{B}}{\operatorname{argmin}} \left\|\mathbf{T} - \mathbf{A}^\top\mathbf{B}\mathbf{X}\right\| + \frac{\lambda}{2}\left(\|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2\right).$$

Here, the text features corresponding to each node are represented by $\mathbf{X} \in \mathbb{R}^{f\times|\mathcal{V}|}$. The target matrix is selected as $\mathbf{T} = 0.5 \cdot \left(\mathbf{W} + \mathbf{W}^2\right)$ where f is the feature vector size and $\mathbf{W}$ is the adjacency matrix. HOPE [Ou+16] demonstrates that several high-order proximity measurements such as *Katz*

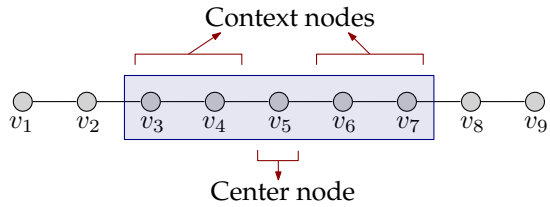


Figure 2.2: Illustration of context and center nodes for a window of size 2. The nodes around v_5 located inside the window are called *context* nodes.

index, *Personalized Pagerank* matrix, and *Adamic-Adar* index share a general formulation that can be expressed as $\mathbf{T} = \mathbf{A}^{-1}\mathbf{B}$. The method obtains embedding vectors by factorizing a high-order proximity matrix with an efficient variation of Singular Value Decomposition (SVD) [Hoco9]. GraRep [CLX15] extracts the embeddings by concatenating the factorizations of positive k-step log probabilistic matrices obtained by SVD. NetMF [Qiu+18] similarly applies SVD for the shifted Positive Pointwise Mutual Information (PPMI) of the node co-occurrence matrix.

2.3.3 *Random Walk-Based Approaches*

The advancements in Graph Representation Learning have been highly inspired by algorithms introduced in the Natural Language Processing (NLP) field. One of the most prominent examples is the class of random walk-based approaches. They generate a set of node sequences by following a random walk strategy, as an analogy to the sentences in documents. Then, they learn embedding vectors by means of a technique borrowed from the pioneering work SkipGram [Mik+13a; Mik+13b]. The idea mainly relies on learning representations by optimizing the likelihood of observing the nodes around a given node. That generates similar embeddings for nodes tend to share common surroundings in the walks.

For a walk $\mathbf{w} = (v_1, \ldots, v_{\mathcal{L}}) \in \mathcal{V}^{\mathcal{L}}$, the surrounding nodes $v_{l-\gamma}, \ldots, v_{l-1}$, $v_{l+1}, \ldots, v_{l+\gamma}$ located within a certain distance γ around *center* node v_l

composes the *context* set of node v_l. The embeddings are further learned by maximizing

$$Pr(v_{l-\gamma}, \ldots, v_{l-1}, v_{l+1}, \ldots, v_{l+\gamma} | v_l).$$

By assuming conditional independence, the objective can be written as

$$\mathcal{F}(\Omega) = \underset{\Omega}{\operatorname{argmax}} \frac{1}{\mathcal{N} \cdot \mathcal{L}} \sum_{\mathbf{w} \in \mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{-\gamma \leq j \leq \gamma \\ j \neq 0}} \log Pr(v_{l+j} | v_l; \Omega), \tag{2.2}$$

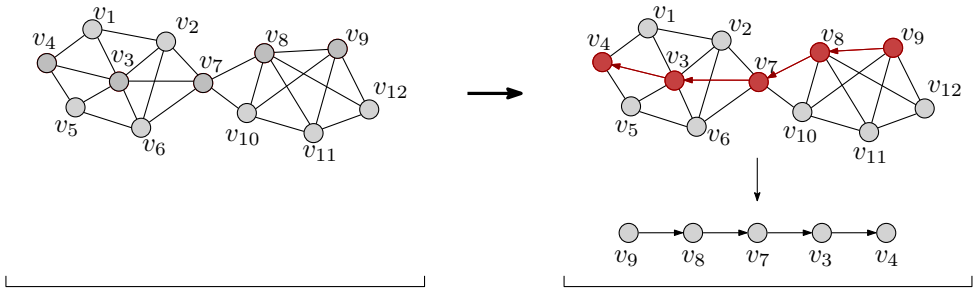
where $\mathbf{w} = (v_1, \ldots, v_{\mathcal{L}}) \in \mathcal{W}$ is a walk of length $\mathcal{L}$, γ is the window size, $\mathcal{N} := |\mathcal{W}|$ is the number of walks and $\Omega = (\mathbf{A}, \mathbf{B})$ is the set of model parameters or representations. Here, $\mathbf{A}[v]$ is used for denoting the embedding vector of v if it is considered as *context* node; otherwise $\mathbf{B}[v]$ indicates its embedding if it is *center* node. A typical choice for the probability measure is *softmax* function defined by

$$Pr(v_{l+j} | v_l) := \frac{\exp\left(\mathbf{A}[v_{l+j}]^\top \cdot \mathbf{B}[v_l]\right)}{\sum_{v \in \mathcal{V}} \exp\left(\mathbf{A}[v]^\top \cdot \mathbf{B}[v_l]\right)}.$$

However, calculating $Pr(v_{l+j}, v_l)$ for each pair (v_{l+j}, v_l) is not feasible due to the normalization term so an approach such as *Hierarchical Softmax* [MH09] can be applied to alleviate the computational complexity from $\mathcal{O}(|\mathcal{V}|)$ to $\mathcal{O}(\log(|\mathcal{V}|))$. Another alternative is the so-called *Negative Sampling* technique [Mik+13b],which is a variation of Noise Contrastive Estimation (NCE) [MT12; MK13]. It defines the conditional probability in a different way and samples negative instances for each center-context pair in order to approximate Equation (2.2) with the sigmoid function:

$$\mathcal{F}(\Omega) = \underset{\Omega}{\operatorname{argmax}} \frac{1}{\mathcal{N} \cdot \mathcal{L}} \sum_{\mathbf{w} \in \mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{-\gamma \leq j \leq \gamma \\ j \neq 0}} \Big(\log Pr(v_{l+j} | v_l) + \sum_{\substack{i=1 \\ s_i \sim p^-}}^{k} \mathbb{E}\left[\log(1 - Pr(s_i | v_l)) \right] \Big),$$

where k is the number of negative samples, the probability measure is chosen as the sigmoid function $\sigma(x) := (1 + \exp(-x))^{-1}$ and p^- is the user-specified distribution to generate the negative instances.



a) Initial network b) Generation of random walks

Figure 2.3: Illustration of a random walk generation.

DeepWalk [PARS14] is the first proposed random walk-based approach. It produces walks by uniformly choosing the next node at random from the neighbors of the current one in which it resides. Node2Vec [GL16] leverages the idea with additional parameters to control the behavior of the random walk as Breadth First Search (BFS) or Depth First Search (DFS) explorations. More precisely, it selects the next node with a probability proportional to the coefficient $\pi_{v,s} = \alpha_{p,q}(u,s)\mathbf{W}[v,s]$ where $\alpha_{p,q}(u,s)$ is defined by

$$\alpha_{p,q}(u,s) = \begin{cases} \frac{1}{p} & x \leq d(u,s) = 0 \\ 1 & 0 \leq x \leq d(u,s) = 1 \\ \frac{1}{q} & d(u,s) = 2, \end{cases}$$

Each $\mathbf{W}[v,s]$ represents the weight of the edge, v is the current node in which the walk resides, u is the previously visited node just before v, $d_G(u,s)$ represents the distance between nodes u and s, and takes values from the set $\{0,1,2\}$. For low values of *return* parameter p, the walk inclines to move to already visited nodes, and it explores unvisited nodes for small values of *in-out* parameter q. BiasedWalk [NM18] extends it by introducing an additional parameter to estimate how far a candidate node is from the initial node of each walk.

As the NetMF method presented earlier and it does not explicitly simulates random walks, it can also be considered as a random walk based approach. In fact, the authors of NetMF have shown that the objective function of various random walk-based methods can be expressed as a matrix factorization task

under certain conditions, drawing inspirations from Levy and Goldberg [LG14]. In particular, NETMF learns embeddings by factorizing the shifted PPMI matrix defined by

$$\mathbf{T} := \log\left(\max\left(\frac{vol(G)}{k \cdot \mathcal{L}} \sum_{l=1}^{\mathcal{L}} \mathbf{P}^{(l)}, 1\right)\right) \mathbf{D}^{-1},$$

where k is the number of negative samples, $\mathbf{D}$ is the diagonal matrix with row sums of the adjacency matrix $\mathbf{W}$ of the graph, $\mathbf{P} := \mathbf{D}^{-1}\mathbf{W}$ and $vol(G)$ is defined by $\sum_{v,u \in \mathcal{V}} \mathbf{W}_{(v,u)}$.

2.3.4 *Neural Network-Based Approaches*

Neural networks are undoubtedly one of the most outstanding models in recent years. They have been successfully applied in many fields, from computer vision to NLP, with plenty of diverse architectures and designs [GBC16]. Various methods relying on neural network architectures have been proposed for graph-structured data. Here, we only name some of the very well-known methods. SDNE [WCZ16] proposes a deep autoencoder model under a semi-supervised architecture framework, jointly optimizing the first-order and second-order proximities:

$$\begin{aligned}
\underset{\mathbf{B}}{\operatorname{argmin}} \left\| (\hat{\mathbf{X}} - \mathbf{X}) \odot \mathbf{C} \right\|_F^2 + \alpha \sum_{(v,u) \in \mathcal{V}^2} s_{v,u} \left\| \mathbf{H}^{(L)}_{(:,v)} - \mathbf{H}^{(L)}_{(:,u)} \right\|_2^2 + \frac{\nu}{2} \sum_{l=1}^{L} \left(\|\mathbf{B}^{(l)}\|_F^2 + \|\hat{\mathbf{B}}^{(l)}\|_F^2 \right) \\
\mathbf{H}^{l}_{(:,v)} = \sigma(\mathbf{B}^{(l)} \mathbf{H}^{(l-1)}_{(:,v)} + \mathbf{C}^{(l)}) \quad \forall\, l \in \{1, \ldots, L\} \quad \forall v \in \mathcal{V} \text{ and} \\
\mathbf{H}^{(0)}_{(:,v)} := \mathbf{X}_{(:,v)} \quad \forall v \in \mathcal{V},
\end{aligned}$$

where $\mathbf{B}^{(l)}$ is trainable weights and $\mathbf{C}$ is the bias term for layer l. The adjacency matrix of the graph is considered as the input $\mathbf{X}$ of the model, $\mathbf{H}^{(L)}_{(:,v)}$ indicates the final embedding vector of node v and $\hat{\mathbf{X}}$ represents the reconstructed data. SDAE [CLX16] is another deep neural network architectures extracting representations from the *PPMI* matrix by means of a *stacked denoising autoencoders* [Vin+10].

We can also learn representations by encoding the network structure and node features together. The intuition relies on the message-passing framework in which each node aggregates information from its local neighborhood or from a determined set of nodes. Graph Convolutional Network (GCN) [KW17] proposes a multi-layer convolutional architecture with the following propagation rule:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-0.5}\tilde{\mathbf{W}}\tilde{\mathbf{D}}^{-0.5}\mathbf{H}^{(l)}\mathbf{B}^{(l)}\right) \quad \text{and} \quad \mathbf{H}^{(0)} = \mathbf{X},$$

where $\widetilde{\mathbf{W}} = \mathbf{W} + \mathbf{I}$ is the adjacency matrix with added self-loops, $\widetilde{\mathbf{D}}$ is the diagonal matrix for $\widetilde{\mathbf{D}}_{(v,v)} = \sum_{w \in \mathcal{V}} \tilde{\mathbf{W}}_{(v,w)}$ and $\mathbf{X}$ is a matrix of node features. Similarly, $\mathbf{B}^{(l)}$ is trainable weights for layer l and $\sigma(\cdot)$ indicates the activation function such as $ReLU(\cdot) = \max(0, \cdot)$. It is a semi-supervised approach extracting the embeddings by encoding both graph structure and node features. GraphSAGE [HYL17a] extends the former method for inductive learning settings. It can efficiently generate node embeddings for previously unseen nodes. The random walk-based approaches [GF18] can also be stated as naive examples since they apply the shallow, two-layer neural networks for the optimization step similar to SkipGram [Mik+13b] model. A plethora of graph neural network models have recently been proposed [KBG19; Boj+20; JB20]. The interested reader could refer to [HYL17b; Ham20].

2.3.5 *Large Scale Embedding Approaches*

The continuously growing size of networks has opened up a new direction towards fast and accurate approaches. It is a promising and ascending field since the high computational and resource requirements of many existing techniques make them inapplicable for large scale networks consisting of millions of nodes and links.

The random walk methods can also be included in this category since they aim to approximate the proximity measures between nodes by generating node sequences. The LINE [Tan+15] method optimizes an objective function that captures both first-order and second-order proximity of each node. An advantage of it is that it can be applied to weighted networks as

well. However, the main limitation of these methods is that they do not scale well for large networks. They have focused on increasing the effectiveness of data mining tasks (e.g., classification, link prediction, network reconstruction), whereas the efficiency side has not received significant concern. To address this problem, recent advances in graph representation learning use random projection or hashing techniques (more specifically, variants of locality-sensitive hashing [PJA10]) to boost performance, trying to maintain effectiveness.

One of the first scalable approaches, RANDNE [Zha+18], is based on iterative Gaussian random projections, adapting to any desired proximity level. The orthogonal projection of the weighted sum of the powers of the adjacency matrix $\mathbf{W}$ gives the final representations $\mathbf{E}$ as shown in Equation (2.3). Each term is recursively defined by $\mathbf{E}_l := \mathbf{W} \cdot \mathbf{E}_{l-1}$ for each $l > 0$ and the initial term $\mathbf{E}_0$ is chosen as the orthogonal projection matrix:

$$\mathbf{E} = \alpha_0 \mathbf{E}_0 + \alpha_1 \mathbf{E}_1 + \cdots + \alpha_{L-1} \mathbf{E}_{L-1} + \alpha_L \mathbf{E}_{\mathcal{L}}, \quad \alpha_l \in \mathbb{R}, \quad \forall l \in \{0, \ldots, \mathcal{L}\}. \tag{2.3}$$

In the same line, FASTRP [Che+19] uses sparse Gaussian random projections. It applies the same general formulation except it omits the initial term $\mathbf{E}_0$.

Recently, embedding techniques rely on hashing have emerged as a promising alternative to enable faster processing while, at the same time, retaining good accuracy results. The NETHASH [Wu+18] algorithm expands each node of the graph into a rooted tree up to a predetermined depth, and then by using a bottom-up approach, encodes structural information as well as attribute values into minhash [Bro97] signatures in a recursive manner. A similar approach has been used in NODESKETCH [Yan+19], which applies a recursive sketching process. It learns integer-valued embeddings in Hamming space in which the pairwise distances approximate the weighted Jaccard similarity [Li15] among the initially designed node vectors.

2.3.6 *Other Variants*

In the literature, there are numerous graph representation learning approaches proposed for studying and analyzing a different aspect of networks.

We have briefly described a number of notable methods from each distinct category, but there are still lots of exciting works deserving to be mentioned. In this part, we name a couple of methods which cannot be exactly classified as one of the types mentioned above.

A graph coarsening framework, HARP [Che+18], firstly extracts embeddings by running a method such as DeepWalk and LINE on the compressed network, and then node representations are further prolonged for the initial network. It essentially focuses on improving the scalability of representation learning algorithms. Another approach Struct2Vec [RSF17] aims to capture the structural identity of nodes in learning node representations and considers a hierarchical metric to measure similarities.

2.4 DESCRIPTION OF DATASETS

In this section, we describe the networks which are commonly used in most of the experiments throughout the dissertation. We examine the performance of algorithms on seven networks of different types and sizes. To be consistent in the experiments, we consider each network as undirected and unweighted.

- *CiteSeer* [Che+18] is a citation network obtained from the *CiteSeer* library, in which each node corresponds to a paper, and the edges indicate reference relationships among papers. The node labels represent the subjects of the paper.

- *Cora* [Sen+08] is another citation network constructed from the publications in the machine learning area; the documents are classified into seven categories.

- *DBLP* [Per+17] is a co-authorship graph, where an edge exists between nodes if two authors have co-authored at least one paper. The labels represent the research areas.

- *PPI* [GL16] is a graph extracted from the protein-protein interaction network for Homo Sapiens in which biological states are used as labels of nodes.

	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{K}\rvert$	$\lvert\mathcal{C}\rvert$	**Avg. Degree**	**Type**
CiteSeer	3,312	4,660	6	438	2.814	Citation
Cora	2,708	5,278	7	78	3.898	Citation
DBLP	27,199	66,832	4	2,115	4.914	Collaboration
PPI	3,890	38,739	50	35	19.917	Biological
AstroPh	17,903	19,7031	-	1	22.010	Collaboration
HepTh	8,638	24,827	-	1	5.7483	Collaboration
Facebook	4,039	88,234	-	1	43.6910	Social network
Gnutella	8,104	26,008	-	1	6.4186	Peer-to-peer

Table 2.1: Statistics of networks $|\mathcal{V}|$: number of nodes, $|\mathcal{E}|$: number of edges, $|\mathcal{K}|$: number of labels and $|\mathcal{C}|$: number of connected components.

- *AstroPh* [LKF07] is a collaboration network built from the papers submitted to the *ArXiv* repository for the *Astro Physics* subject area, from January 1993 to April 2003.
- *HepTh* [LKF07] is constructed in a similar way from the papers submitted to *ArXiv* for the *High Energy Physics - Theory* category.
- *Facebook* [LM12] is a social network extracted from a survey conducted via a *Facebook* application.
- *Gnutella* [RIF02] is the peer-to-peer file-sharing network constructed from the snapshot collected in August 2002 in which nodes and edges correspond to hosts and connections among them, respectively.

The detailed statistics of the datasets are provided in Table 2.1. The networks in the first group have node labels so they are used in the node classification experiments and the others are only used in the link prediction experiments.

2.5 MACHINE LEARNING TASKS AND EVALUATION METRICS

We mainly perform two types of experiments to evaluate and compare our approaches to the baseline methods. We have implemented the experimental setup in Python with the *scikit-learn* [Ped+11] package.

Operator	Symbol	Definition
Average	$\boxdot$	$\left(\mathbf{E}[v]_j + \mathbf{E}[u]_j\right)/2$
Hadamard	$\odot$	$\mathbf{E}[v]_j \times \mathbf{E}[u]_j$
Weighted L1	$\|\cdot\|_1$	$\left\|\mathbf{E}[v]_j - \mathbf{E}[u]_j\right\|$
Weighted L2	$\|\cdot\|_2$	$\left\|\mathbf{E}[v]_j - \mathbf{E}[u]_j\right\|^2$

Table 2.2: Binary operators for constructing edge feature vectors. Each definition corresponds to j-th component of node representations $\mathbf{E}[v]$ and $\mathbf{E}[v]$.

2.5.1 *Node Classification*

For the node classification task, we have access to the labels of a certain fraction of nodes in the network (training set), and our goal is to predict the labels of the remaining nodes (test set). After learning the representation vectors for each node, we split them into varying sizes of training and testing sets. Unless otherwise specified, the experiments are carried out by applying a one-vs-rest logistic regression classifier with L_2 regularization. In order to provide more reliable experimental results, the same procedure is repeated 50 times. We report the average performance in terms of both micro-averaged F_1 and macro-averaged F_1 scores [Vic79], which are defined by

$$\text{Micro-}F_1 = \frac{2PR}{P+R} \quad \text{and} \quad \text{Macro-}F_1 = \left(\frac{2P_kR_k}{P_k+R_k}\right)$$

$$P := \frac{\sum_{k=1}^{\mathcal{K}} tp_k}{\sum_{k=1}^{\mathcal{K}} tp_k + fp_k} \quad R := \frac{\sum_{k=1}^{\mathcal{K}} tp_k}{\sum_{k=1}^{\mathcal{K}} tp_k + fn_k} \quad P_k := \frac{tp_k}{tp_k + fp_k} \quad R_k := \frac{tp_k}{tp_k + fn_k},$$

where $\mathcal{K}$ is the number of classes, tp_k is the true positive, fn_k is the false negative and fp_k is the false positive results for the corresponding class k.

2.5.2 *Link Prediction*

In the link prediction task, we have limited access to the edges of the network, and our goal is to predict the missing (unseen) edges between nodes. We divide the edge set of a given network into two parts to form training and test

sets by randomly removing 50% of the edges (the network remains connected during the process). The removed edges are later used as positive samples in the test set. The same number of node pairs that do not exist in the initial network is sampled to generate negative instances for each training and test sets. The node embedding vectors $\mathbf{E}[v]$ and $\mathbf{E}[u]$ are converted into edge feature vectors by applying the coordinate-wise operations given in Table 2.2, as proposed by [GL16]. We perform experiments using the logistic regression classifier with L_2 regularization. We report the Area Under Curve (AUC) [Alp10] score of the operators showing the best performance for each method.

3

TOPIC-AWARE LATENT MODELS FOR REPRESENTATION LEARNING ON GRAPHS

RANDOM walk-based graph representation learning methods have received particular attention over the last years thanks to their success in several graph analysis problems, including node classification, link prediction, and clustering. They transform the network into a collection of node sequences, aiming to learn node representations by predicting the context of each node within the sequence. In this chapter, we introduce TNE, a generic framework to enhance the embeddings of nodes acquired by means of random walk-based approaches with topic-based information. Similar to the concept of topical word embeddings in Natural Language Processing, the proposed model first assigns each node to a latent community with the favor of various statistical graph models and community detection methods and then learns the enhanced topic-aware representations. The experimental results demonstrate that by incorporating node and community embeddings, TNE outperforms widely-known baseline GRL models.

3.1 INTRODUCTION

Initial studies in the field of graph representation learning have been inspired by the advancements in the area of Natural Language Processing (NLP), borrowing various ideas originally developed for computing *word embeddings*. A prominent example here is the SKIPGRAM architecture [Mik+13b], which aims to find latent representations of words by estimating their context within the sentences of a textual corpus. As we have discussed in Chapter 2, many pioneer studies in GRL [PARS14; GL16; NM18] utilize the idea of random walks to transform graphs into a collection of sentences – as an analogy to the area of natural language – and these sentences or walks are later being used to learn node embeddings.

Although random walk-based approaches are strong enough to capture local connectivity patterns, they mainly suffer to sufficiently convey information about more global structural properties. More precisely, real-world networks have an inherent clustering (or community) structure, which can be utilized to further improve the predictive capabilities of node embeddings. One can interpret such structural information based on an analogy to the concept of *topics* in a collection of documents. In a similar way as word embeddings can be enhanced with topic-based information [Liu+15], here we aim at empowering node embeddings by employing information about the latent community structure of the network—that can be achieved by a process similar to the one of *topic modeling*.

In this chapter, we propose *Topical Node Embeddings* (TNE), a framework in which node embeddings are enhanced with topic (or community) information towards learning topic-aware node representations—something that leads to further improvements in the performance on downstream tasks. Local clustering patterns are of great importance on many applications, enabling to grasp hidden information of the network properly. For instance, consider two individuals sharing common friends or interests, that are not yet represented by a direct link in the network. A careful analysis of the local community structure can further help to infer the missing information (e.g., missing links), and therefore boost the predictive capabilities of node embeddings. Motivated by that, the proposed TNE framework aims to directly leverage the latent community structure of the graph while learning node embedding vectors. The main contributions of the chapter can be summarized as follows:

- *Latent graph models and topic representations*. We show how existing latent space discovery models, such as community detection and topic models, can be incorporated into the node representation learning process.

- *Node representation learning framework*. We propose a new model, called TNE, which first learns community embeddings from the graph and then uses them to improve the node representations extracted by ran-

dom walk-based methods. We examine various instances of this model, studying their properties.

- *Enriched feature vectors*. We perform a detailed empirical evaluation of the embeddings learned by TNE on the tasks of node classification and link prediction. As the experimental results indicate, the proposed model learns feature vectors which can boost the performance on downstream tasks.

The rest of the chapter is organized as follows. Section 3.2 describes several related works, and gives the fundamental concepts by formulating the problem. In Section 3.3, we introduce the concept of topical node representation learning. The proposed TNE model is described in Section 3.4. Section 3.5 presents the experimental results, and finally, we conclude our work in Section 3.6.

SOURCE CODE. Our model has been implemented in Python and the source code can be found at: https://abdcelikkanat.github.io/projects/TNE/.

3.2 BACKGROUND CONCEPTS AND RELATED WORK

In this section, we will first briefly review the objective of the random walk-based approaches. We will also provide an overview of the community detection problem, focusing on aspects that are useful for the presentation of the proposed methodology.

3.2.1 *Learning Node Embeddings with Random Walks*

Let $G = (\mathcal{V}, \mathcal{E})$ be a graph with the vertex set $\mathcal{V}$ and the edge set $\mathcal{E}$. As we have discussed in Chapter 2, our goal is to find a mapping $\mathbf{B} : \mathcal{V} \rightarrow \mathbb{R}^d$, where $\mathbf{B}[v]$ will correspond to the representation of node v in a lower-dimensional space $\mathbb{R}^d$; d is referred to as the embedding or dimension size, and is much smaller than the cardinality of the vertex set. As we have mentioned previously, graph representation learning methods based on the popular SkipGram architecture (e.g., [Mik+13a; Mik+13b]) learn node

representations using node sequences produced by random walks over a given network. They mainly target to maximize the log-likelihood of the occurrences of nodes within a certain distance with respect to each other, as follows:

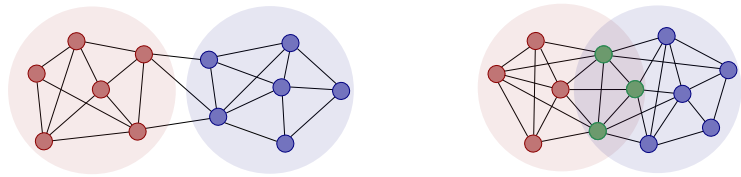
$$\mathcal{F}_N(\mathbf{A}, \mathbf{B}) := \underset{\Omega}{\operatorname{argmax}} \sum_{\mathbf{w} \in \mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{-\gamma \leq j \leq \gamma \\ j \neq 0}} \log Pr(v_{l+j} \mid v_l;\ \Omega), \tag{3.1}$$

where $\Omega = (\mathbf{A}, \mathbf{B})$ is the model parameters that we would like to learn, $\mathcal{W}$ is the set of random walks $\mathbf{w} = (v_1, \ldots, v_l, \ldots, v_{\mathcal{L}}) \in \mathcal{V}^{\mathcal{L}}$ of length $\mathcal{L}$, and γ refers to the window size. A common choice for the probability measure in Equation (3.1) is the softmax or sigmoid function. (Please refer to Subsection 2.3.3 in Chapter 2 for a detailed explanation.) Note that, the nodes appearing inside the window size of *center* node v_l are referred to as *context* nodes. We obtain two different representation vectors $\mathbf{A}[v]$ and $\mathbf{B}[v]$ for each node $v \in \mathcal{V}$ but we will consider only $\mathbf{B}[v]$ in the experimental evaluation, which corresponds to the vector when the node is interpreted as a center node.

3.2.2 *Community Structure*

The integration of community information in learning embedding can be beneficial since it is one of the key elements of complex networks making them distinct from arbitrary graph structures. In general, a set of nodes having dense connections than the rest of the network is referred to as a community, but the formal and precise definition might vary depending on the context in which it is introduced. For instance, the notion of *modularity* has been introduced to measure the quality of a division of the network [New06]. Networks with high modularity represent the existence of clusters of nodes having high intra-connections and sparse links between the nodes from different clusters. It is formally defined by

$$modularity(G) := \frac{1}{2vol(G)} \sum_{v,u \in \mathcal{V}} \left[\mathbf{W}_{(u,v)} - \frac{m_v m_u}{vol(G)} \right] \delta(z_v, z_u), \tag{3.2}$$



(a) No overlaps (b) Overlaps

Figure 3.1: Overlapping community structure types in a network.

where $\mathbf{W}_{(v,u)}$ indicates the edge weight between nodes u and v, $vol(G)$ is $1/2\sum_{v,u\in\mathcal{V}}\mathbf{W}_{(v,u)}$, m_v is defined by $\sum_{u\in\mathcal{V}}\mathbf{W}_{(v,u)}$, and δ_{z_v,z_u} is Kronecker delta function, which is equal to 1 if nodes v and u are assigned to the same community (i.e., $z_u = z_v$), while $\delta(z_v, z_u) = 0$ otherwise.

However, the optimization of modularity is computationally intractable, so Blondel et al. [Blo+08] has proposed a greedy algorithm, namely LOUVAIN, to address this issue. The algorithm is initialized by assigning a distinct community label to each node of the given network, so the number of communities is proportional to the number of nodes at the beginning of the procedure. Then, each node is assigned to the community label of its neighbor producing the highest marginal gain, or it preserves its current label if there is no more gain for any of its neighbors. The process is repeated until no further improvement is feasible.

Real-world networks might also have an overlapping community structure so that a node might be a member of multiple communities as shown in Figure 3.1. Many methods have been proposed to deal with the overlapping community detection problem [MV13; LFK09]. For instance, the BIGCLAM [YL13] approach models the latent interaction strength, $\mathbf{X}^{(k)}_{(v,u)}$, between nodes v and u in a community $k \in \{1, \ldots, \mathcal{K}\}$ by a Poisson distribution with mean defined by $\mathbf{F}_{(k,v)} \cdot \mathbf{F}_{(k,u)}$ where each entry $\mathbf{F}_{(k,v)}$ of the vector indicates a non-negative weight and $\mathcal{K}$ is the number of communities. Therefore, the total amount of interaction strength between nodes v and u is

$$\mathbf{X}_{(v,u)} := \sum_{k=1}^{\mathcal{K}} \mathbf{X}^{(k)}_{(v,u)} \quad \text{where} \quad \mathbf{X}^{(k)}_{(v,u)} \sim Pois(\mathbf{F}_{(k,v)} \cdot \mathbf{F}_{(k,u)}).$$

In other words, if a pair of nodes shares many common communities, the total amount of latent interactions, $\mathbf{X}_{(v,u)}$, between them becomes high, and it leads to high edge probability defined by $Pr(\mathbf{X}_{v,u} > 0)$. More formally and compactly, the approach optimizes the following log-likelihood function:

$$\underset{\mathbf{F}}{\operatorname{argmax}} \sum_{(v,u)\in\mathcal{E}} \log(Pr(\mathbf{X}_{(v,u)} > 0)) + \sum_{(v,u)\notin\mathcal{E}} \log(Pr(\mathbf{X}_{(v,u)} = 0)) \tag{3.3}$$

$$= \underset{\mathbf{F}}{\operatorname{argmax}} \sum_{(v,u)\in\mathcal{E}} \log(1 - \exp(-\mathbf{F}_{(:,v)}^{\top} \cdot \mathbf{F}_{(:,u)})) - \sum_{(v,u)\notin\mathcal{E}} \mathbf{F}_{(:,v)}^{\top} \cdot \mathbf{F}_{(:,u)}. \tag{3.4}$$

After learning the affiliation factor, $\mathbf{F}_{(v,k)}$, the BigClam model assigns node v to community k if $\mathbf{F}_{(v,k)}$ exceeds a certain threshold value.

In the following sections, we will present the proposed TNE model, which independently learns node and community (topic) embeddings, and then combines them to obtain expressive topical representations. We extract the networks' latent community structure by using the community detection methods described above or various statistical models relying on random walks. We will then leverage the extracted community information into the learning procedure of community representations. To the best of our knowledge, very few models benefit from the community structure of real networks while learning embeddings. The ComE model [Cav+17] proposes a closed-loop procedure among the encoding of communities, learning node embeddings and community detection in the network. M-NMF [Wan+17] targets to learn node representations by incorporating community structure information in a non-negative matrix factorization formulation. COSINE [ZLZ18] is a generative model learning the social network embeddings from information diffusion cascades. A recent approach, GEMSEC [Roz+19], learns node embeddings with an explicitly defined community preserving objective function.

3.3 LEARNING TOPIC REPRESENTATIONS

As we have mentioned above, complex networks, such as those arising from social or biological settings, consist of latent clusters of different sizes in which the nodes are more likely to be connected to each other [GN02; LFK09;

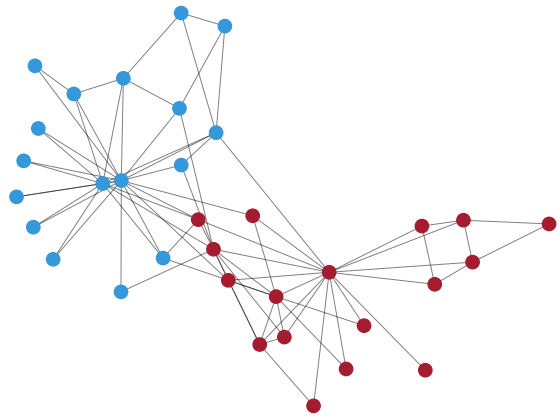


Figure 3.2: The topic-community assignments in Zachary's *Karate Club* network. Each node v is assigned to a community label z, maximizing the posterior probability $Pr(z|v)$ by using the TNE-GLDA model.

MV13; HER09; Hen+]. Our main goal here is to use the latent clusters of a network in order to obtain enriched representations. This can be achieved by enhancing node embedding vectors with *topic representations*. By replacing a node v_l with its community label z_l in a random walk, we learn *community embeddings* by predicting the nodes in the context of a community label. More formally, we can define our objective function to learn topic representations as follows:

$$\mathcal{F}_T(\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}}) := \underset{\widetilde{\Omega}=(\widetilde{\mathbf{A}},\widetilde{\mathbf{B}})}{\operatorname{argmax}} \frac{1}{N \cdot \mathcal{L}} \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{-\gamma\le j\le\gamma \\ j\neq 0}} \log Pr(v_{l+j} \mid z_l;\, \widetilde{\Omega}). \tag{3.5}$$

By maximizing the log-probability above, we obtain the embedding vectors corresponding to each community label $z_l \in \{1, \ldots, \mathcal{K}\}$. In this work, we mainly use two approaches to detect latent communities. The first one is based on novel combination of generative statistical models accompanied with random walks, while the second one is based on traditional community detection algorithms that utilize the network structure itself.

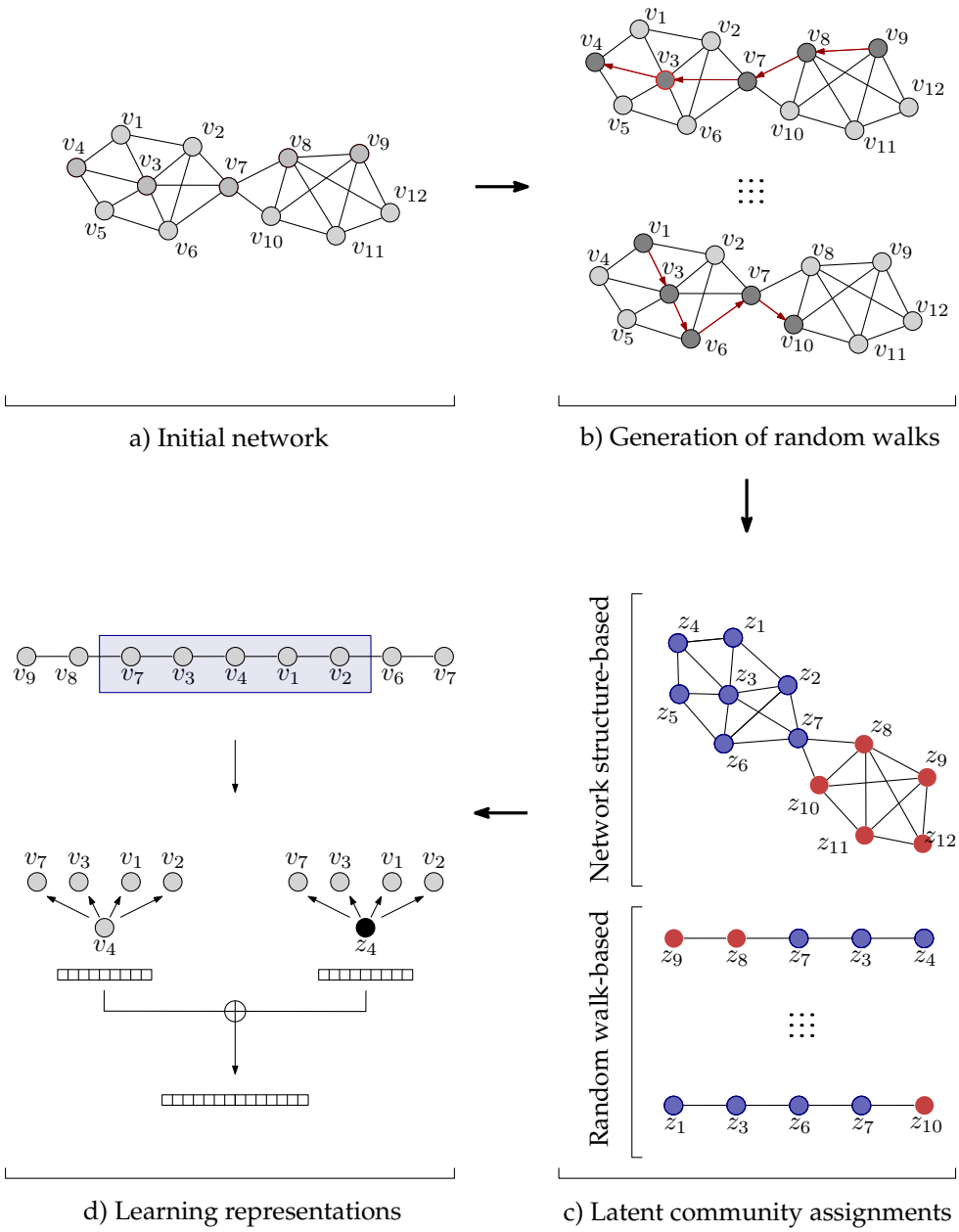
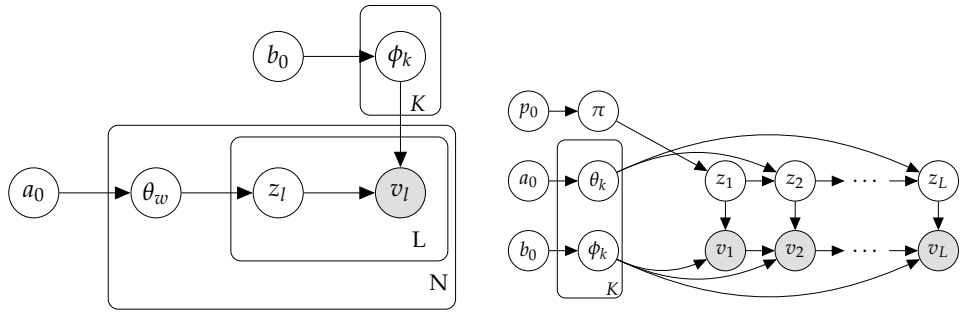


Figure 3.3: Schematic representation of the TNE model. The final representations are learned by combining node and topic embedding vectors. The representation of a node is learned using random walks performed over the network; its topic representation is similarly learned by assigning a topic/community label based either on random walks (TNE-GLDA, TNE-GHMM) or network structure-based approaches (TNE-LOUVAIN, TNE-BIGCLAM).



(a) GLDA model (b) GHMM model

Figure 3.4: Plate representations of random walk-based topic representation models.

3.3.1 *Random Walks and Generative Graph Models*

Most real-world networks can be expressed as a combination of nested or overlapping communities [Pal+05]. Therefore, when a random walk is initialized, it does not only visit neighboring nodes but also traverses communities in the network (see Figure 3.3b). In that regard, we assume that each random walk can be represented as random mixtures over latent communities, and each community can be characterized by a distribution over nodes. In other words, we can write the following generative model for each walk over the network:

1. For each $k \in \{1, \ldots, \mathcal{K}\}$
   - $\phi_k \sim Dir(b_0)$
2. For each walk $\mathbf{w} = (v_1, \ldots, v_l, \ldots, v_{\mathcal{L}})$
   - $\theta_{\mathbf{w}} \sim Dir(a_0)$
   - For each vertex $v_l \in \mathbf{w}$
     - $z_l \sim Multinomial(\theta_{\mathbf{w}})$
     - $v_l \sim Multinomial(\phi_{z_l})$

Here, N indicates the number of walks, $a_0, b_0 \in \mathbb{R}$ are the hyperparameters of the symmetric Dirichlet prior distributions, and the vectors ϕ_k and $\theta_{\mathbf{w}}$ contain $|\mathcal{V}|$ and $\mathcal{K}$ components, respectively. If we consider each random

walk as a document and the collection of random walks as a corpus, it can be seen that the statistical process defined above corresponds to the well known Latent Dirichlet Allocation (LDA) model [BNJ03]. Therefore, each community corresponds to a distinct topic in the terminology of NLP (we use the terms *topic* and *community* interchangeably in the rest of the chapter). We will refer to this model as GLDA (the plate representation is shown in Figure 3.4a). As we show in Lemma 3.1, the relative frequency of the occurrences of a node in the generated walks is proportional to its degree in the network for large number of walks or walk lengths. This property was first empirically demonstrated in the work of Perozzi, Al-Rfou, and Skiena [PARS14]. Here, we provide a formal argument of this empirical observation.

Lemma 3.1. *Let $G = (\mathcal{V}, \mathcal{E})$ be a connected graph, $\{X_l\}_{l \geq 1}$ be a Markov chain with state space $\mathcal{V}$ and $\mathbf{P}$ be a transition matrix, where $\mathbf{P}_{(v,u)}$ is defined as $1/deg(v)$ for each edge $(v,u) \in \mathcal{E}$ and 0 otherwise. If the Markov chain is aperiodic, then*

$$\lim_{\mathcal{L} \to \infty} \frac{1}{\mathcal{L}} \mathbb{E}\left[O_{\mathcal{L}}^{(v)}\right] = \frac{deg(v)}{2|\mathcal{E}|},$$

where $O_{\mathcal{L}}^{(v)}$ is a random variable representing the number of occurrences of the node v in a random walk of length $\mathcal{L}$.

Proof. Since the graph is connected, each state can be accessed by any other one. Thus, the Markov chain is also irreducible having a unique limiting distribution π. Note that π_v is equal to $deg(v) / \sum_{u \in \mathcal{V}} deg(u)$ since it satisfies $\pi \mathbf{P} = \pi$. Then, we can write:

$$\begin{aligned}
\lim_{\mathcal{L} \to \infty} \frac{1}{\mathcal{L}} \mathbb{E}\left[O_{\mathcal{L}}^{(v)}\right] &= \lim_{\mathcal{L} \to \infty} \frac{1}{\mathcal{L}} \mathbb{E}\left[\sum_{l=1}^{\mathcal{L}} \mathbb{1}_{\{X_l = v\}}\right] = \lim_{\mathcal{L} \to \infty} \frac{1}{\mathcal{L}} \sum_{l=1}^{\mathcal{L}} \mathbb{E}\left[\mathbb{1}_{\{X_l = v\}}\right] \\
&= \lim_{\mathcal{L} \to \infty} \frac{1}{\mathcal{L}} \sum_{l=1}^{\mathcal{L}} Pr\left(X_l = v\right) = \pi_v = \frac{deg(v)}{\sum_{u \in \mathcal{V}} deg(u)} \\
&= \frac{deg(v)}{2|\mathcal{E}|},
\end{aligned}$$

where the equality in the second line follows from *Cesàro Theorem* [DeV07], since $Pr(X_l = v)$ converges to π_v as $\mathcal{L}$ goes to infinity. □

In the previous GLDA model, the latent community assignment of each node is independently chosen from the community label of the previous node in the random walk. However, the hidden state of the current node can play an important role towards determining the next vertex to visit, as the random walk also traverses through communities. Therefore, we can write the following generative process, by modifying the GLDA model:

1. For each $k \in \{1, \ldots, \mathcal{K}\}$
   - $\phi_k \sim Dir(b_0)$
   - $\theta_k \sim Dir(a_0)$
2. $\pi \sim Dir(p_0)$
3. For each walk $\mathbf{w} = (v_1, \ldots, v_i, \ldots, v_L)$
   - $z_1 \sim Dir(\pi)$
   - For each vertex $v_l \in \mathbf{w}$, for all $2 \leq l < L$
     - $v_l \sim Multinomial(\phi_{z_l})$
     - $z_{l+1} \sim Multinomial(\theta_{z_l})$
   - $v_{\mathcal{L}} \sim Multinomial(\phi_{z_{\mathcal{L}}})$

Here, $a_0, b_0, p_0 \in \mathbb{R}$ are the hyperparameters of the Dirichlet distributions, and the vectors θ_k and ϕ_k contain $\mathcal{K}$ and $|\mathcal{V}|$ components, respectively. The above model, in fact, corresponds to the well-known Hidden Markov Model (HMM) with symmetric Dirichlet priors over transition and emission distributions. In our experiments, we adopt the Infinite Hidden Markov Model (IHMM) [VG+08] (we will refer to this model as GHMM; the plate representation is shown in Figure 3.4b). Note that, unlike the GLDA model, in the generation of each node sequence the same transition probabilities are used.

3.3.2 *Network Structure-Based Modeling*

In the previous models, the generated random walks are used to detect the community (or topic) assignment of each node in the given node sequence.

Here, we utilize two additional community detection models described in Section 3.2, which directly target to extract communities of nodes from a given network. The first one corresponds to the well-known LOUVAIN algorithm by [Blo+08] that extracts communities based on modularity maximization, while the second one to the BIGCLAM model for overlapping community detection [YL13].

3.4 TOPICAL NODE EMBEDDINGS

In this section, we will describe the proposed Topical Node Embedding (TNE) model for learning topic-aware node representations. An overview of the model is given in Figure 3.3. TNE aims to enhance node embeddings using information about the underlying topics of the graph obtained by the models described in Section 3.3. This can be achieved by learning node and topic embedding vectors independently of each other, jointly maximizing the objectives defined in Equations (3.1) and (3.5). Combining these two objectives, we derive the following:

$$\underset{\Omega,\widetilde{\Omega}}{\operatorname{argmax}} \frac{1}{N \cdot \mathcal{L}} \sum_{\mathbf{w} \in \mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{-\gamma \leq j \leq \gamma \\ j \neq 0}} \left[\underbrace{\log Pr\left(v_{l+j} | v_l; \Omega\right)}_{\text{node embedding}} + \underbrace{\log Pr\left(v_{l+j} | z_l; \widetilde{\Omega}\right)}_{\text{community embedding}} \right].$$

In our approach, we use the *sigmoid* function for the probability measure in the above equation and we adopt the negative sampling strategy [Mik+13b], in order to make our computations more efficient. After obtaining the node and topic representations, our final step is to efficiently incorporate these two feature vectors, $\mathbf{B}[v]$ and $\widetilde{\mathbf{B}}[z]$ of node v and community label z respectively, so as to obtain the final topic-enhanced node embedding. For this purpose, we concatenate node embedding vector $\mathbf{B}[v]$ with the expected topic vector with respect to the distribution $Pr(\cdot|v)$. Our strategy can be formulated as follows:

$$\mathbf{B}[v] \oplus \sum_{k \in \{1,\ldots,\mathcal{K}\}} Pr(k|v) \cdot \widetilde{\mathbf{B}}[k], \tag{3.6}$$

Algorithm 3.1 Topical Node Embeddings (TNE)

Input: Graph: $G = (\mathcal{V}, \mathcal{E})$
Number of walks: $\mathcal{N}$
Walk length: $\mathcal{L}$
Window size: γ
Number of communities: $\mathcal{K}$
Topic representation learning method: T
Node embedding size: d_n
Community embedding size: d_t
Output: Embedding vectors of length: $d_n + d_t$
1: Perform N random walks of length $\mathcal{L}$ for each node
2: Learn node representations by optimizing Equation (3.1)
3: Learn topic representations by optimizing Equation (3.5), using any of the models T of Section 3.3
4: Concatenate node and topic embeddings with Equation (3.6)

where $\oplus$ indicates the concatenation operation. We refer to the final vector obtained after concatenating the node and topic feature vectors as *topical node embedding*.

Algorithm 3.1 provides the pseudocode of the proposed TNE model. The general structure of our framework as follows. First, we need a collection of walks over the network to learn node and topic embeddings. We further produce node-context pairs and use SkipGram to learn node embeddings following Equation (3.1). Then, we choose a strategy (shown as T in Algorithm 3.1) to learn topic representations. This step is quite flexible in the formulation of TNE. One approach is to generate topic assignments z_l of each node $v_l \in \mathcal{V}$ in the walk $\mathbf{w} \in \mathcal{W}$, based on the random walk-based generative models GLDA and GHMM, defined in Section 3.3.1. Alternatively, we can directly infer the latent clustering structure based on BIGCLAM and LOUVAIN models, as described in Section 3.3.2. We learn topic representations by replacing the center node with its corresponding community label through Equation 3.5. Lastly, we combine node and topic embeddings using Equation (3.6) to obtain the final topical node embedding vectors. Depending on the method used to learn topical representations, we will refer to the corresponding instances of TNE as TNE-GLDA, TNE-GHMM, TNE-LOUVAIN and TNE-BIGCLAM.

3.4.1 *Complexity Analysis*

The time complexity of TNE varies depending on the algorithm used to detect latent topics and communities. For a given node sequences and topic assignments, the representations can be learned in the order of $\mathcal{O}(d \cdot \gamma \cdot k \cdot |\mathcal{W}| \cdot \mathcal{L})$ with the *negative sampling* technique for k sampled instances because we have $|\mathcal{W}|$ number of walks and there are $2 \cdot \gamma \cdot (k+1) \cdot \mathcal{L}$ center-context pairs for each walk. If the LOUVAIN algorithm is chosen, the communities can be detected in $\mathcal{O}(|\mathcal{V}| \cdot \log^2(|\mathcal{V}|))$ operations, while $\mathcal{O}(|V| \cdot \mathcal{K})$ steps are required for BIGCLAM when the number of communities is high. The models that rely on random walks and generative models (GLDA, GHMM), can be run in $\mathcal{O}(\mathcal{W} \cdot \mathcal{L} \cdot \mathcal{K} \cdot I)$, where I is the number of iterations [Por+08; VG+08].

3.5 EXPERIMENTAL EVALUATION

In this section, we present the experimental setup and parameter settings for TNE and for the baseline models. The performance of the proposed model is examined in two downstream tasks: node classification and link prediction. We use eight different networks in our experiments as described in Section 2.4 and the statistics of the networks can be found in Table 2.1.

3.5.1 *Baseline Methods*

We consider seven baseline methods to compare the performance of our approach. We have described most of them in Section 2, but we also provide an overview here, including parameter settings:

(i) DEEPWALK [PARS14] performs uniform random walks to generate the set of node sequences; then, the SKIPGRAM model [Mik+13a; Mik+13b] is used to learn node representations.

(ii) NODE2VEC [GL16] combines SKIPGRAM with biased random walks, using two extra parameters that control the walk in order to simulate a BFS or DFS exploration. In the experiments, we set those parameters to

1.0. As we have already mentioned, in our approach we sample context nodes using this biased random walk strategy.

(iii) LINE [Tan+15] learns nodes embeddings relying on first- and second-order proximity information of nodes.

(iv) HOPE [Ou+16] is a matrix factorization approach aiming at capturing similarity patterns based on a higher-order node similarity measure. In the experiments, we consider the *Katz* index, since it demonstrates the best performance among other proximity indices.

(v) NETMF [Qiu+18] targets to factorize the matrix approximated by the pointwise mutual information of center and context pairs. The experiments have been conducted for large window sizes ($\gamma = 10$) with a rank size of 256 due to its good performance.

(vi) GEMSEC [Roz+19] leverages the community structure of real-world graphs, learning node embeddings and the cluster assignments simultaneously. We have performed parameter tuning to set the number of communities from $\{5, 15, 20, 25, 50, 75, 100\}$.

(vii) Lastly, M-NMF [Wan+17] extracts node embeddings under a modularity-based community detection framework based on non-negative matrix factorization. We have observed that the algorithm poses good performance by setting its parameters $\alpha = 0.1$ and $\beta = 5$. We have performed parameter tuning for the number of communities using values from the following set: $\{5, 15, 20, 25, 50, 75, 100\}$.

3.5.2 *Parameter Settings*

The instances of TNE are fed with random walks generated by the NODE2VEC model. Throughout the thesis, we apply the same parameter values for the random walk-based approaches, where the number of walks per node is set to 80, the walk length to 10, and the window size to 10. Moreover, we sample 5 negative instances for each center-context node pair. To speed up the training process, we adopt the negative sampling [Mik+13b] strategy.

Stochastic Gradient Descent (SGD) has been used for optimization, setting the initial learning rate to 0.0025 and its minimum value to 10^{-5}. We learn topic and node embedding vectors of sizes 32 and 96, respectively, so as the obtain feature vectors of length 128.

In the learning process of topic assignments of nodes with the TNE-GLDA model, we perform collapsed Gibbs sampling [GS04] for parameter estimation and for inference. The Monte Carlo Markov Chain (MCMC) approach is used for the parameter estimation of TNE-GHMM, with beam sampling for latent sequence resampling steps [VG+08]. The number of topics for TNE-GLDA is set to 100 for *DBLP*, 125 for *Cora* and set to 150 for the other networks. The initial number of states for TNE-GHMM is chosen as 20.

3.5.3 *Node Classification*

The detailed results for the different instances of TNE framework as well as for the baseline methods are provided in Tables 3.1 to 3.4. Experiments are reported for different ratios of training data (2%–90%). For each model, the first row corresponds to Micro-F_1 scores, while the second one to Macro-F_1 scores.

As we can observe, the TNE-LOUVAIN and TNE-BIGCLAM models perform quite well, outperforming most of the baseline methods for different training ratios. For instance, they have higher Macro-F_1 scores around 4.55% and 6.82% than the best baseline approach for 10% training ratio of the *PPI* network. In the case of the *Cora* network, TNE-GLDA performs better especially for higher training ratios, with 1.33% performance gain in terms of Micro-F_1 score. It is also the best performing approach on the *PPI* network for high training sizes. The overall better performance of the two network structure-based instances TNE-LOUVAIN and TNE-BIGCLAM over random walk-based models (e.g., TNE-GLDA, TNE-GHMM), can possibly be explained by the way that latent communities are extracted. While with TNE-GLDA and TNE-GHMM we are able to utilize random walks for both node and topic representations, it seems that those random walk are not expressive enough to recover the clustering structure as effectively as algorithms tailored to this task, such as BIGCLAM and LOUVAIN.

		2%	4%	6%	8%	10%	30%	50%	70%	90%
Baselines	DeepWalk	0.421	0.462	0.488	0.502	0.517	0.569	0.587	0.596	0.597
		0.374	0.419	0.446	0.461	0.476	0.524	0.540	0.547	0.546
	Node2Vec	0.451	0.491	0.514	0.529	0.543	0.583	0.595	0.600	0.603
		0.397	0.443	0.466	0.483	0.497	0.535	0.546	0.549	0.550
	LINE	0.300	0.353	0.384	0.407	0.418	0.476	0.494	0.505	0.512
		0.243	0.303	0.334	0.357	0.367	0.423	0.440	0.448	0.454
	HOPE	0.197	0.204	0.207	0.208	0.216	0.253	0.276	0.299	0.316
		0.060	0.065	0.066	0.068	0.075	0.121	0.150	0.178	0.202
	NetMF	0.401	0.473	0.507	0.525	0.538	0.579	0.590	0.594	0.601
		0.346	0.421	0.454	0.474	0.487	0.528	0.538	0.542	0.548
	GEMSEC	0.384	0.439	0.459	0.479	0.491	0.532	0.545	0.552	0.558
		0.339	0.400	0.422	0.439	0.451	0.488	0.497	0.499	0.501
	M-NMF	0.264	0.317	0.340	0.370	0.382	0.439	0.449	0.456	0.457
		0.161	0.229	0.261	0.293	0.306	0.370	0.381	0.390	0.390
TNE	GLDA	0.395	0.451	0.493	0.514	0.535	0.593	0.610	0.616	0.618
		0.352	0.411	0.454	0.474	0.494	0.548	0.562	0.568	0.567
	GHMM	0.407	0.474	0.502	0.518	0.526	0.568	0.583	0.583	0.594
		0.360	0.425	0.457	0.473	0.481	0.521	0.534	0.533	0.543
	Louvain	**0.453**	**0.496**	**0.517**	**0.537**	**0.551**	**0.604**	**0.619**	**0.627**	**0.638**
		0.404	**0.452**	**0.476**	**0.493**	**0.506**	**0.560**	**0.576**	**0.582**	**0.594**
	BigClam	0.449	0.488	0.508	0.527	0.546	0.597	0.612	0.619	0.624
		0.402	0.443	0.464	0.483	0.502	0.549	0.562	0.566	0.568

Table 3.1: Node classification for varying training sizes on *Citeseer*. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

		2%	4%	6%	8%	10%	30%	50%	70%	90%
Baselines	DeepWalk	0.617	0.688	0.715	0.732	0.747	0.799	0.815	0.825	0.832
		0.569	0.664	0.698	0.717	0.735	0.788	0.806	0.815	0.821
	Node2Vec	0.659	0.720	0.743	0.759	0.770	0.816	0.831	0.839	0.845
		0.612	0.694	0.724	0.743	0.755	0.804	0.820	0.828	0.832
	LINE	0.416	0.498	0.546	0.581	0.609	0.701	0.728	0.741	0.747
		0.306	0.425	0.492	0.543	0.578	0.691	0.720	0.733	0.738
	HOPE	0.284	0.301	0.301	0.302	0.302	0.302	0.302	0.304	0.306
		0.067	0.066	0.067	0.066	0.066	0.067	0.067	0.068	0.074
	NetMF	0.636	**0.716**	**0.748**	**0.767**	**0.773**	0.821	0.834	0.841	0.844
		0.591	0.694	**0.731**	0.751	0.760	0.811	0.824	0.832	0.835
	GEMSEC	0.470	0.530	0.568	0.587	0.601	0.674	0.714	0.735	0.744
		0.406	0.477	0.527	0.546	0.562	0.646	0.689	0.713	0.722
	M-NMF	0.507	0.580	0.622	0.642	0.656	0.717	0.732	0.736	0.742
		0.459	0.550	0.598	0.622	0.638	0.706	0.722	0.728	0.734
TNE	GLDA	0.612	0.672	0.710	0.731	0.751	0.817	**0.840**	**0.850**	**0.856**
		0.575	0.650	0.694	0.715	0.738	0.807	**0.829**	**0.837**	**0.841**
	GHMM	0.624	0.697	0.734	0.749	0.762	0.809	0.824	0.828	0.834
		0.585	0.672	0.715	0.734	0.746	0.798	0.812	0.816	0.823
	Louvain	**0.679**	**0.721**	0.743	0.756	0.765	0.815	0.835	0.846	0.850
		0.645	**0.699**	0.724	0.738	0.749	0.802	0.822	0.832	0.836
	BigClam	0.631	0.693	0.720	0.740	0.754	0.809	0.828	0.837	0.851
		0.589	0.671	0.702	0.725	0.739	0.798	0.818	0.828	0.838

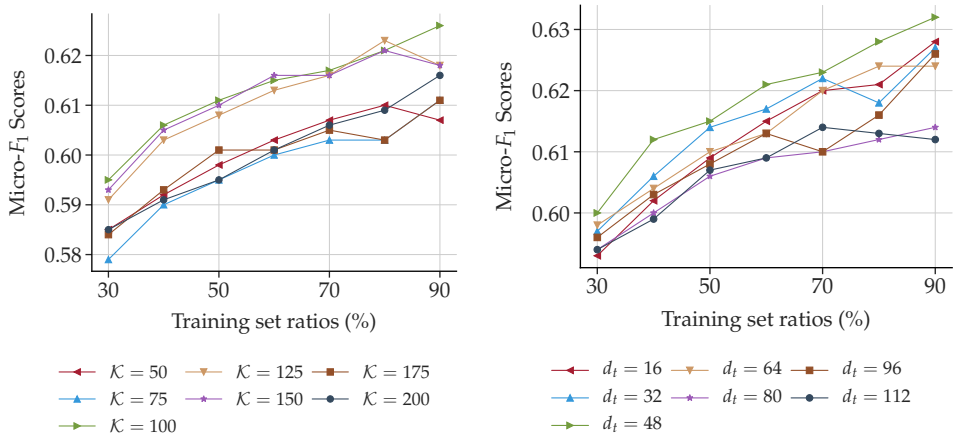
Table 3.2: Node classification for varying training sizes on *Cora*. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

		2%	4%	6%	8%	10%	30%	50%	70%	90%
Baselines	DeepWalk	0.550	0.588	0.603	0.611	0.616	0.630	0.633	0.635	0.636
		0.496	0.527	0.540	0.547	0.551	0.563	0.565	0.566	0.567
	Node2Vec	0.575	0.599	0.612	0.618	0.623	0.636	0.639	0.641	0.641
		0.517	0.541	0.551	0.557	0.561	0.571	0.574	0.575	0.574
	LINE	0.553	0.576	0.585	0.590	0.594	0.606	0.610	0.611	0.611
		0.469	0.498	0.510	0.517	0.522	0.536	0.540	0.541	0.541
	HOPE	0.379	0.379	0.379	0.379	0.379	0.380	0.383	0.387	0.391
		0.137	0.137	0.137	0.137	0.138	0.140	0.146	0.152	0.160
	NetMF	**0.577**	0.589	0.596	0.601	0.605	0.617	0.620	0.623	0.623
		0.490	0.506	**0.513**	0.518	0.522	0.530	0.531	0.533	0.533
	GEMSEC	0.538	0.566	0.583	0.591	0.597	0.611	0.614	0.615	0.615
		0.477	0.501	0.513	0.519	0.523	0.534	0.535	0.537	0.536
	M-NMF	0.501	0.527	0.540	0.545	0.551	0.568	0.574	0.577	0.579
		0.345	0.383	0.400	0.410	0.418	0.443	0.450	0.454	0.455
TNE	GLDA	0.561	0.592	0.605	0.613	0.617	0.631	0.634	0.635	0.638
		0.505	0.532	0.542	0.550	0.554	0.563	0.565	0.566	0.569
	GHMM	0.574	0.597	0.608	0.615	0.619	0.632	0.635	0.636	0.634
		0.516	0.536	0.545	0.550	0.552	0.563	0.565	0.567	0.564
	Louvain	0.573	**0.601**	**0.614**	**0.621**	**0.625**	**0.638**	**0.642**	0.642	**0.642**
		0.517	**0.543**	**0.555**	**0.562**	0.564	0.574	0.577	0.578	0.576
	BigClam	0.568	0.598	0.612	0.620	**0.625**	**0.638**	0.641	**0.643**	**0.642**
		0.516	0.542	0.553	0.561	**0.566**	**0.576**	**0.578**	**0.579**	**0.579**

Table 3.3: Node classification for varying training sizes on *DBLP*. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

		2%	4%	6%	8%	10%	30%	50%	70%	90%
Baselines	DeepWalk	0.110	0.131	0.143	0.151	0.156	0.188	0.206	0.218	0.226
		0.076	0.099	0.113	0.123	0.129	0.164	0.181	0.190	0.192
	Node2Vec	0.115	0.136	0.148	0.157	0.164	0.196	0.211	0.221	0.226
		0.078	0.101	0.116	0.125	0.132	0.167	0.180	0.188	0.190
	LINE	0.109	0.133	0.149	0.162	0.170	0.210	0.226	0.235	0.242
		0.063	0.084	0.099	0.111	0.120	0.164	0.181	0.191	0.192
	HOPE	0.068	0.069	0.069	0.070	0.069	0.071	0.077	0.086	0.093
		0.019	0.019	0.019	0.019	0.018	0.020	0.025	0.030	0.033
	NetMF	0.085	0.100	0.116	0.126	0.131	0.176	0.193	0.203	0.211
		0.045	0.064	0.080	0.090	0.095	0.137	0.152	0.161	0.160
	GEMSEC	0.078	0.082	0.085	0.087	0.089	0.112	0.131	0.143	0.151
		0.059	0.063	0.067	0.070	0.073	0.098	0.113	0.122	0.125
	M-NMF	0.097	0.112	0.119	0.123	0.128	0.143	0.153	0.162	0.168
		0.071	0.089	0.097	0.103	0.108	0.125	0.135	0.141	0.141
TNE	GLDA	0.117	0.139	0.152	0.164	0.173	0.214	**0.233**	**0.245**	**0.251**
		0.085	**0.111**	**0.125**	**0.137**	**0.147**	**0.183**	**0.200**	**0.207**	**0.207**
	GHMM	0.119	0.138	0.152	0.163	0.172	0.210	0.227	0.234	0.238
		0.088	**0.111**	**0.125**	**0.137**	0.144	0.178	0.192	0.196	0.191
	Louvain	0.119	0.141	0.158	0.170	0.178	0.212	0.226	0.234	0.240
		0.083	0.103	0.118	0.131	0.138	0.172	0.185	0.191	0.192
	BigClam	**0.126**	**0.147**	**0.161**	**0.170**	**0.180**	**0.215**	0.229	0.235	0.244
		0.085	0.107	0.122	0.132	0.141	0.178	0.191	0.196	0.196

Table 3.4: Node classification for varying training sizes on *PPI*. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.



(a) Number of topics on TNE-GLDA (b) Topic embedding size for TNE-LOUVAIN

Figure 3.5: Parameter sensitivity of TNE-GLDA and TNE-LOUVAIN models.

3.5.4 *The Effect of Number of Topics and Topic Embedding Sizes*

We have further analyzed the effect of the chosen number of topics $\mathcal{K}$ on the performance of the TNE-GLDA model on the *CiteSeer* network. As shown in Figure 3.5a, the increase in the number of topics makes positive contribution to the classification performance up to a certain level for TNE-GLDA model, reaching its highest F_1-score for around $\mathcal{K} = 150$. The chosen number of topics seems to be a more crucial parameter especially for larger training data sizes.

The size of the topic embedding vector learned by Equation (3.5) is another factor affecting the performance of TNE. We examine the influence of the topic embedding size on the TNE-LOUVAIN model over the *Citeseer* network. In the experiment, we have fixed the total embedding length to $d_n + d_t$ to 128, and we vary the length of topic (d_t) and node (d_n) embeddings. As it can be observed in Figure 3.5b, for small node embedding sizes, the scores diminish drastically since topic embeddings alone do not convey sufficient information to effectively represent the nodes On the contrary, when we accurately balance the lengths of topic and node embeddings (for $d_t = 48$ in this particular case), the contribution of topic-enhanced embeddings over purely node embeddings is evident.

		Citeseer	*Cora*	*DBLP*	*PPI*	*AstroPh*	*HepTh*	*Facebook*	*Gnutella*
Baselines	DeepWalk	**0.828**	0.779	0.944	0.860	0.961	0.896	0.984	0.695
	Node2Vec	0.827	0.781	0.944	0.861	0.961	0.896	0.983	0.694
	LINE	0.739	0.696	0.930	0.878	0.969	0.844	0.967	**0.839**
	HOPE	0.744	0.712	0.873	0.880	0.931	0.836	0.975	0.748
	NetMF	0.780	0.745	0.910	0.785	0.872	0.858	0.974	0.654
	GEMSEC	0.749	0.734	0.920	0.771	0.816	0.823	0.733	0.639
	M-NMF	0.751	0.714	0.891	**0.886**	0.948	0.861	0.977	0.755
TNE	GLDA	0.809	0.775	0.958	0.772	0.977	0.903	**0.993**	0.728
	GHMM	0.793	**0.806**	**0.959**	0.872	**0.979**	**0.908**	**0.993**	0.796
	Louvain	0.809	0.780	**0.959**	0.780	0.977	0.905	**0.993**	0.721
	BigClam	0.795	0.767	0.958	0.844	0.977	0.904	**0.993**	0.723

Table 3.5: Area Under Curve (AUC) scores for the link prediction task.

3.5.5 *Link Prediction*

We also evaluate the performance of TNE variants in the link prediction task by following the experimental setup described in Section 2. In Table 3.5, we provide the Area Under Curve (AUC) scores over various types of networks. TNE-GHMM model shows better performance; it outperforms the baselines over almost every network, with gains in percentage ranging from 0.92% (*Facebook*) upto 3.16% (*Cora*) compared to the best baseline method. The variations of TNE always obtain the first rank except *Citeseer*, *PPI* and *Gnutella* networks. TNE-Louvain and TNE-BigClam also demonstrate very close and comparable achievements to TNE-GHMM. The variants of the TNE model possess the highest score on *Facebook*.

3.5.6 *Running Time*

We have performed all the experiments on an Intel Xeon 2.4GHz CPU server (32 Cores) with 60GB of memory. In order to examine the exact running time of the TNE variants, we have performed an experiment on artificially generated Erdös-Rényi random graphs [ER60] of varying sizes, ranging from 2^8 to 2^{13} nodes. The running times are reported in Figure 3.6. As expected, the TNE-Louvain is the most scalable instance, due to the efficient way to infer the community structure.

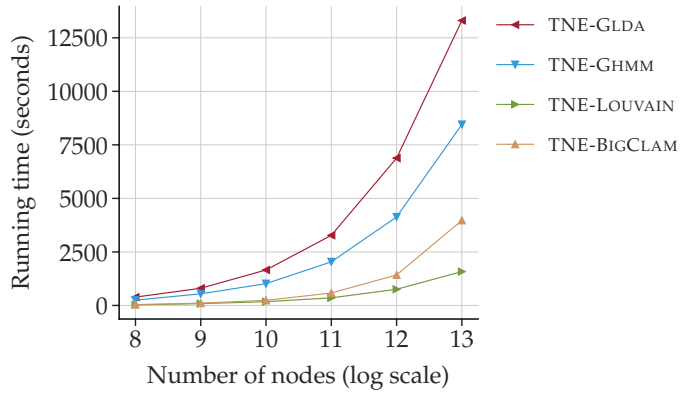


Figure 3.6: Running time analysis of the TNE model on Erdös-Rényi random graphs.

3.5.7 *Further Empirical Analysis of* TNE

We further analyze the behaviour of the various TNE instances on artificially generated networks, focusing on controlled classification experiments. We generate random graphs by following a similar approach as in the work [BG19]. In particular, we use the Stochastic Block Model (SBM) to construct graphs that consist of three clusters $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, each one consists of $1,000$ nodes. Figure 3.7 provides a schematic representation of the process. An intra-community link is added with probability p, while an edge between communities $\mathcal{A}$ and $\mathcal{B}$ is added with probability q. We use an additional parameter c in order to introduce asymmetry for inter-community links for the community pairs $\mathcal{A}-\mathcal{C}$ and $\mathcal{B}-\mathcal{C}$. We have constructed three networks ($\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_3$) for parameters $p = [0.06, 0.065, 0.07]$, $q = [0.04, 0.035, 0.003]$ and $c = [1.25, 1.429, 1.667]$, respectively. In our classification experiments, node labels correspond to community assignments.

Table 3.6 provides the Micro-F_1 scores for different walk lengths on $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$, with 40% of the datasets used as training set. We perform uniform random walks following DEEPWALK's strategy, and the instances of TNE are fed with the same node sequences. As we can observe, the performance of the models increases depending on the walk length, since by increasing the number of center-context node pairs improves the ability of the models to capture the structural properties of the graph. Comparing the extreme cases

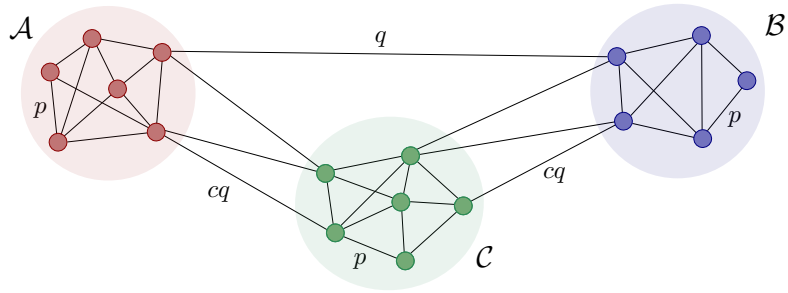


Figure 3.7: Link sampling in the Stochastic Block Model.

	$\mathcal{G}_1$			$\mathcal{G}_2$			$\mathcal{G}_3$		
Walk length, $\mathcal{L}$	10	50	100	10	50	100	10	50	100
DeepWalk	0.708	0.721	0.720	0.859	0.856	0.861	0.914	0.931	0.936
TNE-Glda	0.708	0.750	0.759	0.860	0.875	0.876	0.913	0.938	0.944
TNE-Ghmm	0.707	0.752	0.762	0.861	0.877	0.875	0.916	0.938	0.944
TNE-Louvain	**0.717**	**0.769**	**0.771**	**0.864**	**0.882**	**0.877**	**0.964**	**0.964**	**0.969**
TNE-BigClam	0.710	0.749	0.757	0.856	0.873	0.872	0.912	0.933	0.940

Table 3.6: Classification on the SBM with Micro-F_1 scores for 40% training set ratio.

of $\mathcal{G}_1$ and $\mathcal{G}_3$, the latter shows more distinguishable community structure. Therefore, TNE has better performance on $\mathcal{G}_3$ since the community detection algorithms are able to correctly assign topic-community labels. Among them, TNE-Louvain is the best performing instance. It detects good modularity communities from the graph structure itself, and thus, contrary to TNE-Glda and TNE-Ghmm, it does not rely on the the generated random walk sequences. Furthermore, it is more successful to find the clusters of the stochastic block model graphs compared to TNE-BigClam, since BigClam focuses on overlapping communities that do not appear on these artificially generated graphs.

3.6 CONCLUSION AND FUTURE WORK

In this chapter, we have proposed TNE, a topic-aware family of models for GRL. TNE takes advantage of the latent community structure of graphs, leading to the concept of topical node embeddings. We have examined four

instances of the proposed model, that either use random walks to learn topic representations or rely on well-known community detection algorithms. TNE is capable of producing enriched representations, compared to traditional random walk-based approaches or matrix factorization-based models, leading to improved performance results in the tasks of node classification and link prediction.

As future work, we are interested to extend the TNE model towards utilizing the hierarchical community structure of real-world networks, learning hierarchical node representations. Furthermore, we plan to examine the robustness of the learning representations with respect to changes on the structure of the graph.

4

EXPONENTIAL FAMILY GRAPH EMBEDDINGS

Graph representation learning techniques relying on random walks generate node sequences to capture the relationships among nodes. The extracted information is further used in learning node embeddings with an approach similar to the popular SkipGram model. This chapter emphasizes exponential family distributions to capture rich interaction patterns between nodes. We introduce the generic *exponential family graph embedding* model that generalizes random walk-based network representation learning techniques to exponential family conditional distributions. We study three particular instances of this model, analyzing their properties and showing their relationship to existing unsupervised learning models. Our experimental evaluation on real-world datasets demonstrates that the proposed techniques outperform well-known baseline methods in two downstream machine learning tasks: node classification and link prediction.

4.1 INTRODUCTION

As we have discussed in Chapter 2, random walk-based approaches typically sample a set of node sequences from the input graph. The main point that essentially differentiates these methods from each other is their strategy to generate (i.e., sample) these sequences. They construct *center-context* node pairs by examining the occurrences of nodes in a certain distance with respect to each other inside the walks. Then, widely used NLP models, such as the SkipGram model [Mik+13b], are applied to learn latent node representations.

Although SkipGram approach models the conditional distribution of nodes within a random walk based on the *softmax* function, it might prohibit to capture rich interaction patterns among nodes. Motivated by the limitation of current random walk-based GRL methodologies, we argue that

considering more expressive *conditional probability models* might lead to more informative latent node representations.

In particular, we capitalize on *exponential family* models to grasp interactions between nodes in random walks, which are a mathematically convenient parametric set of probability distributions allowing us to represent relationships among entities flexibly. More precisely, we introduce the concept of *Exponential Family Graph Embeddings* (EFGE), which generalizes random walk-based GRL techniques to exponential family conditional distributions. We study three particular instances of the proposed EFGE model that correspond to widely known exponential family distributions, namely the Bernoulli, Poisson, and Normal distributions. The extensive experimental evaluation of the proposed models in the tasks of node classification and link prediction suggests that the proposed EFGE approaches can further improve the predictive capabilities of node embeddings, compared to traditional SkipGram-based and other baseline methods. In addition, we further study the objective function of the proposed parametric models, providing connections to well-known unsupervised graph learning models under appropriate parameter settings.

The major contributions of the chapter can be summarized as follows:

- We introduce a novel representation learning model, called EFGE, which generalizes classical SkipGram-based approaches to exponential family distributions, towards more expressive GRL methods that rely on random walks. We study three instances of the model, namely the EFGE-Bern, EFGE-Pois, and EFGE-Norm models, that correspond to well-known distributions.

- We show that the objective functions of existing unsupervised representation learning models, including word embedding in NLP [Mik+13a] and overlapping community detection [YL13], can be re-interpreted under the EFGE model.

- In a thorough experimental evaluation, we demonstrate that the proposed exponential family graph embedding models generally outperform widely used baseline approaches in various learning tasks on

graphs. In addition, the running time to learn the representations is similar to other SkipGram-based models.

The rest of the chapter is organized as follows. In Section 4.2 we introduce some preliminary and background concepts. Then, Section 4.3 introduces the proposed approach. The experimental evaluation is presented in Section 4.4. Finally, we summarize and conclude our paper in Section 4.5.

SOURCE CODE. The implementation of the proposed models is provided in the following adress: https://abdcelikkanat.github.io/projects/EFGE/.

4.2 BACKGROUND CONCEPTS AND RELATED WORK

In this section, we will briefly review some preliminary concepts related to random-walk based approaches and exponential families.

4.2.1 *Random Walk-based Node Embeddings*

As we have discussed in the previous chapters, the objective function of random walk-based node representation learning models is defined in the following way:

$$\underset{\Omega}{\operatorname{argmax}} \frac{1}{N \cdot \mathcal{L}} \sum_{\mathbf{w} \in \mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{v \in \mathcal{C}_{\gamma}^{(l)}(\mathbf{w})} \log p\left(y_{\mathbf{w},v}^{(l)};\ \Omega\right), \tag{4.1}$$

where $y_{\mathbf{w},v_j}^{(l)}$ represents the relationship between the center w_l and context node v in the walk $\mathbf{w} = (w_1, \ldots, w_{\mathcal{L}}) \in \mathcal{W}$. Here, $\mathcal{C}_{\gamma}^{(l)}(\mathbf{w}) := \{w_{l+j} : \max\{1, l-\gamma\} \leq j \neq 0 \leq \min\{l+\gamma, \mathcal{L}\}\}$ denotes the set of context nodes of node w_l, $\mathcal{N}$ is the number of walks, $\mathcal{L}$ is walk length and $\Omega = (\mathbf{A}, \mathbf{B})$ is the model parameters. We consider $\mathbf{A}[v]$ to represent the embedding vector of v in all downstream machine learning applications.

Random walk-based graph representation learning methods can use different approaches to sample the context of a particular node. For instance, DeepWalk performs uniform truncated random walks, while Node2Vec

is based on second-order random walks to capture context information. Another crucial point related to SkipGram based models is the way that how the relationship among center and context nodes in Equation (4.1) is modeled. In particular, DeepWalk uses the *softmax* function to model the conditional distribution $p(\cdot)$ of a context node for a given center, in such a way that nodes occurring in similar contexts tend to get close to each other in the latent representation space. In a similar way, Node2Vec adopts the negative sampling strategy, where it uses the sigmoid function to model the occurrence of a node in the context of another one. As we will present in the following section, we rely on exponential family distributions in order to further extend and generalize random-walk GRL models to conditional probability distribution beyond the *softmax* function—towards capturing richer types of node interaction patterns.

4.2.2 *Exponential Families*

We introduce the *exponential family distributions*, a parametric set of probability distributions that include, among others, the Gaussian, Binomial, and Poisson distributions. A class of probability distributions is called exponential family distributions if they can be expressed as

$$p(y) = h(y) \exp\Big(\eta T(y) - A(\eta)\Big), \tag{4.2}$$

where h is the *base measure*, η is the *natural parameter*, T is the *sufficient statistic* of the distribution and $A(\eta)$ is the *log-normalizer* or *log-partition* function [And70]. Different choices of base measure and sufficient statistics lead us to obtain different probability distributions. For instance, the base measure and sufficient statistic of the *Bernoulli* distribution are $h(y) = 1$ and $T(y) = y$ respectively, while for the *Poisson* distribution we have $h(y) = 1/y!$ and $T(y) = y$.

As we mentioned above, exponential families contain a wide range of commonly used distributions, providing a general class of models by reparameterizing distributions in terms of the natural parameters η. In the related literature, exponential family distributions have been utilized to learn

embeddings for high-dimensional data of different types (e.g., market basket analysis) [Rud+16; Liu+17; Rud+17]. Our approach generalizes exponential family embedding models to graphs by redefining the natural parameter $\eta_{v,u}$ as the product of context and center vectors in the following way:

$$\eta_{v,u} := f\Big(\mathbf{A}[v]^\top \cdot \mathbf{B}[u]\Big),$$

where f is called the *link function*. As we will present shortly in the following section, we have many alternative options for the form of the link function.

4.3 LEARNING NODE EMBEDDINGS WITH EXPONENTIAL FAMILIES

In this section, we propose the *exponential family graph embedding* models, referred to as EFGE. The main idea behind this family of models is to utilize the expressive power of exponential family distribution towards conditioning context nodes with respect to the center node of interest. Initially, we will describe the formulation of the general objective function of the EFGE model, and then we will present particular instances of the model based on different exponential family distributions.

Let $\mathcal{W}$ be a collection of node sequences generated by following a random walk strategy over a given graph G, as defined in Chapter 2. Based on that, we can define a generic objective function to learn node embeddings in the following way:

$$\mathcal{L}(\mathbf{A}, \mathbf{B}) := \underset{\Omega}{\operatorname{argmax}} \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{v\in\mathcal{V}} \log p(y^{(l)}_{(\mathbf{w},v)}; \Omega) + \frac{\lambda}{2}\left(\|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2\right), \quad (4.3)$$

where $y^{(l)}_{(\mathbf{w},v)}$ is the observed value indicating the relationship between the center w_l and context node v. We incorporate the objective with the regularization term to obtain more informative embedding, which also improves the performance of the model in downstream tasks. The term λ refers to the regularization parameter and $\|\cdot\|_F$ is the *Frobenius* norm. Here, we aim to find embedding vectors $(\mathbf{A}, \mathbf{B})$ by maximizing the log-likelihood in Equation (4.3). Note that, the objective function in Equation (4.3) is quite similar to the

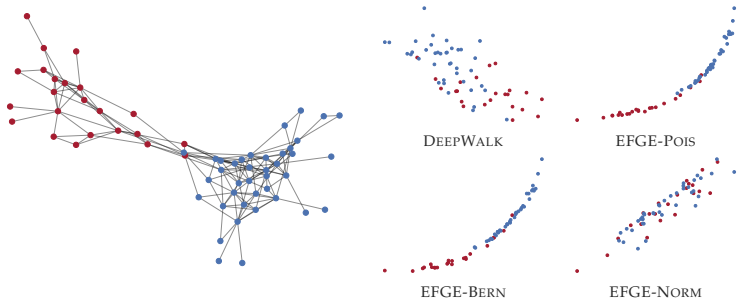


Figure 4.1: Visualization of the *Dolphins* network composed by 2 communities and the corresponding embeddings for $d = 2$.

one of the SkipGram model [Mik+13a] presented in Equation (4.1), except that we also include nodes that are not belonging to context sets.

Instead of restricting ourselves to the *Sigmoid* or *Softmax* functions in order to model the probability in the objective function of Equation (4.3), we provide a generalization assuming that each $y^{(l)}_{\mathbf{w},v}$ follows an exponential family distribution. That way, the objective function used to learn node embedding vectors $\Omega = (\mathbf{A}, \mathbf{B})$ can be rewritten as follows:

$$\underset{\Omega}{\operatorname{argmax}} \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{v\in\mathcal{V}} \log h(y^{(l)}_{\mathbf{w},v}) + \eta_{w_l,v} T(y^{(l)}_{\mathbf{w},v}) - A(\eta_{w_l,v}) + \mathcal{R}(\mathbf{A}, \mathbf{B}), \quad (4.4)$$

where $\mathcal{R}(\mathbf{A}, \mathbf{B})$ indicates the regularization term in Equation 4.3 and it will be omitted in the rest of the chapter for simplicity. As we can observe, Equation (4.4) which is the objective function of the generic EFGE graph embeddings model, generalizes the classic models to exponential family conditional distributions given in Equation (4.2). That way, the proposed EFGE approaches have the additional flexibility to utilize a wide range of exponential distributions, allowing them to capture more complex types of node interactions beyond simple co-occurrence relationships. It is also important to stress out that, the first term of Equation (4.4) does not depend on parameter $\eta_{w_l,v}$; this brings an advantage during the optimization process.

A general overview of our proposed model is as follows: Initially, we sample a set of $\mathcal{N}$ random walks relying on a chosen walk strategy. This strategy can be any context sampling process, such as uniform random

walks (as in DEEPWALK [PARS14]) or biased random walks (as in NODE2VEC [GL16] or in BIAEDWALK [NM18]). Then, based on the chosen instance of the EFGE model, we learn center and context embedding vectors. In this chapter, we examine three particular instances of the EFGE model, that represent well known exponential family distributions. In particular, we utilize the Bernoulli, Poisson, and Normal distributions leading to the corresponding EFGE-BERN, EFGE-POIS and EFGE-NORM models. For illustration purposes, Figure 4.1 depicts the *Dolphins* network composed by two communities and the embeddings in two dimensions as computed by different models. As we can observe, for this particular toy example, the proposed EFGE-BERN and EFGE-POIS approches learn representations that are able to differentiate nodes with respect to their communities. In the following sections, we analyze the properties of these models in detail.

4.3.1 *The* EFGE-BERN *Model*

Our first model is the EFGE-BERN, in which we will model the occurrence of a node in the context set of another one. Let $Y^{(l)}_{\mathbf{w},v}$ be a random variable following a Bernoulli distribution which is equal to 1 if node v appears in the context set $\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})$ of node w_l. It can be written by $Y^{(l)}_{(\mathbf{w},v)} = X^{(l-\gamma)}_{(\mathbf{w},v)} \vee \cdots \vee X^{(l-1)}_{(\mathbf{w},v)} \vee X^{(l+1)}_{(\mathbf{w},v)} \vee \cdots \vee X^{(l+\gamma)}_{(\mathbf{w},v)}$, where each $X^{(l+j)}_{(\mathbf{w},v)}$ indicates the appearance of v at the specific position $l+j$ $(-\gamma \leq j \neq 0 \leq \gamma)$ in the walk $\mathbf{w} \in \mathcal{W}$. Here, we assume that $X^{(l+j)}_{(\mathbf{w},v)}$'s, are independent variables. We can express the objective function of the EFGE-BERN model, $\mathcal{L}_B(\mathbf{A}, \mathbf{B})$, by dividing Equation (4.4) into two parts with respect to the values of $Y_{(\mathbf{w},v)}$ and $X^{(l+j)}_{(\mathbf{w},v)}$:

$$\begin{aligned}\mathcal{L}_B(\mathbf{A}, \mathbf{B}) &= \underset{\Omega}{\operatorname{argmax}} \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \left(\sum_{v\in\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log p(y^{(l)}_{(\mathbf{w},v)}) + \sum_{v\notin\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log p(y^{(l)}_{(\mathbf{w},v)}) \right) \\ &= \underset{\Omega}{\operatorname{argmax}} \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{-\gamma\leq j\leq\gamma \\ j\neq 0}} \left(\log p(x^{(l+j)}_{(\mathbf{w},w_{l+j})}) + \sum_{\substack{v\in\mathcal{V} \\ v\neq w_{(l+j)}}} \log p(x^{(l+j)}_{(\mathbf{w},v)}) \right). \end{aligned} \tag{4.5}$$

Note that, the exponential form of a Bernoulli distribution with a parameter π is equal to $\exp(\eta x - A(\eta))$, where the log-normalizer $A(\eta)$

is $\log(1+\exp(\eta))$ and the parameter π is the sigmoid function $\sigma(\eta) = 1/(1-\exp(-\eta))$. Therefore, we can rewrite the objective function $\mathcal{L}_B(\mathbf{A},\mathbf{B})$ as follows:

$$\mathcal{L}_B(\mathbf{A},\mathbf{B}) = \underset{\Omega}{\operatorname{argmax}} \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{-\gamma\leq j\leq\gamma \\ j\neq 0}} \left(\log \sigma(\eta_{(w_l,w_{l+j})}) + \sum_{\substack{v\in\mathcal{V} \\ v\neq w_{l+j}}} \log\sigma(-\eta_{(w_l,v)}) \right).$$

We choose the identity map for the link function $f(\cdot)$, so $\eta_{w_l,v}$ directly becomes equal to the product of vectors $\mathbf{A}[v]$ and $\mathbf{B}[w_l]$.

4.3.1.1 *Relationship to negative sampling*

Although the *negative sampling* strategy [Mik+13b; MK13] was proposed to approximate the objective function of the SkipGram model, any rigorous theoretical argument showing the connection between them has not been provided. In Lemma 4.1, we show that the log-likelihood $\mathcal{L}_B(\mathbf{A},\mathbf{B})$ of the EFGE-Bern model in fact can approximated by the objective function of negative sampling given in Equation (4.6). In our implementation, we adopt negative sampling in order to improve the efficiency of the computation.

Lemma 4.1. *The log-likelihood function $\mathcal{L}_B(\mathbf{A},\mathbf{B})$ can be approximated by*

$$\sum_{\boldsymbol{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{-\gamma\leq j\leq\gamma \\ j\neq 0}} \left(\log p\big(x^{(l+j)}_{(\boldsymbol{w},w_{l+j})}\big) + \sum_{r=1}^{k} \mathop{\mathbb{E}}_{s\sim q^-} \left[\log p\big(x^{(l+j)}_{(\boldsymbol{w},s)}\big) \right] \right) \tag{4.6}$$

for large values of k.

Proof. Let $q^-(\cdot|w_l)$ be the true conditional distribution of a random walk method for generating negative samples defined over $\mathcal{V}$. Then, Equation 4.5 can be rewritten in the following way:

$$\begin{aligned} &\sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{-\gamma\leq j\leq\gamma \\ j\neq 0}} \left(\log p\big(x^{(l+j)}_{(\mathbf{w},w_{l+j})}\big) + \sum_{r=1}^{k} \mathop{\mathbb{E}}_{s\sim q^-} \Big(\log p\big(x^{(l+j)}_{(\mathbf{w},s)}\big) \Big) \right) \\ \approx &\sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{v\in\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \left(\log p\big(y^{(l)}_{\mathbf{w},v}\big) + \sum_{r=1}^{k} \mathop{\mathbb{E}}_{s\sim q^-_y} \Big(\log p\big(y^{(l)}_{(\mathbf{w},s)}\big) \Big) \right) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \left(\sum_{v\in\mathcal{C}_{\mathbf{w}}(w_l)} \log p(y^{(l)}_{(\mathbf{w},v)}) + \sum_{c=1}^{|\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})|} \sum_{r=1}^{k} \mathop{\mathbb{E}}_{s\sim q_y^-} \left(\log p(y^{(l)}_{(\mathbf{w},s)}) \right) \right) \\
&\approx \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \left(\sum_{v\in\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log p(y^{(l)}_{(\mathbf{w},v)}) + \sum_{c=1}^{|\mathcal{C}^{(l)}_{\gamma}(\mathbf{w}))|} \sum_{r=1}^{k} \frac{1}{|\mathcal{V}\backslash\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})|} \sum_{v\notin\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log p(y^{(l)}_{(\mathbf{w},v)}) \right) \\
&= \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{L} \left(\sum_{v\in\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log p(y^{(l)}_{(\mathbf{w},v)}) + \sum_{v\notin\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log p(y^{(l)}_{(\mathbf{w},v)}) \right) \\
&= \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{v\in\mathcal{V}} \log p(y^{(l)}_{(\mathbf{w},v)}) \\
&= \mathcal{L}_B(\mathbf{A},\mathbf{B}),
\end{aligned}
$$

where the fourth line follows from the law of large numbers for the sample size of $|\mathcal{V}\backslash\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})|$, and k is set to $|\mathcal{V}\backslash\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})|/|\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})|$ in the fourth line. □

4.3.2 *The* EFGE-Pois *Model*

In this model, we will use the Poisson distribution to capture the relationship between context and center nodes in a random walk sequence. Let $Y^{(l)}_{(\mathbf{w},v)}$ be a value indicating the number of occurrences of node v in the context of w_l. We assume that $Y^{(l)}_{(\mathbf{w},v)}$ follows a Poisson distribution, with the mean value $\widetilde{\lambda}_{(w_l,v)}$ being the number of appearances of node v in the surroundings of w_l within the window size γ. Similar to the previous model, it can be expressed as $Y^{(l)}_{(\mathbf{w},v)} = X^{(l-\gamma)}_{(\mathbf{w},v)} + \cdots + X^{(l-1)}_{(\mathbf{w},v)} + X^{(l+1)}_{(\mathbf{w},v)} + \cdots + X^{(l+\gamma)}_{(\mathbf{w},v)}$, where $X^{(l+j)}_{(\mathbf{w},v)} \sim Pois(\lambda^{(l+j)}_{(w_l,v)})$ for $-\gamma \leq j \neq 0 \leq \gamma$. That way, we obtain $\widetilde{\lambda}_{(w_l,v)} = \sum_{j=-\gamma}^{\gamma} \lambda^{(l+j)}_{(w_l,v)}$, since the sum of independent Poisson random variables is also Poisson. By plugging the exponential form of the Poisson distribution into Equation (4.3), we can derive the objective function $\mathcal{L}_P(\mathbf{A},\mathbf{B})$ of the model as:

$$
\underset{\Omega}{\operatorname{argmax}} \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{v\in\mathcal{V}} \left(\log h(y^{(l)}_{(\mathbf{w},v)}) + \left(\widetilde{\eta}_{(w_l,v)} y^{(l)}_{(\mathbf{w},v)} - \exp(\widetilde{\eta}_{(w_l,v)}) \right) \right),
$$

where the base measure $h(y^{(l)}_{(\mathbf{w},v)})$ is equal to $1/y^{(l)}_{(\mathbf{w},v)}!$. Note that, the number of occurrence $y^{(l)}_{(\mathbf{w},v)}$ is equal to 0 if v does not appear in the context of $w_l \in \mathcal{V}$. Following a similar strategy as in the EFGE-Bern model, the equation can be split into two parts for the cases where $y_{(\mathbf{w},v)} > 0$ and $y_{(\mathbf{w},v)} = 0$. That way, we can adopt the negative sampling strategy (given in Equation (4.6)) as follows:

$$\sum_{\mathbf{w}\in\mathcal{W}}\sum_{l=1}^{L}\sum_{\substack{-\gamma\leq j\leq\gamma\\ j\neq 0}}\Bigg(\Big(-\log(x^{(l+j)}_{(\mathbf{w},w_{l+j})}!)+\eta_{(w_l,w_{l+j})}x^{(l+j)}_{(\mathbf{w},w_{l+j})}-\exp(\eta_{(w_l,w_{l+j})})\Big)+ \sum_{r=1}^{k}\mathbb{E}_{s\sim q^-}\Big[-\exp(\eta_{(w_l,s)})\Big]\Bigg).$$

Note that, in the EFGE-Pois model, we do not specify any particular link function; thus, the natural parameter is equal to the product of the embeddings vectors.

4.3.2.1 *Relationship to overlapping community detection*

As we have discussed in Section 3.2 of Chapter 3, BigClam [YL13] is a widely used overlapping community detection method. It can be seen that an objective function similar to BigClam [YL13], can be obtained by unifying the objectives of the EFGE-Bern and EFGE-Pois models. The relationship is provided in Lemma 4.2. Besides, each entry of the node embedding vector can be considered as a coefficient indicating the node's membership strength to the corresponding community. In this case of BigClam, the vector entries are restricted to non-negative values.

Lemma 4.2. *Let $Z^{(l)}_{(\boldsymbol{w},v)}$ be independent random variables following Poisson distribution with natural parameter $\eta_{(w_l,v)}$ which is defined by $\log(\mathbf{A}[v]^\top\cdot\mathbf{B}[w_l])$ and let $Y^{(l)}_{(\boldsymbol{w},v)} \sim Bern(\pi^{(l)}_{(\boldsymbol{w},v)})$. For a set, $\mathcal{W}$, of walks of length 2 containing all distinct node pairs, if the parameter $\pi^{(l)}_{(\boldsymbol{w},v)}$ is chosen by $p(Z^{(l)}_{(\boldsymbol{w},v)} > 0)$, then the objective function of* EFGE-Bern *model is equal to*

$$2\left(\sum_{(u,v)\in\mathcal{E}}\log\Big(1-\exp\big(-\mathbf{A}[v]^\top\cdot\mathbf{B}[w_l]\big)\Big)-\sum_{(u,v)\notin\mathcal{E}}\mathbf{A}[v]^\top\cdot\mathbf{B}[w_l]\right).$$

Proof. Let $Y^{(l)}_{(\mathbf{w},v)}$ be Bernoulli distributions with parameter $\pi_{(w_l,v)}$ as defined above. Then, the objective function $\mathcal{L}_B(\mathbf{A},\mathbf{B})$ defined in Subsection 4.3.1 can be divided into parts and it can be written as follows:

$$
\begin{aligned}
\mathcal{L}_{\mathcal{B}}(\mathbf{A},\mathbf{B}) &= \sum_{\mathbf{w}\in\mathcal{W}}\sum_{l=1}^{\mathcal{L}}\sum_{v\in\mathcal{V}} \log p(y^{(l)}_{(\mathbf{w},v)}) \\
&= \sum_{\mathbf{w}\in\mathcal{W}}\sum_{l=1}^{\mathcal{L}}\left(\sum_{v\in\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log p(y^{(l)}_{(\mathbf{w},v)}=1) + \sum_{v\notin\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log p(y^{(l)}_{(\mathbf{w},v)}=0)\right) \\
&= \sum_{\mathbf{w}\in\mathcal{W}}\sum_{l=1}^{\mathcal{L}}\left(\sum_{v\in\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log p(z^{(l)}_{(\mathbf{w},v)}>0) + \sum_{v\notin\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log p((z^{(l)}_{\mathbf{w},v)}=0)\right) \\
&= \sum_{\mathbf{w}\in\mathcal{W}}\sum_{l=1}^{\mathcal{L}}\left(\sum_{v\in\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log\left(1-p(z^{(l)}_{\mathbf{w},v}=0)\right) + \sum_{v\notin\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log p(z^{(l)}_{(\mathbf{w},v)}=0)\right) \\
&= \sum_{\mathbf{w}\in\mathcal{W}}\sum_{l=1}^{\mathcal{L}}\left(\sum_{v\in\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log\left(1-\exp(\eta_{(w_l,v)})\right) - \sum_{v\notin\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \eta_{(w_l,v)}\right) \\
&= \sum_{\mathbf{w}\in\mathcal{W}}\sum_{l=1}^{\mathcal{L}}\left(\sum_{v\in\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \log\left(1-\exp\left(-\mathbf{A}[v]^{\top}\cdot\mathbf{B}[w_l]\right)\right) - \sum_{v\notin\mathcal{C}^{(l)}_{\gamma}(\mathbf{w})} \mathbf{A}[v]^{\top}\cdot\mathbf{B}[w_l]\right) \\
&= 2\sum_{(u,v)\in\mathcal{E}} \log\left(1-\exp\left(-\mathbf{A}[v]^{\top}\cdot\mathbf{B}[w_l]\right)\right) - 2\sum_{(u,v)\notin\mathcal{E}} \mathbf{A}[v]^{\top}\cdot\mathbf{B}[w_l].
\end{aligned}
$$

The last line follows from the assumption that the set of walks, $\mathcal{W}$, consists of all distinct node pairs of length 2, which in fact corresponds to the edge set of the graph, and the window size, γ, is set to 1.

□

4.3.3 *The* EFGE-Norm *Model*

If a node v appears in the context of w_l more frequently with respect to other nodes, we can say that v has a higher interaction with w_l than the rest of the nodes. Therefore, we will consider each $y^{(l)}_{(\mathbf{w},v)}$ in this model as a weight indicating the strength of the relationship between the nodes w_l and v. We assume that $X^{(l+j)}_{(\mathbf{w},v)} \sim \mathcal{N}(1,\sigma^2_+)$ if $v \in \mathcal{C}^{(l)}_{\gamma}(\mathbf{w})$, and $X^{(l+j)}_{(\mathbf{w},v)} \sim \mathcal{N}(0,\sigma^2_-)$

otherwise. Hence, we can obtain that $Y^{(l)}_{(\mathbf{w},v)} \sim \mathcal{N}(\widetilde{\mu}, \widetilde{\sigma}^2)$ if we follow a similar assumption $Y^{(l)}_{(\mathbf{w},v)} = \sum_{j=-\gamma}^{\gamma} X^{(l+j)}_{(\mathbf{w},v)}$ as in the previous model where $\widetilde{\mu}$ is the number of occurrences of v in the context. The definition of the objective function, $\mathcal{L}_N(\mathbf{A}, \mathbf{B})$, for the EFGE-Norm model is defined as follows:

$$\mathcal{L}_N(\mathbf{A}, \mathbf{B}) = \sum_{\mathbf{w} \in \mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{-\gamma \leq j \leq \gamma \\ j \neq 0}} \Bigg(\Big(\log h(x^{(l+j)}_{(\mathbf{w},w_{l+j})}) + \Big(x^{(l+j)}_{(\mathbf{w},w_{l+j})} \frac{\eta_{(w_l,w_{l+j})}}{\sigma^+} - \frac{\eta^2_{(w_l,w_{l+j})}}{2} \Big) \Big) + \sum_{r=1}^{k} \mathbb{E}_{s \sim q^-} \left[\log h(x^{(l+j)}_{(\mathbf{w},s)}) + \Big(x^{(l+j)}_{(\mathbf{w},s)} \frac{\eta_{(w_l,s)}}{\sigma^-} - \frac{\eta^2_{(w_l,s)}}{2} \Big) \right] \Bigg),$$

where the base measure $h(x)$ is $\exp(-x^2/2\sigma^2)/\sqrt{2\pi}\sigma$ for known variance. In this model, we choose the link function as $f(x) = \exp(-x)$, so $\eta_{(w_l,v)}$ is defined as $\exp(-\mathbf{A}[v]^\top \cdot \mathbf{B}[w_l])$.

4.3.4 *Optimization*

For the optimization step of our models, we use *Stochastic Gradient Descent* (SGD) [Bot91] to learn representations $\Omega = (\mathbf{A}, \mathbf{B})$. Since it is computationally costly to compute the gradients for each node pair, we take advantage of the formulation of the objective function of each model. It can be divided into two parts according to the values of $X^{(l+j)}_{(\mathbf{w},v)}$; thus, we adopt the negative sampling strategy, setting sampling size to $k = 5$ in all the experiments. We sample negative instances from the vertex set with respect to the number of occurrences of nodes in the generated walks raised to the power of 0.75 similar to the approach described in [Mik+13b].

4.4 EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of the proposed approach in the node classification and link prediction tasks. In the experimental evaluation of the proposed approach, we have used the same datasets described in Chapter 2. The baseline methods have been run with the same parameter settings explained in Chapter 3. For the EFGE model, we set the regular-

ization parameter, λ, to 0.01 for *PPI* network, and it is set to 0.1 for the other networks in the classification experiments. For the link prediction, it is simply considered as 0.01 for all cases.

4.4.1 *Node Classification*

EXPERIMENTAL SET-UP. For the classification task, we apply the same strategy described in the previous chapter. In brief, our goal is to predict the correct labels of nodes in the test set with having access only to a limited number of node labels in the training set. We split the nodes into a wide range of training ratios varying from 2% up to 90% in order to evaluate the models better. We perform our experiments applying a one-vs-rest logistic regression classifier with L_2 regularization.

EXPERIMENTAL RESULTS. We report the results in Tables 4.1 to 4.4 in which the first and the second rows of each method indicates the Micro-F_1 and the Macro-F_1 scores, respectively. The best and second best performing models are indicated with bold and underlined text. As we have discussed in Section 4.3, the EFGE-BERN approach models the occurrence of a node in the context of one another node. The model shows superior performances over *Cora* network, especially for small training ratios. For instance, the approach has 5.90% Micro-F_1 and 8.49% Macro-F_1 gains in scores for 2% training ratio concerning the highest baseline score and its performance gains are 1.71% and 2.01% in terms of Micro-F_1 and Macro-F_1 scores for the training set size of 10%. The model also holds the highest scores after EFGE-POIS model on the *DBLP* network. The EFGE-POIS is the best performing approach among the three EFGE instances in general. The percentage gain for Micro-F_1 score compared to the best baseline technique, varies from 3.93% up to 8.99% on *Citeseer* and differs from 2.30% up to 6.39 on the *DBLP* network. Since the approach models the number of appearances of a node in the context sets, it is more sensitive to the occurrence counts.

The last instance of EFGE that we have employed is the EFGE-NORM model, which utilizes normal distribution to represent the interaction weights between nodes. The model presents comparable performances on *Cora* and

DBLP networks and it is the best approach over the *PPI* network with up to 12.08% Micro-F_1 and 9.90% Macro-F_1 gains. The approach possesses a similar trend to EFGE-Pois model since the mean of the distribution depends on the occurrence counts. If a node appears more frequently in the context of a particular node, it will have a higher interaction weight with it.

Overall, the classification experiments show that the proposed EFGE-Pois and EFGE-Norm models perform well, outperforming most baselines, especially on a limited number of training data. This can qualitatively be explained by the fact that those exponential family distribution models enable to capture the number of occurrences of a node within the context of another one while learning the embedding vectors. Of course, the structural properties of the network, such as the existence of community structure, might affect the performance of these models. For instance, as we have seen in the toy example of Fig. 4.1, the existence of well-defined communities at the *Dolphins* network allows the EFGE-Pois model to learn more discriminative embeddings with respect to the underlying communities (as we expect to have repetitions of nodes that belong to the same community while sampling the context of a node based on random walks).

4.4.2 *Link Prediction*

EXPERIMENTAL SET-UP. For this experiment, we randomly remove half of the edges of a given network, keeping the residual network connected. Then, we learn node representations using the residual network, which are further used to build edge feature vectors with various operators. Please refer to Section 2.5 of Chapter 2 for more details about the link prediction experiment, concerning the construction of training and test sets.

EXPERIMENTAL RESULTS. Table 4.5 shows the Area Under Curve (AUC) scores for the link prediction task. The performances of EFGE variants are distinctive for the networks such as *DBLP*, *AstroPh*, *HepTh* and *Facebook* with gains 1.34%, 0.82%, 1.17% and 0.77%, respectively and the models show slightly better performance over *Citeseer* and *Cora*. Since we use the walks produced by Node2Vec, the performance of the EFGE instances highly

		2%	4%	6%	8%	10%	30%	50%	70%	90%
Baselines	DeepWalk	0.421	0.462	0.488	0.502	0.517	0.569	0.587	0.596	0.597
		0.374	0.419	0.446	0.461	0.476	0.524	0.540	0.547	0.546
	Node2Vec	0.451	0.491	0.514	0.529	0.543	0.583	0.595	0.600	0.603
		0.397	0.443	0.466	0.483	0.497	0.535	0.546	0.549	0.550
	LINE	0.300	0.353	0.384	0.407	0.418	0.476	0.494	0.505	0.512
		0.243	0.303	0.334	0.357	0.367	0.423	0.440	0.448	0.454
	HOPE	0.197	0.204	0.207	0.208	0.216	0.253	0.276	0.299	0.316
		0.060	0.065	0.066	0.068	0.075	0.121	0.150	0.178	0.202
	NetMF	0.401	0.473	0.507	0.525	0.538	0.579	0.590	0.594	0.601
		0.346	0.421	0.454	0.474	0.487	0.528	0.538	0.542	0.548
	GEMSEC	0.384	0.439	0.459	0.479	0.491	0.532	0.545	0.552	0.558
		0.339	0.400	0.422	0.439	0.451	0.488	0.497	0.499	0.501
	M-NMF	0.264	0.317	0.340	0.370	0.382	0.439	0.449	0.456	0.457
		0.161	0.229	0.261	0.293	0.306	0.370	0.381	0.390	0.390
EFGE	Bern	0.472	0.514	0.539	0.554	0.565	**0.608**	**0.621**	0.626	0.626
		0.420	0.465	0.489	0.505	0.517	**0.559**	**0.572**	0.576	0.575
	Pois	**0.491**	**0.525**	**0.547**	**0.560**	**0.571**	**0.608**	**0.621**	**0.629**	**0.627**
		0.434	**0.470**	**0.493**	**0.507**	**0.519**	0.556	0.569	**0.577**	**0.574**
	Norm	0.469	0.515	0.537	0.552	0.564	0.603	0.617	0.623	0.625
		0.412	0.460	0.486	0.500	0.513	0.554	0.566	0.571	0.571

Table 4.1: Node classification on *Citeseer* for varying training sizes . For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

		2%	4%	6%	8%	10%	30%	50%	70%	90%
Baselines	DeepWalk	0.617	0.688	0.715	0.732	0.747	0.799	0.815	0.825	0.832
		0.569	0.664	0.698	0.717	0.735	0.788	0.806	0.815	0.821
	Node2Vec	0.659	0.720	0.743	0.759	0.770	0.816	0.831	0.839	0.845
		0.612	0.694	0.724	0.743	0.755	0.804	0.820	0.828	0.832
	LINE	0.416	0.498	0.546	0.581	0.609	0.701	0.728	0.741	0.747
		0.306	0.425	0.492	0.543	0.578	0.691	0.720	0.733	0.738
	HOPE	0.284	0.301	0.301	0.302	0.302	0.302	0.302	0.304	0.306
		0.067	0.066	0.067	0.066	0.066	0.067	0.067	0.068	0.074
	NetMF	0.636	0.716	0.748	0.767	0.773	**0.821**	0.834	**0.841**	0.844
		0.591	0.694	0.731	0.751	0.760	**0.811**	**0.824**	**0.832**	**0.835**
	GEMSEC	0.470	0.530	0.568	0.587	0.601	0.674	0.714	0.735	0.744
		0.406	0.477	0.527	0.546	0.562	0.646	0.689	0.713	0.722
	M-NMF	0.507	0.580	0.622	0.642	0.656	0.717	0.732	0.736	0.742
		0.459	0.550	0.598	0.622	0.638	0.706	0.722	0.728	0.734
EFGE	Bern	0.698	**0.748**	**0.768**	**0.778**	**0.787**	0.820	**0.835**	**0.841**	**0.847**
		0.664	**0.729**	**0.753**	**0.766**	**0.775**	0.809	0.823	0.829	0.834
	Pois	**0.701**	**0.748**	0.767	0.776	0.784	0.812	0.822	0.831	0.834
		0.656	0.727	0.751	0.763	0.772	0.801	0.811	0.820	0.819
	Norm	0.682	0.743	0.766	**0.778**	0.786	0.818	0.829	0.837	0.837
		0.639	0.721	0.751	0.765	0.774	0.808	0.818	0.826	0.825

Table 4.2: Node classification on *Cora* for varying training sizes. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

		2%	4%	6%	8%	10%	30%	50%	70%	90%
Baselines	DeepWalk	0.550	0.588	0.603	0.611	0.616	0.630	0.633	0.635	0.636
		0.496	0.527	0.540	0.547	0.551	0.563	0.565	0.566	0.567
	Node2Vec	0.575	0.599	0.612	0.618	0.623	0.636	0.639	0.641	0.641
		0.517	0.541	0.551	0.557	0.561	0.571	0.574	0.575	0.574
	LINE	0.553	0.576	0.585	0.590	0.594	0.606	0.610	0.611	0.611
		0.469	0.498	0.510	0.517	0.522	0.536	0.540	0.541	0.541
	HOPE	0.379	0.379	0.379	0.379	0.379	0.380	0.383	0.387	0.391
		0.137	0.137	0.137	0.137	0.138	0.140	0.146	0.152	0.160
	NetMF	0.577	0.589	0.596	0.601	0.605	0.617	0.620	0.623	0.623
		0.490	0.506	0.513	0.518	0.522	0.530	0.531	0.533	0.533
	GEMSEC	0.538	0.566	0.583	0.591	0.597	0.611	0.614	0.615	0.615
		0.477	0.501	0.513	0.519	0.523	0.534	0.535	0.537	0.536
	M-NMF	0.501	0.527	0.540	0.545	0.551	0.568	0.574	0.577	0.579
		0.345	0.383	0.400	0.410	0.418	0.443	0.450	0.454	0.455
EFGE	Bern	0.603	0.619	0.629	0.634	0.639	0.651	0.655	0.656	0.656
		0.545	0.562	0.572	0.577	**0.581**	0.590	0.593	**0.594**	0.594
	Pois	**0.614**	**0.626**	**0.632**	**0.637**	**0.640**	0.652	0.655	0.656	**0.658**
		0.552	**0.566**	**0.573**	**0.578**	**0.581**	**0.591**	**0.593**	0.594	**0.596**
	Norm	0.610	0.624	0.631	0.636	**0.640**	**0.653**	**0.656**	**0.657**	**0.658**
		0.547	0.564	0.571	0.577	0.580	**0.591**	**0.593**	0.594	0.595

Table 4.3: Node classification on *DBLP* for varying training sizes. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

		2%	4%	6%	8%	10%	30%	50%	70%	90%
Baselines	DeepWalk	0.110	0.131	0.143	0.151	0.156	0.188	0.206	0.218	0.226
		0.076	0.099	0.113	0.123	0.129	0.164	0.181	0.190	0.192
	Node2Vec	0.115	0.136	0.148	0.157	0.164	0.196	0.211	0.221	0.226
		0.078	0.101	0.116	0.125	0.132	0.167	0.180	0.188	0.190
	LINE	0.109	0.133	0.149	0.162	0.170	0.210	0.226	0.235	0.242
		0.063	0.084	0.099	0.111	0.120	0.164	0.181	0.191	0.192
	HOPE	0.068	0.069	0.069	0.070	0.069	0.071	0.077	0.086	0.093
		0.019	0.019	0.019	0.019	0.018	0.020	0.025	0.030	0.033
	NetMF	0.085	0.100	0.116	0.126	0.131	0.176	0.193	0.203	0.211
		0.045	0.064	0.080	0.090	0.095	0.137	0.152	0.161	0.160
	GEMSEC	0.078	0.082	0.085	0.087	0.089	0.112	0.131	0.143	0.151
		0.059	0.063	0.067	0.070	0.073	0.098	0.113	0.122	0.125
	M-NMF	0.097	0.112	0.119	0.123	0.128	0.143	0.153	0.162	0.168
		0.071	0.089	0.097	0.103	0.108	0.125	0.135	0.141	0.141
EFGE	Bern	0.125	0.148	0.162	0.172	0.179	0.212	0.227	0.236	0.240
		0.087	**0.111**	**0.126**	**0.137**	**0.144**	0.178	0.191	0.197	0.195
	Pois	0.126	**0.151**	0.166	0.176	0.184	0.217	0.231	0.239	0.242
		0.084	0.108	0.123	0.136	0.142	0.177	0.191	0.197	0.194
	Norm	**0.128**	**0.151**	**0.167**	**0.177**	**0.185**	**0.220**	**0.234**	**0.241**	**0.247**
		0.085	0.109	0.124	0.135	0.143	**0.179**	**0.193**	**0.198**	**0.197**

Table 4.4: Node classification on *PPI* for varying training sizes . For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

		Citeseer	*Cora*	*DBLP*	*PPI*	*AstroPh*	*HepTh*	*Facebook*	*Gnutella*
Baselines	DEEPWALK	0.828	0.779	0.944	0.860	0.961	0.896	0.984	0.695
	NODE2VEC	0.827	0.781	0.944	0.861	0.961	0.896	0.983	0.694
	LINE	0.739	0.696	0.930	0.878	0.969	0.844	0.967	**0.839**
	HOPE	0.744	0.712	0.873	0.880	0.931	0.836	0.975	0.748
	NETMF	0.780	0.745	0.910	0.785	0.872	0.858	0.974	0.654
	GEMSEC	0.749	0.734	0.920	0.771	0.816	0.823	0.733	0.639
	M-NMF	0.751	0.714	0.891	**0.886**	0.948	0.861	0.977	0.755
EFGE	BERN	0.820	0.753	0.954	0.728	**0.977**	0.899	**0.991**	0.622
	POIS	**0.829**	0.777	**0.957**	0.739	0.973	0.902	**0.991**	0.624
	NORM	0.783	**0.782**	0.956	0.806	0.966	**0.907**	0.990	0.668

Table 4.5: Area Under Curve (AUC) scores for link prediction.

affected by these generated walks; this can be especially seen over *Gnutella* network. The average clustering coefficients of *Gnutella*, *PPI* and *Citeseer* are relatively small compared to the rest of the network and the scores reveal the possible correlation between the clustering coefficient and the performance of EFGE variants.

4.4.3 *Parameter Sensitivity*

In this subsection, we evaluate how the performance of our models is affected under different parameter settings. In particular, we mainly examine the effect of embedding dimension d and the effect of the window size γ used to sample context nodes.

THE EFFECT OF DIMENSION SIZE. The dimension size d of embedding vectors is a crucial parameter affecting the performance of models. Therefore, we have conducted experiments examining the influence of embedding dimension d on the *Citeseer* network. As shown in Figure 4.2, the increase in the dimension size has a positive impact for all models over Micro-F_1 scores. When the dimension size increases from 32 up to 224, we observe a gain of around 18% for EFGE-BERN, around 17% for EFGE-POIS and 14.43% for EFGE-NORM model over the training set constructed from 50% of the network. While there is a big leap in the scores from the dimension size

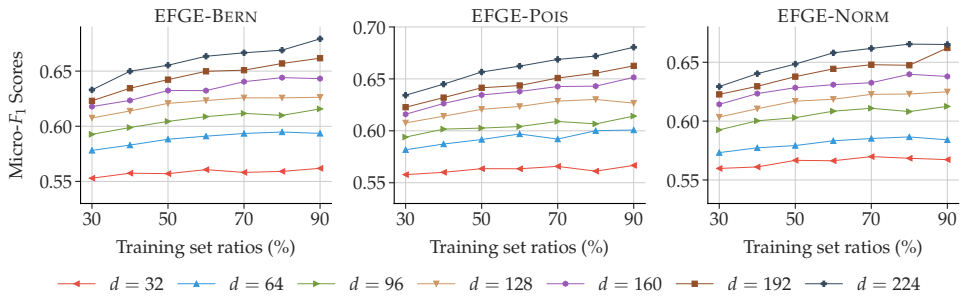
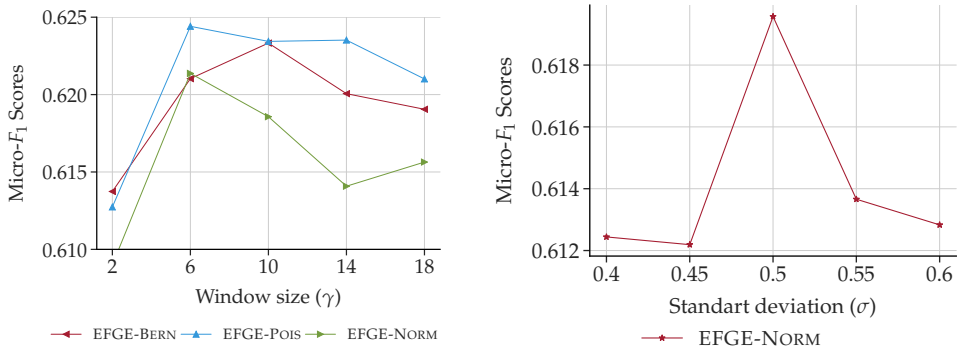


Figure 4.2: Influence of the dimension size (d) over *Citeseer* the network.

64 to 96 for EFGE-Norm, the rest of the EFGE instances show a similar characteristic between the dimension sizes 32 and 64. The impact of the dimension size is more prominent for these variants of EFGE than for the EFGE-Norm model.

THE EFFECT OF WINDOW SIZE. We examine our approach's sensitivity under various window sizes ranging from 2 to 18 over the *Citeseer* network. Figure 4.3a depicts the Micro-F_1 scores for the training set composed by 50% of the network. As we can observe, the variants of the EFGE model tend to show better performance around window size of 8. EFGE-Pois shows superior performance compared to the other models since it models the number of occurrences of nodes within a random walk sequence, and potentially are benefited by a large γ value. On the contrary, the performance of the EFGE-Bern model (which in fact captures simple co-occurrence relationships) deteriorates for large window sizes more. EFGE-Norm is the most sensitive approach to the window size since possible outlier nodes occurring in the context sets causes a significant drop in its performance.

THE EFFECT OF STANDARD DEVIATION OF EFGE-NORM MODEL. The EFGE-Norm model has an extra parameter σ which can influence the performance of the method. We examine the impact of σ, with five different values, performing experiments over the *CiteSeer* network. Figure 4.3b depicts how the Micro-F_1 scores alter for various value. The results clearly indicate that the model performs well for the value 0.5 of σ. For this reason, we have set



(a) Window size

(b) Standard deviation

Figure 4.3: Influence of the window size (γ) and standard deviation (σ) of EFGE-NORM on the *CiteSeer* network for the training set ratio of 50%.

this value for all the experiments conducted in the node classification and link prediction tasks.

4.5 CONCLUSION AND FUTURE WORK

In this chapter, we introduced exponential family graph embeddings (EFGE), proposing three instances (EFGE-BERN, EFGE-POIS and EFGE-NORM) that generalize random walk approaches to exponential families. The benefit of these models stems from the fact that they allow utilizing exponential family distributions over center-context node pairs, going beyond simple co-occurrence relationships. We have also examined how the objective functions of the models can be expressed in a way that negative sampling can be applied to scale the learning process. The experimental results have demonstrated that instances of the EFGE model can outperform widely used baseline methods. As future work, we further plan to generalize the proposed model by integrating various topological properties of the network and possible node features into the node representation learning procedure.

5

MULTIPLE KERNEL REPRESENTATION LEARNING ON GRAPHS

Matrix *factorization* and *random walk*-based methods have become popular approaches for learning representations of nodes in a lower-dimensional space. In this chapter, we aim to bring together the best of both worlds, towards learning node embeddings. In particular, we propose a weighted matrix factorization model that encodes random walk-based information about nodes of the network. The benefit of this novel formulation is that it enables us to utilize kernel functions without realizing the exact proximity matrix so that it enhances the expressiveness of existing matrix decomposition methods with kernels and alleviates their computational complexities. We further extend the approach with a multiple kernel learning formulation that provides the flexibility of learning the kernel as the linear combination of a dictionary of kernels in data-driven fashion. We perform an empirical evaluation on real-world networks, showing that the proposed model outperforms baseline node embedding algorithms in two downstream machine learning tasks.

5.1 INTRODUCTION

As we have discussed in Chapter 2, the area of GRL has been highly impacted by both traditional dimensionality reduction approaches [BN01; RS00b] and by the field of NLP [Mik+13b]. Specifically, models relying on matrix factorization techniques [BN01; CLX15; Ou+16] and random-walk based embedding models have gained considerable attention (e.g., [PARS14; GL16; ÇM20; CM20]).

In this part of the dissertation, we will examine how *kernel functions* can be used to further enhance node embeddings. Kernel functions have often been introduced along with popular learning algorithms, such as PCA

[SSM97], SVM [SS01], Spectral Clustering [DGK04], and Collaborative Filtering [Liu+16], to name a few. Most traditional learning models are insufficient to capture the underlying substructures of complex datasets, since they rely on linear techniques to model nonlinear patterns existing in data. Kernel functions [HSS08] on the other hand, allow to map non-linearly separable points into a (generally) higher dimensional feature space, so that the inner product in the new space can be computed without needing to compute the exact feature maps—bringing further computational benefits. Besides, to further reduce model bias, *multiple kernel learning* approaches have been proposed to learn optimal combinations of kernel functions [GA11]. Nevertheless, despite their wide applications in various fields [HSS08; Wan+12], kernel and multiple kernel methods have not been thoroughly investigated for learning node embeddings.

In this chapter, we aim to combine matrix factorization and random walks in a kernelized model for learning node embedding. The potential advantage of such a modeling approach is that it allows to leverage and combine the elegant mathematical formulation offered by that matrix factorization with the expressive power of random walks to capture a notion of "stochastic" node similarity in an efficient way. More importantly, this formulation enables to leverage kernel functions in the node representation learning task. Because of the nature of matrix factorization-based models, node similarities can be viewed as an inner products of vectors lying in a latent space—which allows to utilize kernels towards interpreting the embeddings in a higher dimensional feature space using non-linear maps. Besides, multiple kernels can be utilized to learn more discriminative node embeddings. Note that, although graph kernels is a well-studied topic in graph learning [KJM20], it mainly focuses on graph classification—a task outside of the scope of this chapter. To the best of our knowledge, random walk-based multiple kernel matrix factorization has not been studied before for learning node embeddings.

The main contributions of the chapter can be summarized as follows:

- We propose KernelNE (Kernel Node Embeddings), a novel approach for learning node embeddings by incorporating kernel functions with models relying on weighted matrix factorization, encoding random

walk-based structural information of the graph. We further examine the performance of the model with different types of kernel functions.

- To further improve expressiveness, we introduce MKernelNE, a multiple kernel learning formulation of the model. MKernelNE extends the kernelized weighted matrix factorization framework by learning a linear combination of a predefined set of kernels.
- We demonstrate how both models (simple and multiple kernel learning) leverage negative sampling to efficiently compute node embeddings.
- We extensively evaluate the proposed method's performance in the downstream tasks of node classification and link prediction. We show that the proposed approaches generally outperform well-known baseline methods on various real-world graph datasets. Besides, due to the efficient model optimization mechanism, the running time is comparable to the one of random walk models.

The rest of the chapter is organized as follows. We present the related works of the field in Section 5.2 and then, the problem formulation is described in Section 5.3. The proposed approach is introduced and experimental evaluations are carried out in Section 5.4. Finally, we conclude the paper in Section 5.6.

SOURCE CODE. The implementation of the proposed methodology in C++ can be reached at the following address: https://abdcelikkanat.github.io/projects/kernelNE/.

5.2 RELATED WORK

Although most algorithms in the broader field of machine learning have been developed for the linear case, real-world data often requires nonlinear models capable of unveiling the underlying complex relationships towards improving the performance on downstream tasks. To that end, kernel functions allow computing the inner product among data points in a typically high-dimensional feature space, in which linear models could still be applied,

without explicitly computing the feature maps [HSS08]. Because of the generality of the inner product, kernel methods have widely been used in a plethora of models, including Support Vector Machine [SS01], PCA [SSM97], spectral clustering [DGK04; AS10], collaborative filtering [Liu+16], and nonnegative matrix factorization for image processing tasks [FHK14], to name a few.

Undoubtedly, the most prominent application of kernels in network analysis concerns the definition of a *graph kernel* [KJM20]—a kernel function that mainly targets to measure the similarity between a pair of graphs, with direct applications in the graph classification task. Recent approaches have also been proposed to leverage graph kernels for node classification, but in a supervised manner [Tia+19]. Other related applications of kernel methods on graphs include topology inference [SBG17; RIG17], signal reconstruction [RMG17], and anomaly detection [Mat+19].

The expressiveness of kernel methods can further be enhanced using multiple kernel functions in a way that the best possible combination of a set of predefined kernels can be learned [BLJ04; GA11]. Besides improving prediction performance, multiple kernels have also be used to combine information from distinct heterogeneous sources (please see [GA11] and [WZH21] for a detailed presentation of several multiple kernel learning algorithms and their applications).

Despite the widespread applications of the kernel and multiple kernel learning methods, utilizing them for learning node embeddings via matrix factorization in an unsupervised way, is a problem that has not been thoroughly investigated. Previously, works that are close to our problem settings, which follow a methodologically different factorization approach (e.g., leveraging nonnegative matrix factorization [Liu+16]), targeting different application domains (e.g., collaborative filtering [AYC11]). The multiple kernel framework for graph-based dimensionality reduction proposed by Lin et al. [LLF11] is also close to our work. Nevertheless, their work focuses mainly on leveraging multiple kernel functions to fuse image descriptors in computer vision tasks properly.

As will be presented shortly, in this chapter, we propose novel unsupervised models for node embeddings that implicitly perform weighted matrix

decomposition in a higher-dimensional space through kernel functions. The target pairwise node proximity matrix is properly designed to leverage information from random walk sequences, and thus, our models do not require the exact realization of this matrix. Both single and multiple kernel learning formulations are studied. Emphasis is also put on optimization aspects to ensure that node embeddings are computed efficiently.

5.3 MODELING AND PROBLEM FORMULATION

Our goal is to find node representations in a latent space, preserving properties of the network. More formally, we define the general objective function of our problem as a weighted matrix factorization [SJ03], as follows:

$$\underset{\mathbf{A},\mathbf{B}}{\operatorname{argmin}} \underbrace{\left\| \mathbf{W} \odot (\mathbf{M} - \mathbf{A}\mathbf{B}^\top) \right\|_F^2}_{\text{Error term}} + \underbrace{\frac{\lambda}{2} \left(\|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2 \right)}_{\text{Regularization term}}, \tag{5.1}$$

where $\mathbf{M} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ is the target matrix constructed based on the desired properties of a given network, which is used to learn node embeddings $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{|\mathcal{V}| \times d}$, and $\| \cdot \|_F$ denotes the *Frobenius* norm. We will use $\mathcal{R}(\mathbf{A}, \mathbf{B})$ to denote the regularization term of Equation (5.1). Each element $\mathbf{W}_{(v,u)}$ of the weight matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ captures the importance of the approximation error between nodes v and u, and $\odot$ indicates the *Hadamard* product. Depending on the desired graph properties that we are interested to encode, there are many possible alternatives to choose matrix $\mathbf{M}$; such include the number of common neighbors between a pair of nodes, higher-order node proximity based on the *Adamic-Adar* or *Katz* indices [Ou+16], as well leveraging multi-hop information [CLX15]. Here, we will design $\mathbf{M}$ as a sparse binary matrix utilizing information of random walks over the network. Note that, matrices $\mathbf{M}$ and $\mathbf{W}$ do not need to be symmetric.

Let $\mathcal{W}$ be a collection of walks of length $\mathcal{L}$ and γ be the window size. We consider $\mathbf{M}_{(v,u)}$ as a binary value which equals to 1 if node u appears in the context of v in any walk. We set $\mathbf{F}_{(v,u)}$ to $2 \cdot \gamma \cdot \#(v)$, where $\#(v)$ indicates the total number of occurrences of node v in the generated walks. Setting each

term $\mathbf{W}_{(v,u)}$ as the square root of $\mathbf{F}_{(v,u)}$, the objective function in Equation (5.1) can be expressed under a random walk-based formulation, as follows:

$$
\begin{aligned}
&\underset{\mathbf{A},\mathbf{B}}{\operatorname{argmin}} \left\| \mathbf{W} \odot \left(\mathbf{M} - \mathbf{A}\mathbf{B}^\top \right) \right\|_F^2 + \mathcal{R}(\mathbf{A}, \mathbf{B}) \\
=&\underset{\mathbf{A},\mathbf{B}}{\operatorname{argmin}} \left\| \sqrt{\mathbf{F}} \odot \left(\mathbf{M} - \mathbf{A}\mathbf{B}^\top \right) \right\|_F^2 + \mathcal{R}(\mathbf{A}, \mathbf{B}) \\
=&\underset{\mathbf{A},\mathbf{B}}{\operatorname{argmin}} \sum_{v \in \mathcal{V}} \sum_{u \in \mathcal{V}} \mathbf{F}_{(v,u)} \left(\mathbf{M}_{(v,u)} - \left\langle \mathbf{A}[u], \mathbf{B}[v] \right\rangle \right)^2 + \mathcal{R}(\mathbf{A}, \mathbf{B}) \\
=&\underset{\mathbf{A},\mathbf{B}}{\operatorname{argmin}} \sum_{\mathbf{w} \in \mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{j=-\gamma \\ j \neq 0}}^{\gamma} \sum_{u \in \mathcal{V}} \left(\mathbb{1}_{\{w_{l+j}\}}(u) - \left\langle \mathbf{A}[u], \mathbf{B}[w_l] \right\rangle \right)^2 + \mathcal{R}(\mathbf{A}, \mathbf{B}), \quad (5.2)
\end{aligned}
$$

where each $\mathbf{w} = (w_1, \ldots, w_l, \ldots, w_{\mathcal{L}}) \in \mathcal{V}^{\mathcal{L}}$ indicates a random walk of length $\mathcal{L}$ in the collection $\mathcal{W}$, and $\mathcal{R}(\mathbf{A}, \mathbf{B})$ is the regularization term. As mentioned in the previous chapters, matrix $\mathbf{A}$ in Equation (5.2) indicates the embedding vectors of nodes when they are considered as *contexts*; those will be the embeddings that are used in the experimental evaluation. The choice of matrix $\mathbf{M}$ and the reformulation of the objective function, offers a computational advantage during the optimization step. Moreover, such formulation also allows us to further explore kernelized version in order to exploit possible nonlinearity of the model.

5.4 KERNEL-BASED NODE REPRESENTATION LEARNING

Most matrix factorization techniques that aim to find latent low-dimensional representations (e.g., [Qiu+18; Qiu+19; Ou+16]), adopt the *Singular Value Decomposition* (SVD); it provides the best approximation of the objective function stated in Equation (5.1), as long as the weight matrix is uniform [EY36]. Nevertheless, in our case the weight matrix is not uniform, therefore we need the exact realization of the target matrix in order to perform SVD. To overcome this limitation, we leverage kernel functions to learn node representations via matrix factorization.

Let (X, d_X) be a metric space and $\mathbb{H}$ be a Hilbert space of real-valued functions defined on X. A Hilbert space is called *reproducing kernel Hilbert*

space (RKHS) if the point evaluation map over $\mathbb{H}$ is a continuous linear functional. Furthermore, a *feature map* is defined as a function $\Phi : \mathsf{X} \to \mathbb{H}$ from the input space X into *feature space* $\mathbb{H}$. Every feature map defines a *kernel* $\mathsf{K} : \mathsf{X} \times \mathsf{X} \to \mathbb{R}$ as follows:

$$\mathsf{K}(\mathbf{x}, \mathbf{y}) := \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle \qquad \forall (\mathbf{x}, \mathbf{y}) \in \mathsf{X}^2.$$

It can be seen that $\mathsf{K}(\cdot, \cdot)$ is symmetric and positive definite due to the properties of an inner product space.

A function $g : \mathsf{X} \to \mathbb{R}$ is *induced by* K, if there exists $h \in \mathbb{H}$ such that $g = \langle h, \Phi(\cdot) \rangle$ for a feature vector Φ of kernel K. Note that, this is independent of the definition of the feature map Φ and space $\mathbb{H}$ [Steo2]. Let $\mathcal{I}_\kappa := \{g : \mathsf{X} \to \mathbb{R} \mid \exists h \in \mathbb{H} \text{ s.t. } g = \langle h, \Phi(\cdot) \rangle\}$ be the set of induced functions by kernel K. Then, a continuous kernel K on a compact metric space $(\mathsf{X}, d_\mathsf{X})$ is *universal*, if the set $\mathcal{I}_\kappa$ is dense in the space of all continuous real-valued functions $\mathcal{C}(\mathsf{X})$. In other words, for any function $f \in \mathcal{C}(\mathsf{X})$ and $\epsilon > 0$, there exists $g_h \in \mathcal{I}_\kappa$ satisfying

$$\|f - g_h\|_\infty \leq \epsilon,$$

where g_h is defined as $\langle h, \Phi(\cdot) \rangle$ for some $h \in \mathbb{H}$.

In this chapter, we consider universal kernels, since we can always find $h \in \mathbb{H}$ satisfying $|\langle h, \phi(x_i) \rangle - \alpha_i| \leq \epsilon$ for given $\{x_1, \ldots, x_N\} \subset \mathsf{X}$, $\{\alpha_1, ..., \alpha_N\} \subset \mathbb{R}$ and $\epsilon > 0$ by Proposition 5.1. If we choose α_i's as the entries of a row of our target matrix $\mathbf{M}$, then the elements h and $\phi(x_i)$ indicate the corresponding column vectors of $\mathbf{A}$ and $\mathbf{B}$, respectively. Then, we can obtain a decomposition of the target matrix by repeating the process for each row. However, the element h might not always be in the range of feature map Φ; in this case, we will approximate the correct values of $\mathbf{M}$.

Proposition 5.1 (Universal kernels [Steo2])**.** *Let* $(\mathsf{X}, d_\mathsf{X})$ *be a compact metric space and* $\mathsf{K}(\cdot, \cdot)$ *be a universal kernel on* X*. Then, for all compact and mutually*

disjoint subsets $\mathsf{S}_1, \ldots, \mathsf{S}_n \subset \mathsf{X}$, *all* $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$, *and all* $\epsilon > 0$, *there exists a function* g *induced by* K *with* $\|g\|_\infty \leq \max_i |\alpha_i| + \epsilon$ *such that*

$$\|g_{|\mathsf{S}} - \sum_{i=1}^{n} \alpha_i \mathbb{1}_{\mathsf{S}_i}\|_\infty \leq \epsilon,$$

where $\mathsf{S} := \bigcup_{i=1}^{n} \mathsf{S}_i$ *and* $g_{|\mathsf{S}}$ *is the restriction of* g *to* S.

Universal kernels also provide a guarantee for the injectivity of the feature maps, as shown in Lemma 5.1; therefore, we can always find $y \in \mathsf{X}$, such that $\Phi(y) = h$ if $h \in \Phi(\mathsf{X})$. Otherwise, we can learn an approximate pre-image solution by using a gradient descent technique [HR11].

Lemma 5.1 ([Ste02]). *Every feature map of a universal kernel is injective.*

Proof. Let $\Phi(\cdot) : \mathsf{X} \to \mathbb{H}$ be a feature vector of the kernel $\mathsf{K}(\cdot, \cdot)$. Assume that Φ is not injective, so we can find a pair of distinct elements $x, y \in \mathsf{X}$ such that $\Phi(x) = \Phi(y)$ and $x \neq y$. By Proposition 5.1, for any given $\epsilon > 0$, there exists a function $g = \langle h, \Phi(\cdot) \rangle$ induced by K for some $h \in \mathbb{H}$, which satisfies

$$\left\| g_{|S} - (\mathbb{1}_{S_1} - \mathbb{1}_{S_2}) \right\|_\infty \leq \epsilon,$$

where $\mathsf{S} := \mathsf{S}_1 \cup \mathsf{S}_2$ for the compact sets $\mathsf{S}_1 = \{x\}$ and $\mathsf{S}_2 = \{y\}$. Then, we have that $|\langle \Phi(x), h \rangle - \langle \Phi(y), h \rangle)| \geq 2 - 2\epsilon$. In other words, we obtain $\Phi(x) \neq \Phi(y)$, which contradicts our initial assumption. □

5.4.1 *Single Kernel Node Representation Learning*

Following the kernel formulation described above, we can now perform matrix factorization in the feature space by leveraging kernel functions. In particular, we can move the inner product from the input space X to the feature space $\mathbb{H}$, by reformulating Equation (5.2) as follows:

$$\underset{\mathbf{A},\mathbf{B}}{\operatorname{argmin}} \sum_{\mathbf{w} \in \mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{j=-\gamma \\ j \neq 0}}^{\gamma} \sum_{u \in \mathcal{V}} \left(\mathbb{1}_{\{w_{l+j}\}}(u) - \left\langle \Phi(\mathbf{A}[u]), \Phi(\mathbf{B}[w_l]) \right\rangle \right)^2 + \mathcal{R}(\mathbf{A}, \mathbf{B})$$

$$= \underset{\mathbf{A},\mathbf{B}}{\operatorname{argmin}} \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{j=-\gamma \\ j\neq 0}}^{\gamma} \sum_{u\in\mathcal{V}} \Big(\mathbb{1}_{\{w_{l+j}\}}(u) - \mathsf{K}\big(\mathbf{A}[u], \mathbf{B}[w_l]\big) \Big)^2 + \mathcal{R}(\mathbf{A}, \mathbf{B}). \quad (5.3)$$

To this end, we obtain a kernelized matrix factorization model for node embeddings based on random walks. For the numerical evaluation of our method, we use the following universal kernels [MXZ06; Ste02]:

$$\mathsf{K}_G(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{-\|\mathbf{x}-\mathbf{y}\|^2}{\sigma^2} \right) \qquad \sigma \in \mathbb{R}$$

$$\mathsf{K}_S(\mathbf{x}, \mathbf{y}) = \frac{1}{\left(1 + \|\mathbf{x}-\mathbf{y}\|^2\right)^{\sigma}} \qquad \sigma \in \mathbb{R}_+,$$

where K_G and K_S correspond to the *Gaussian* and *Schoenberg* kernels respectively. We will refer to the proposed kernel-based node embeddings methodology as KernelNE (the two different kernels will be denoted by Gauss and Sch). A schematic representation of the basic components of the proposed model is given in Figure 5.1.

5.4.1.1 *Model optimization*

The estimation problem for both parameters $\mathbf{A}$ and $\mathbf{B}$ is, unfortunately, non-convex. Nevertheless, when we consider each parameter separately by fixing the other one, it turns into a convex problem. By taking advantage of this property, we employ Stochastic Gradient Descent (SGD) [Bot91] in the optimization step of each embedding matrix. Note that, for each center node $w_l \in \mathcal{V}$ in Equation (5.3), we have to compute the gradient for each $u \in \mathcal{V}$, which is computationally intractable. However, Equation (5.3) can be divided into two parts concerning the values of $\mathbb{1}_{\{w_{l+j}\}}(u) \in \{0, 1\}$, as follows:

$$\begin{aligned} &\sum_{u\in\mathcal{V}} \Big(\mathbb{1}_{\{w_{l+j}\}}(u) - \mathsf{K}\big(\mathbf{A}[u], \mathbf{B}[w_l]\big) \Big)^2 \\ &= \Big(1 - \mathsf{K}\big(\mathbf{A}[w_{l+j}], \mathbf{B}[w_l]\big)\Big)^2 + \sum_{s^-\in\mathcal{V}\setminus\{w_{l+j}\}} \Big(\mathsf{K}\big(\mathbf{A}[s^-], \mathbf{B}[w_l]\big) \Big)^2 \end{aligned}$$

$$\approx \Big(1 - \mathsf{K}\big(\mathbf{A}[w_{l+j}], \mathbf{B}[w_l]\big)\Big)^2 + |\mathcal{V}\backslash\{w_{l+j}\}| \mathop{\mathbb{E}}_{s^- \sim p^-} \Big[\mathsf{K}\big(\mathbf{A}[s^-], \mathbf{B}[w_l]\big)\Big]^2$$

$$= \Big(\underbrace{\big(1 - \mathsf{K}\big(\mathbf{A}[w_{l+j}], \mathbf{B}[w_l]\big)\big)^2}_{\text{positive sample}} + \underbrace{k \mathop{\mathbb{E}}_{s^- \sim p^-} \Big[\mathsf{K}\big(\mathbf{A}[s^-], \mathbf{B}[w_l]\big)\Big]^2}_{\text{negative sample}},$$

where k is defined as $|\mathcal{V}| - 1$. That way, we can apply *negative sampling* [Mik+13b]. For each center node w_l, we sample k negative instances s^- from the noise distribution p^-. Then, we can rewrite the objective function of Equation (5.3) in the following way:

$$\mathcal{F}_S := \operatorname*{argmin}_{\mathbf{A},\mathbf{B}} \sum_{\mathbf{w} \in \mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{j=-\gamma \\ j \neq 0}}^{\gamma} \Bigg(\big(1 - \mathsf{K}\big(\mathbf{A}[w_{l+j}], \mathbf{B}[w_l]\big)^2 + \sum_{\substack{r=1 \\ s_r^- \sim p^-}}^{k} \mathsf{K}\big(\mathbf{A}[s_r^-], \mathbf{B}[w_l]\big)^2 \Bigg) + \mathcal{R}(\mathbf{A}, \mathbf{B}). \tag{5.4}$$

Equation (5.4) corresponds to the objective function of the proposed KERNELNE model. In the following subsection, we will examine how this model could be further extended to leverage multiple kernels.

5.4.2 *Multiple Kernel Node Representation Learning*

Selecting a proper kernel function $\mathsf{K}(\cdot, \cdot)$ and the corresponding parameters (e.g., the bandwidth of a Gaussian kernel) is a critical task during the learning phase. Nevertheless, choosing a single kernel function might impose potential bias, causing limitations on the performance of the model. Having the ability to properly utilize multiple kernels could increase the expressiveness of the model, capturing different notions of similarity among embeddings [GA11]. Besides, learning how to combine such kernels, might further improve the performance of the underlying model. In particular, given a set of base kernels $\{\mathsf{K}_j\}_{j=1}^K$, we aim to find an optimal way to combine the given kernels, as follows:

$$\mathsf{K}^{\mathbf{c}}(\mathbf{x}, \mathbf{y}) = f_{\mathbf{c}}\big(\{\mathsf{K}_i(\mathbf{x}, \mathbf{y})\}_{i=1}^K | \mathbf{c}\big),$$

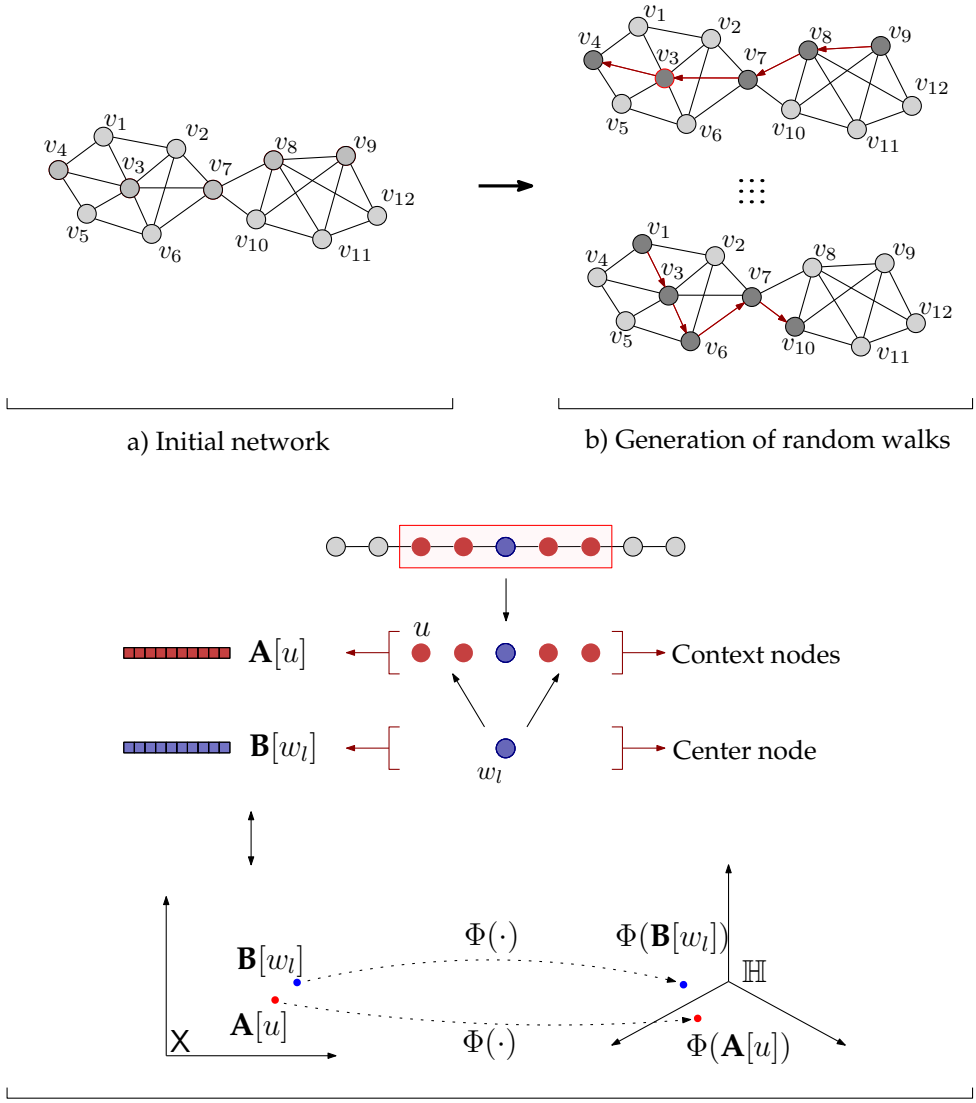


Figure 5.1: Schematic representation of the KernelNE model. A set of node sequences is firstly generated by following a random walk strategy. By using the co-occurrences of node pairs within a certain window size, node representations are learned by optimizing their maps in the feature space.

where the combination function $f_{\mathbf{c}}$ is parameterized on $\mathbf{c} \in \mathbb{R}^K$ that indicates kernel weights. Due to the generality of the multiple kernel learning framework, $f_{\mathbf{c}}$ can be either a linear or nonlinear function.

In this paragraph, we examine how to further strengthen the proposed kernelized weighted matrix factorization, by *linearly* combining multiple kernels. More precisely, let $\mathsf{K}_1, \ldots, \mathsf{K}_K$ be a set of kernel functions satisfying the properties presented previously. Then, we restate the objective as follows:

$$\begin{aligned}\mathcal{F}_M := \underset{\mathbf{A},\mathbf{B},\mathbf{c}}{\operatorname{argmin}} \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{\mathcal{L}} \sum_{\substack{j=-\gamma \\ j\neq 0}}^{\gamma} \Bigg(& \Big(1 - \sum_{i=1}^{K} c_i \mathsf{K}_i(\mathbf{A}[w_{l+j}], \mathbf{B}[w_l])\Big)^2 \\ & + \sum_{\substack{r=1 \\ s_r^- \sim p^-}}^{k} \Big(\sum_{i=1}^{K} c_i \mathsf{K}_i(\mathbf{A}[s_r^-], \mathbf{B}[w_l])\Big)^2 \Bigg) \\ & + \frac{\lambda}{2}\left(\|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2\right) + \frac{\beta}{2}\|\mathbf{c}\|_2^2, \end{aligned} \tag{5.5}$$

where $\mathbf{c} = [c_1, \ldots, c_K]^\top \in \mathbb{R}^K$. Here, we introduce an additional parameter c_j representing the contribution of the corresponding kernel K_i. $\beta > 0$ is a tradeoff parameter for the regularization term, and similarly, the coefficients c_1,...,c_K are optimized by fixing the remaining model parameters $\mathbf{A}$ and $\mathbf{B}$. Equation (5.5) corresponds to the objective function of the proposed multiple kernel learning model MKernelNE. Unlike the common usage of multiple kernels [GA11], we do not constrain the coefficients to non-negative values. We interpret each entry of the target matrix as a linear combination of inner products of different kernels' feature maps. As discussed in the previous sections, our main intuition relies on obtaining more expressive embeddings by moving the factorization step into a higher dimensional space.

Algorithm 5.1 provides the pseudocode of the proposed approach. For a given collection of random walks $\mathcal{W}$, we firstly determine the center-context node pairs (w_l, w_{l+j}) in the node sequences. Recall that, for each *center* node in a walk, its surrounding nodes within a certain distance γ, define its *context*. Furthermore, the corresponding embedding vectors $\mathbf{B}[w_l]$ of center node w_l and $\mathbf{A}[w_{l+j}]$ of context w_{l+j} are updated by following the rules which we describe in detail below. Note that, we obtain two different representations,

$\mathbf{A}[v]$ and $\mathbf{B}[v]$, for each node $v \in \mathcal{V}$ since the node can have either a center or context role in the walks. The gradients in Algorithm 5.2 are given below. For notation simplicity, we denote each $\mathsf{K}_i(\mathbf{A}[u], \mathbf{B}[v])$ by the term $\mathsf{K}_i(u, v)$.

$$\begin{aligned}
\nabla_{\mathbf{A}[x]} \mathcal{F}_M &:= \mathbb{1}_{\{u\}}(x)\Big(-2\sum_{i=1}^{K} c_i \nabla_{\mathbf{A}[u]} \mathsf{K}_j(u,v)\Big)\Big(1-\sum_{i=1}^{K} c_i \mathsf{K}_i(u,v)\Big) + \\
&\quad 2\sum_{\substack{r=1 \\ s_r^- \sim p^-}}^{k} \mathbb{1}_{\{s_r^-\}}(x)\Big(\sum_{i=1}^{K} c_i \nabla_{\mathbf{A}[s_r^-]} \mathsf{K}_i(s_r^-, v)\mathsf{K}_i(s_r^-, v)\Big) + \lambda \mathbf{A}[x] \\
\nabla_{\mathbf{B}[v]} \mathcal{F}_M &:= -2\Big(\sum_{i=1}^{K} c_i \nabla_{\mathbf{B}[v]} \mathsf{K}_i(u,v)\Big)\Big(1-\sum_{i=1}^{K} c_i \mathsf{K}_i(u,v)\Big) + \\
&\quad 2\sum_{\substack{r=1 \\ s_r^- \sim p^-}}^{k} \Big(\sum_{i=1}^{K} c_i \nabla_{\mathbf{B}[v]} \mathsf{K}_i(s_r^-, v)\mathsf{K}_i(s_r^-, v)\Big) + \lambda \mathbf{B}[v] \\
\nabla_{c_t} \mathcal{F}_M &= -2\Big(1-\sum_{i=1}^{K} c_i \mathsf{K}_i(u,v)\Big)\mathsf{K}_t(u,v) + \\
&\quad 2\sum_{\substack{r=1 \\ s_r^- \sim p^-}}^{k} \Big(\sum_{i=1}^{K} c_i \mathsf{K}_i(s_r^-, v)\Big)\mathsf{K}_t(s_r^-, v) + \beta c_t
\end{aligned}$$

5.4.3 *Complexity Analysis*

For the generation of walks, we apply the biased random walk strategy proposed in NODE2VEC, which can be performed in $\mathcal{O}(|\mathcal{W}| \cdot \mathcal{L})$ steps [GL16] for the precomputed transition probabilities, where $\mathcal{L}$ indicates the walk length and $\mathcal{W}$ denotes the set of walks. For the algorithm's learning procedure, we can carry out Line 5 of Algorithm 5.1 at most $2\gamma \cdot |\mathcal{W}| \cdot \mathcal{L}$ times for each center-context pair, where γ represents the window size. The dominant operation in Algorithm 5.2 is the multiplication operation of the update rules in Lines 6, 7, and 9; the running time can be bounded by $\mathcal{O}(k \cdot K \cdot d)$, where k is the number of negative samples generated per center-context pair, K is the number of kernels, and d is the representation size. To sum up, the overall running time of the proposed approach can be bounded by $\mathcal{O}(\gamma \cdot |\mathcal{W}| \cdot \mathcal{L} \cdot K \cdot d \cdot k)$ steps.

Algorithm 5.1 MKernelNE

Input: Graph: $G = (\mathcal{V}, \mathcal{E})$
Representation size: d
Set of walks: $\mathcal{W}$
Window size: γ
Kernel function: K
Kernel parameter(s): σ
Output: Embedding matrix: $\mathbf{A}$

Initialize matrices: $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{|\mathcal{V}| \times d}$
/* Extract center-context node pairs */
for all $\mathbf{w} = (w_1, \ldots, w_{\mathcal{L}}) \in \mathcal{W}$ **do**
  for $l \leftarrow 1$ **to** $\mathcal{L}$ **do**
    for $j \neq 0 \leftarrow -\gamma$ **to** γ **do**
      $\mathbf{A}, \mathbf{B}, \mathbf{c} \leftarrow$ UpdateEmb$(\mathbf{A}, \mathbf{B}, \mathbf{c}, w_l, w_{l+j}, \mathsf{K}, \sigma)$
    end for
  end for
end for

5.5 EXPERIMENTAL EVALUATION

In this section, we assess the performance of the proposed approach by following the same experimental setup and network datasets that we have employed in the previous sections.

5.5.1 *Parameter Settings*

The different instances of KernelNE and MKernelNE are fed with random walks similar to those used in Node2Vec, setting hyper-parameters p, q to 1.0. For the training process, we adopt the negative sampling strategy [Mik+13a] as described in Subsection 5.4.1. For the kernel parameters, the value of σ has been chosen as 2.0 for the single kernel version of the model (KernelNE). For MKernelNE, we have considered three kernels and their parameters are set to $1.0, 2.0, 3.0$ and $1.0, 1.5, 2.0$ for MkernelNE-Gauss and MkernelNE-Sch, respectively. The regularization parameters are set to $\lambda = 10^{-2}$ and $\beta = 0.1$.

Algorithm 5.2 UPDATEEMB

Input: Embedding matrices: $\mathbf{A}$ and $\mathbf{B}$
Kernel coefficients: $\mathbf{c}$
Kernel function(s): K
Center and context nodes: v and u
Learning rate: η
Distribution for generating negative samples: p^-
Output: Embedding matrix: $\mathbf{A}$

node_list $\leftarrow [u]$
/* Extract negative samples */
for $s \leftarrow 1$ **to** k **do**
  node_list $\leftarrow$ SAMPLENODE(p^-)
end for
/* Update embedding vectors */
for each x in node_list **do**
  $\mathbf{A}[x] \leftarrow \mathbf{A}[x] - \eta \nabla_{\mathbf{A}[x]} \mathcal{F}_M$
  $\mathbf{B}[v] \leftarrow \mathbf{B}[v] - \eta \nabla_{\mathbf{B}[v]} \mathcal{F}_M$
  /* Update individual kernel weights */
  if number of kernels > 1 **then**
    $\mathbf{c} \leftarrow \mathbf{c} - \eta \nabla_c \mathcal{F}_M$
  end if
end for

5.5.2 *Node Classification*

EXPERIMENTAL SETUP. In the node classification task, we have access to the labels of a certain fraction of nodes in the network (training set), and our goal is to predict the labels of the remaining nodes (test set). The experiments are carried out by applying an one-vs-rest logistic regression classifier with L_2 regularization [Ped+11].

EXPERIMENTAL RESULTS. Tables 5.1 to 5.4 report the Micro-F_1 and Macro-F_1 scores of the classification task. With boldface and underline, we indicate the best and second-best performing model, respectively. As we can observe, the single and multiple kernel versions of the proposed methodology outperform the baseline models, showing different characteristics depending on the graph dataset. While the *Schoenberg* kernel comes into prominence on *PPI*

		2%	4%	6%	8%	10%	30%	50%	70%	90%
Baselines	DeepWalk	0.421	0.462	0.488	0.502	0.517	0.569	0.587	0.596	0.597
		0.374	0.419	0.446	0.461	0.476	0.524	0.540	0.547	0.546
	Node2Vec	0.451	0.491	0.514	0.529	0.543	0.583	0.595	0.600	0.603
		0.397	0.443	0.466	0.483	0.497	0.535	0.546	0.549	0.550
	LINE	0.300	0.353	0.384	0.407	0.418	0.476	0.494	0.505	0.512
		0.243	0.303	0.334	0.357	0.367	0.423	0.440	0.448	0.454
	HOPE	0.197	0.204	0.207	0.208	0.216	0.253	0.276	0.299	0.316
		0.060	0.065	0.066	0.068	0.075	0.121	0.150	0.178	0.202
	NetMF	0.401	0.473	0.507	0.525	0.538	0.579	0.590	0.594	0.601
		0.346	0.421	0.454	0.474	0.487	0.528	0.538	0.542	0.548
	GEMSEC	0.384	0.439	0.459	0.479	0.491	0.532	0.545	0.552	0.558
		0.339	0.400	0.422	0.439	0.451	0.488	0.497	0.499	0.501
	M-NMF	0.264	0.317	0.340	0.370	0.382	0.439	0.449	0.456	0.457
		0.161	0.229	0.261	0.293	0.306	0.370	0.381	0.390	0.390
KernelNE	Gauss	0.465	0.504	0.528	0.543	0.555	**0.600**	0.611	0.616	**0.620**
		0.415	0.456	0.479	0.496	0.508	**0.551**	**0.562**	**0.566**	**0.566**
	Sch	0.497	0.531	0.544	0.555	0.561	0.591	0.602	0.610	0.609
		0.428	0.465	0.480	0.492	0.498	0.531	0.542	0.549	0.546
MKernelNE	Gauss	0.493	0.530	0.547	0.559	0.566	**0.600**	**0.613**	**0.619**	**0.620**
		0.434	0.473	**0.491**	**0.504**	**0.511**	0.546	0.559	**0.566**	0.565
	Sch	**0.500**	**0.538**	**0.553**	**0.563**	**0.570**	0.597	0.609	0.614	0.615
		0.433	**0.475**	0.490	0.500	0.508	0.537	0.550	0.557	0.556

Table 5.1: Node classification on *Citeseer* for varying training sizes. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

network, the *Gaussian* kernel shows good performance for the rest networks. We further observe that leveraging multiple kernels with MKernelNE often has superior performance compared to the single kernel model, especially for smaller training ratios. For instance, MKernelNE-Gauss provides an increase in the Micro-F_1 score ranging from 2.15% to 6.85% and improvement in the Macro-F_1 scores varying from 2.62% to 5.67% on the *DBLP* network. In general, we have observed that the integration of Gaussian kernel exhibits higher performance than the Schoenberg kernel in the classification experiments.

5.5.3 *Link Prediction*

EXPERIMENTAL SET-UP. As we have described in the previous chapters, for the link prediction task, we remove half of the network's edges by retaining its connectivity, and we learn node embedding on the residual network. For networks consisting of disconnected components, we consider

		2%	4%	6%	8%	10%	30%	50%	70%	90%
Baselines	DeepWalk	0.617	0.688	0.715	0.732	0.747	0.799	0.815	0.825	0.832
		0.569	0.664	0.698	0.717	0.735	0.788	0.806	0.815	0.821
	Node2Vec	0.659	0.720	0.743	0.759	0.770	0.816	0.831	0.839	0.845
		0.612	0.694	0.724	0.743	0.755	0.804	0.820	0.828	0.832
	LINE	0.416	0.498	0.546	0.581	0.609	0.701	0.728	0.741	0.747
		0.306	0.425	0.492	0.543	0.578	0.691	0.720	0.733	0.738
	HOPE	0.284	0.301	0.301	0.302	0.302	0.302	0.302	0.304	0.306
		0.067	0.066	0.067	0.066	0.066	0.067	0.067	0.068	0.074
	NetMF	0.636	0.716	0.748	0.767	0.773	**0.821**	0.834	0.841	0.844
		0.591	0.694	0.731	0.751	0.760	**0.811**	0.824	0.832	0.835
	GEMSEC	0.470	0.530	0.568	0.587	0.601	0.674	0.714	0.735	0.744
		0.406	0.477	0.527	0.546	0.562	0.646	0.689	0.713	0.722
	M-NMF	0.507	0.580	0.622	0.642	0.656	0.717	0.732	0.736	0.742
		0.459	0.550	0.598	0.622	0.638	0.706	0.722	0.728	0.734
KernelNE	Gauss	0.696	0.739	0.759	0.772	0.780	0.820	**0.837**	**0.846**	**0.851**
		0.664	0.721	0.743	0.758	0.767	0.809	**0.826**	**0.836**	**0.840**
	Sch	0.695	0.736	0.750	0.759	0.765	0.790	0.800	0.806	0.812
		0.631	0.697	0.721	0.734	0.745	0.780	0.790	0.795	0.799
MKernelNE	Gauss	**0.701**	**0.748**	**0.764**	**0.775**	**0.781**	0.812	0.823	0.828	0.833
		0.656	**0.723**	**0.746**	**0.761**	**0.769**	0.801	0.813	0.818	0.820
	Sch	0.699	0.742	0.757	0.767	0.772	0.797	0.806	0.812	0.818
		0.637	0.708	0.733	0.749	0.756	0.787	0.796	0.802	0.804

Table 5.2: Node classification on *Cora* for varying training sizes. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

		2%	4%	6%	8%	10%	30%	50%	70%	90%
Baselines	DeepWalk	0.550	0.588	0.603	0.611	0.616	0.630	0.633	0.635	0.636
		0.496	0.527	0.540	0.547	0.551	0.563	0.565	0.566	0.567
	Node2Vec	0.575	0.599	0.612	0.618	0.623	0.636	0.639	0.641	0.641
		0.517	0.541	0.551	0.557	0.561	0.571	0.574	0.575	0.574
	LINE	0.553	0.576	0.585	0.590	0.594	0.606	0.610	0.611	0.611
		0.469	0.498	0.510	0.517	0.522	0.536	0.540	0.541	0.541
	HOPE	0.379	0.379	0.379	0.379	0.379	0.380	0.383	0.387	0.391
		0.137	0.137	0.137	0.137	0.138	0.140	0.146	0.152	0.160
	NetMF	0.577	0.589	0.596	0.601	0.605	0.617	0.620	0.623	0.623
		0.490	0.506	0.513	0.518	0.522	0.530	0.531	0.533	0.533
	GEMSEC	0.538	0.566	0.583	0.591	0.597	0.611	0.614	0.615	0.615
		0.477	0.501	0.513	0.519	0.523	0.534	0.535	0.537	0.536
	M-NMF	0.501	0.527	0.540	0.545	0.551	0.568	0.574	0.577	0.579
		0.345	0.383	0.400	0.410	0.418	0.443	0.450	0.454	0.455
KernelNE	Gauss	0.575	0.595	0.606	0.613	0.617	0.630	0.633	0.635	0.636
		0.517	0.536	0.545	0.551	0.554	0.564	0.566	0.567	0.569
	Sch	0.609	0.617	0.621	0.624	0.627	0.635	0.637	0.638	0.640
		0.531	0.546	0.552	0.556	0.559	0.568	0.570	0.571	0.572
MKernelNE	Gauss	0.614	**0.624**	**0.630**	**0.633**	**0.636**	**0.646**	**0.648**	**0.650**	**0.650**
		0.547	**0.560**	**0.566**	**0.570**	**0.573**	**0.582**	**0.583**	**0.584**	**0.584**
	Sch	**0.615**	0.621	0.625	0.628	0.631	0.639	0.641	0.643	0.643
		0.538	0.551	0.557	0.561	0.564	0.572	0.574	0.576	0.576

Table 5.3: Node classification on *DBLP* for varying training sizes. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

		2%	4%	6%	8%	10%	30%	50%	70%	90%
Baselines	DeepWalk	0.110	0.131	0.143	0.151	0.156	0.188	0.206	0.218	0.226
		0.076	0.099	0.113	0.123	0.129	0.164	0.181	0.190	0.192
	Node2Vec	0.115	0.136	0.148	0.157	0.164	0.196	0.211	0.221	0.226
		0.078	**0.101**	**0.116**	**0.125**	**0.132**	0.167	0.180	0.188	0.190
	LINE	0.109	0.133	0.149	0.162	0.170	0.210	0.226	0.235	0.242
		0.063	0.084	0.099	0.111	0.120	0.164	0.181	0.191	0.192
	HOPE	0.068	0.069	0.069	0.070	0.069	0.071	0.077	0.086	0.093
		0.019	0.019	0.019	0.019	0.018	0.020	0.025	0.030	0.033
	NetMF	0.085	0.100	0.116	0.126	0.131	0.176	0.193	0.203	0.211
		0.045	0.064	0.080	0.090	0.095	0.137	0.152	0.161	0.160
	GEMSEC	0.078	0.082	0.085	0.087	0.089	0.112	0.131	0.143	0.151
		0.059	0.063	0.067	0.070	0.073	0.098	0.113	0.122	0.125
	M-NMF	0.097	0.112	0.119	0.123	0.128	0.143	0.153	0.162	0.168
		0.071	0.089	0.097	0.103	0.108	0.125	0.135	0.141	0.141
KernelNE	Gauss	0.102	0.121	0.134	0.142	0.148	0.180	0.192	0.200	0.203
		0.051	0.067	0.078	0.086	0.092	0.125	0.138	0.146	0.147
	Sch	0.126	0.154	0.172	0.185	**0.195**	**0.232**	**0.244**	**0.250**	**0.256**
		0.069	0.091	0.107	0.119	0.128	0.171	**0.187**	**0.196**	**0.198**
MKernelNE	Gauss	**0.128**	**0.156**	**0.174**	**0.186**	**0.195**	0.231	0.242	0.249	0.254
		0.071	0.094	0.109	0.122	0.131	**0.172**	**0.187**	0.195	0.197
	Sch	**0.128**	**0.156**	0.173	**0.186**	0.194	0.230	0.241	0.247	0.251
		0.071	0.094	0.110	0.122	0.130	0.169	0.184	0.192	0.193

Table 5.4: Node classification on *PPI* for varying training sizes. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

the giant component among them as our initial graph. The removed edges constitute the positive samples for the testing set; the same number of node pairs that do not exist in the original graph is sampled at random to form the negative samples. Then, the entries of the feature vector corresponding to each node pair (u, v) in the test set are computed based on various element-wise operations that we have described in Chapter 2. In the experiments, we apply logistic regression with L_2 regularization.

EXPERIMENTAL RESULTS. Table 5.5 presents the *Area Under Curve* (AUC) scores for the link prediction task. In the case of the single kernel model KernelNE, the Schoenberg kernel (KernelNE-Sch) performs significantly better than the Gaussian one. It has higher scores ranging from 0.74% to 9.28% on most of the networks except *PPI*, *Facebook*, and *Gnutella*. We observe that MKernelNE provides a performance gain of up to 0.12% for the Schoenberg kernel and up to 5.29% for the Gaussian kernel. Since the single kernel model, KernelNE-Sch is the best performing approach for

	Baselines							KernelNE		MKernelNE	
	DeepWalk	Node2Vec	LINE	HOPE	NetMF	GEMSEC	M-NMF	Gauss	Sch	Gauss	Sch
Citeseer	0.828	0.827	0.739	0.744	0.780	0.749	0.751	0.807	**0.882**	0.850	0.863
Cora	0.779	0.781	0.696	0.712	0.745	0.734	0.714	0.774	**0.823**	0.802	0.810
DBLP	0.944	0.944	0.930	0.873	0.910	0.920	0.891	0.950	0.960	**0.961**	0.960
PPI	0.860	0.861	0.878	0.880	0.785	0.771	**0.886**	0.846	0.801	0.768	0.788
AstroPh	0.961	0.961	0.969	0.931	0.872	0.816	0.948	0.963	0.970	**0.979**	0.970
HepTh	0.896	0.896	0.844	0.836	0.858	0.823	0.861	0.899	**0.917**	**0.917**	0.916
Facebook	0.984	0.983	0.967	0.975	0.974	0.733	0.977	**0.990**	0.988	0.989	0.989
Gnutella	0.695	0.694	**0.839**	0.748	0.654	0.639	0.755	0.816	0.673	0.653	0.671

Table 5.5: Area Under Curve (AUC) scores for link prediction.

most networks, the impact of adopting multiple kernels for the Gaussian case is more distinctive than the Schoenberg kernel. The baselines surpass the performance of our proposed models only on the *PPI* and *Gnutella* networks. Although KernelNE and MKernelNE methods attain their highest scores for the *Weighted L2* operator that we have described in Chapter 2, KernelNE-Gauss achieves a remarkable score with the average metric on *Gnutella* network. Overall, both the KernelNE and MKernelNE approaches perform well across different datasets compared to the rest of baseline models.

5.5.4 *Parameter Sensitivity*

In this subsection, we examine how the performance of the proposed models is affected with respect to the choice of parameters.

THE EFFECT OF DIMENSION SIZE. The dimension size is a critical parameter for node representation learning approaches since the desired properties of networks are aimed to be preserved in a lower-dimensional space. Figure 5.2 depicts the Micro-F_1 score of the proposed models for varying embedding dimension sizes, ranging from $d = 32$ up to $d = 224$ over the *Citeseer* network. As it can be seen, all the different node embedding instances, both single and multiple kernel ones, have the same tendency with respect to d; the

performance increases proportionally to the size of the embedding vectors. Focusing on 10% training ratio, we observe that the performance gain almost stabilizes for embedding sizes greater than 96.

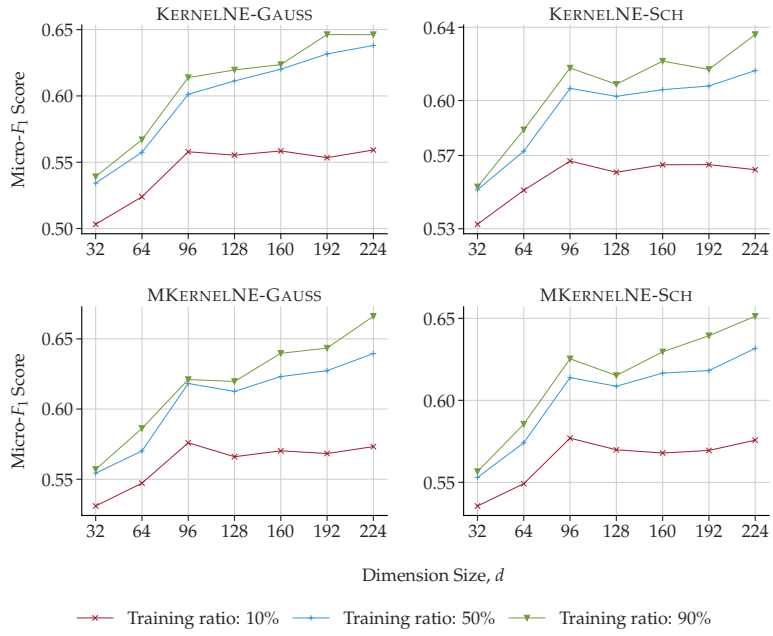


Figure 5.2: Influence of the dimension size (d) on the *Citeseer* network.

THE EFFECT OF KERNEL PARAMETERS. We have further studied the behavior of the kernel parameter σ of the Gaussian and Schoenberg kernel respectively (as described in Section 5.4.1). Figure 5.3 show how the Micro F_1 node classification score is affected with respect to σ for various training ratios on the *Citeseer* network. For the Gaussian kernel, we observe that the performance is almost stable for σ values varying from 0.25 up to 2.0, showing a sudden decrease after 4.0. For the case of the Schoenberg kernel, we observe an almost opposite behavior. Recall that, parameter σ has different impact on these kernels (Section 5.4.1). In particular, the Micro-F_1 score increases for σ values ranging from 0.25 to 0.50, while the performance almost stabilizes in the range of 4.0 to 8.0.

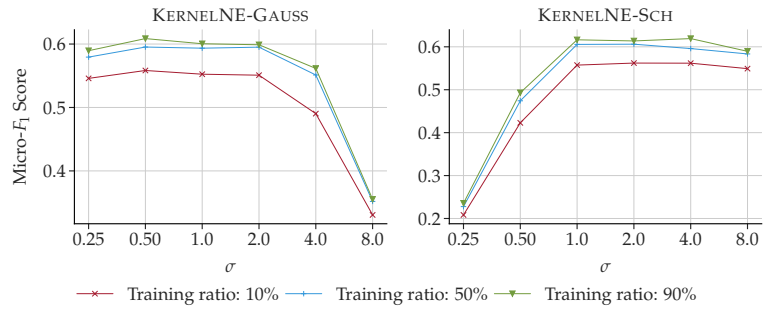


Figure 5.3: Influence of kernel parameters on the *Citeseer* network.

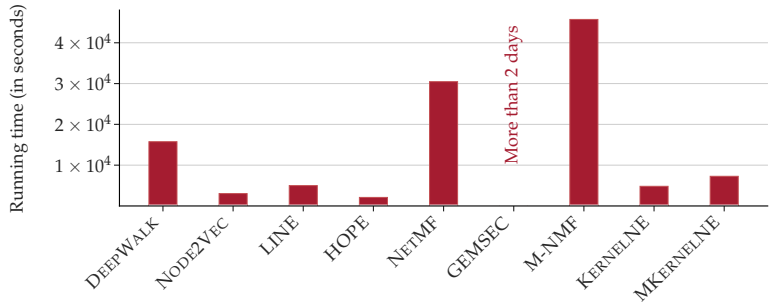


Figure 5.4: Comparison of running times of the KERNELNE and MKERNELNE models with the baselines.

5.5.5 *Running Time Comparison*

For the running time comparison of the different models, we have generated an *Erdös-Rényi* $\mathcal{G}_{n,p}$ graph model [ER60], by choosing the number of nodes $n = 10^5$ and the edge probability $p = 10^{-4}$. Figure 5.4 depicts the running time (in seconds) for both the baseline and the proposed models.

For this particular experiment, we have run all the models on a machine of 1TB memory with a single thread when it is possible. We do not report the running time of GEMSEC, since the experiment did not manage to complete within 2 days. As we can observe, both the proposed models have comparable running time—utilizing multiple kernels with MKERNELNE improves performance on downstream tasks without heavily affecting efficiency. Besides, KERNELNE runs faster than LINE, while MKERNELNE with three kernels shows also a comparable performance. It is also important to note

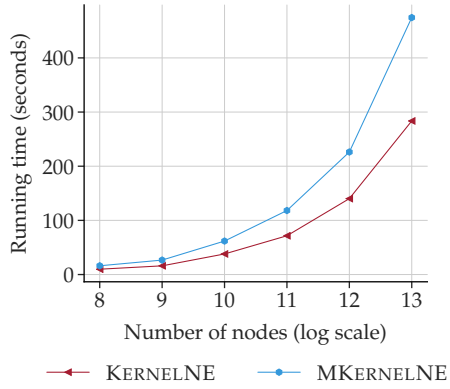


Figure 5.5: Running time analysis of the KernelNE and MKernelNE models on Erdös-Renyi graphs of varying sizes.

that although NetMF is a well-performing baseline model (especially in the task of node classification) which also relies on matrix factorization of a random walk proximity matrix, the model is not very efficient because of the way that the factorization is performed (please see Section. 5.2 for more details). On the contrary, our kernelized matrix factorization models are more efficient and scalable by leveraging negative sampling to improve scalability.

In addition to the running time comparison of the proposed approach against the different baseline models, we further examine the running time of KernelNE and MKernelNE (using three base kernels) over Erdös-Rényi graphs of varying sizes, ranging from 2^8 to 2^{13} nodes. In the generation of random graphs, we set the edge probabilities so that the expected node degree of graphs is 10. Figure 5.5 shows the running time of the proposed models. Since the Gaussian and Schoenberg kernels have similar performance, we report the running time for the Gaussian kernel only. As we observe, considering multiple kernels does not significantly affect the scalability properties of the MKernelNE model.

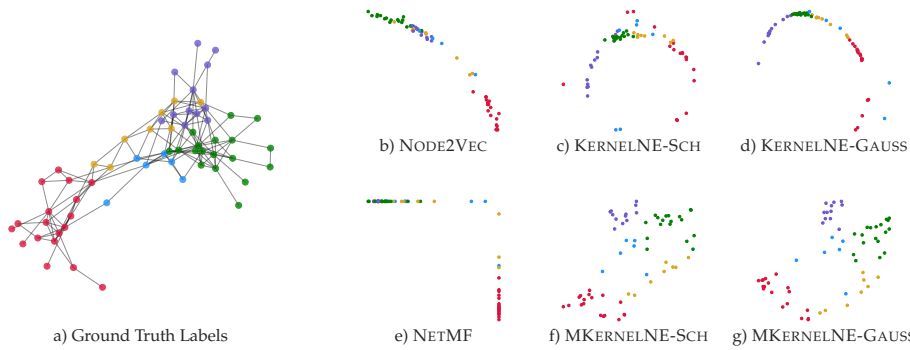


Figure 5.6: Visualization of node embeddings of *Dolphins* in the 2D space. The node colors indicate community labels, computed using the LOUVAIN algorithm.

5.5.6 *Visualization and Clustering of Embedding Vectors*

As we have discussed in Chapter 3, *Modularity* is a measure designed to assess the quality of the clustering structure of a graph [New10]. High modularity implies good clustering structure—the network consists of substructures in which nodes densely connected with each other to each other. Here, we perform a simple visualization and clustering experiment to examine the ability of the different node embedding models to capture the underlying community structure, preserving network's modularity in the embedding space. Note that, to keep the settings of the experiment simple, we leverage the raw embedding vectors visualizing them in the two-dimensional space (instead of performing visualization with t-SNE [MH08] or similar algorithms).

We perform experiments on the *Dolphins* toy network, which contains 62 nodes and 159 edges. We use the LOUVAIN algorithm [Blo+08] to detect the communities in the network. Each of these detected five communities is represented with a different color in Figure 5.6a. We also use the proposed and the baseline models to learn node embeddings in two-dimensional space.

As we can observe in Figure 5.6, different instances of MKERNELNE learn embedding vectors in which nodes of the same community are better distributed in the two-dimensional space. Besides, to further support this observation, we run the *k*-MEANS clustering algorithm [Ped+11] on the embedding

vectors, computing the corresponding Normalized Mutual Information (NMI) scores [MV13], assuming as ground-truth communities the ones computed by the LOUVAIN algorithm in the graph space. While the NMI scores for NODE2VEC and NETMF are 0.532 and 0.572 respectively, KERNELNE-SCH achieves 0.511 while for KERNELNE-GAUSS we have 0.607. The NMI scores significantly increase in the case of the multiple kernel models MKERNELNE-SCH and MKERNELNE-GAUSS, which are 0.684 and 0.740 respectively.

5.6 CONCLUSION AND FUTURE WORK

In this chapter, we have studied the problem of learning node embeddings with kernel functions. We have first introduced KERNELNE, a model that aims at interpreting random-walk based node proximity under a weighted matrix factorization framework, allowing to utilize kernel functions. To further boost performance, we have introduced MKERNELNE, extending the proposed methodology to the multiple kernel learning framework. Besides, we have discussed how parameters of both models can be optimized via negative sampling in an efficient manner. Extensive experimental evaluation showed that the proposed kernelized models substantially outperform baseline GRL methods in node classification and link prediction tasks.

The proposed kernelized matrix factorization opens up further research directions in network representation learning that we aim to explore in future work. To incorporate the sparsity property prominent in real-world networks, a probabilistic interpretation [MS08] of the proposed matrix factorization mode would be suitable. Besides, it would be interesting to examine how the proposed models could be extended in the case of dynamic networks [Kaz+20].

5.7 AN OVERVIEW OF THE PROPOSED LEARNING-BASED MODELS

We have, so far, introduced three different graph representation learning approaches addressing the same problem from different perspectives. We have devised our models based on the generation of random walks in order

to have a flexible architecture to learn node embeddings. In the next chapter, we will present the last contribution of the dissertation, which focuses on a different objective concerning the design of scalable algorithms. Therefore, here we will briefly review and compare the methods proposed so far in this thesis.

The first approach, TNE, has explicitly leveraged the extracted community information in learning node representations. Later on, we have extended the traditional SkipGram model by modeling node interactions with exponential family distributions. Finally, we have proposed a random walk-based kernelized matrix factorization method to learn more expressive embeddings.

We have mainly evaluated the performance of these approaches in two downstream tasks: node classification and link prediction. For each network, we report the score of the baseline method performing the best—so, the chosen baseline model might differ depending on the dataset. The results are depicted in Tables 5.6 to 5.9. In all tables, we use boldface to denote the best performing model, while underline indicates the best score among the variations of each method.

		TNE				EFGE			KernelNE		MKernelNE	
	Baseline	*GLDA*	*GHMM*	*Louvain*	*BigClam*	*Bern*	*Pois*	*Norm*	*Gauss*	*Sch*	*Gauss*	*Sch*
Citeseer	0.543	0.535	0.526	0.551	0.546	0.565	**0.571**	0.564	0.555	0.561	0.566	0.570
Cora	0.773	0.751	0.762	0.765	0.754	**0.787**	0.784	0.786	0.780	0.765	0.781	0.772
DBLP	0.623	0.617	0.619	0.625	0.625	0.639	**0.640**	**0.640**	0.617	0.627	0.636	0.631
PPI	0.164	0.173	0.172	0.178	0.180	0.179	0.184	0.185	0.148	**0.195**	**0.195**	0.194

Table 5.6: Comparison of methods for node classification with 10% training ratio.

		TNE				EFGE			KernelNE		MKernelNE	
	Baseline	*GLDA*	*GHMM*	*Louvain*	*BigClam*	*Bern*	*Pois*	*Norm*	*Gauss*	*Sch*	*Gauss*	*Sch*
Citeseer	0.595	0.610	0.583	0.619	0.612	**0.621**	**0.621**	0.617	0.611	0.602	0.613	0.609
Cora	0.834	**0.840**	0.824	0.835	0.828	0.835	0.822	0.829	0.837	0.800	0.823	0.806
DBLP	0.639	0.634	0.635	0.642	0.641	0.655	0.655	**0.656**	0.633	0.637	0.648	0.641
PPI	0.226	0.233	0.227	0.226	0.229	0.227	0.231	0.234	0.192	**0.244**	0.242	0.241

Table 5.7: Comparison of methods for node classification with 50% training ratio.

		TNE				EFGE			KernelNE		MKernelNE	
	Baseline	GLDA	GHMM	Louvain	BigClam	Bern	Pois	Norm	Gauss	Sch	Gauss	Sch
Citeseer	0.603	0.618	0.594	**0.638**	0.624	0.626	0.627	0.625	0.620	0.609	0.620	0.615
Cora	0.845	**0.856**	0.834	0.850	0.851	0.847	0.834	0.837	0.851	0.812	0.833	0.818
DBLP	0.641	0.638	0.634	0.642	0.642	0.656	**0.658**	0.658	0.636	0.640	0.650	0.643
PPI	0.242	0.251	0.238	0.240	0.244	0.240	0.242	0.247	0.203	**0.256**	0.254	0.251

Table 5.8: Comparison of methods for node classification with 90% training ratio.

		TNE				EFGE			KernelNE		MKernelNE	
	Baseline	GLDA	GHMM	Louvain	BigClam	Bern	Pois	Norm	Gauss	Sch	Gauss	Sch
Citeseer	0.828	0.809	0.793	0.809	0.795	0.820	0.829	0.783	0.807	**0.882**	0.850	0.863
Cora	0.781	0.775	0.806	0.780	0.767	0.753	0.777	0.782	0.774	**0.823**	0.802	0.810
DBLP	0.944	0.958	0.959	0.959	0.958	0.954	0.957	0.956	0.950	0.960	**0.961**	0.960
PPI	**0.886**	0.772	0.872	0.780	0.844	0.728	0.739	0.806	0.846	0.801	0.768	0.788
AstroPh	0.969	0.977	**0.979**	0.977	0.977	0.977	0.973	0.966	0.963	0.97	0.979	0.970
HepTh	0.896	0.903	0.908	0.905	0.904	0.899	0.902	0.907	0.899	**0.917**	**0.917**	0.916
Facebook	0.984	**0.993**	**0.993**	**0.993**	**0.993**	0.991	0.991	0.990	0.99	0.988	0.989	0.989
Gnutella	**0.839**	0.728	0.796	0.721	0.723	0.622	0.624	0.668	0.816	0.673	0.653	0.671

Table 5.9: Comparison of learning-based models in the link prediction task.

To have a comprehensive overview of the methods in terms of performance, for the node classification task we report the Micro-F_1 scores for training ratios of 10%, 50%, and 90%. As we can observe in Tables 5.6 to 5.8, the instances of EFGE become prominent for various networks—modeling the node interactions with a proper distribution, produces more expressive embeddings. This model performs especially well for 10% training size. On the other hand, TNE and KerneNE show comparable results for larger training sets.

In the link prediction experiments (Table 5.9), the KernelNE-Sch and MKernelNE-Gauss variants show good performance—being the best performing models over the four networks. KernelNE-Gauss is also the method having the highest score over the *PPI* and *Gnutella* networks after the best-performing baseline. We observe that the integration of community infor-

mation positively contributes to the performance of the TNE instances, that possess comparable scores.

6

RANDOM WALK DIFFUSION MEETS HASHING FOR SCALABLE GRAPH EMBEDDINGS

Most widely used models face computational challenges to scale to large networks, as the size of the networks increases. While there is a recent effort towards designing algorithms that solely deal with scalability issues, most of them behave poorly in terms of accuracy on downstream tasks. In this chapter, we aim at studying models that balance the trade-off between efficiency and accuracy. In particular, we propose NodeSig, a scalable embedding model that computes binary node representations. NodeSig exploits random walk diffusion probabilities via stable random projection hashing towards efficiently computing embeddings in the Hamming space. Our extensive experimental evaluation on various graphs has demonstrated that the proposed model achieves a good balance between accuracy and efficiency compared to well-known baseline models on two downstream tasks.

6.1 INTRODUCTION

As we have discussed in Chapters 2 to 5, the majority of the existing node embedding approaches propose *learning-based* techniques relying either on matrix factorization or on node context sampling to infer nodes proximities [HYL17b]. However, the computational cost and the high memory usage burden bring limitations to their applicability on large-scale networks. Recent graph representation studies aim to improve running time complexity via matrix sparsification techniques [Qiu+19] or capitalizing on hierarchical graph representations [Blo+08] but the quality of the embeddings deteriorates significantly.

Besides the computational burden of model optimization, most of the proposed algorithms learn low-dimensional embeddings in the *Euclidean* space.

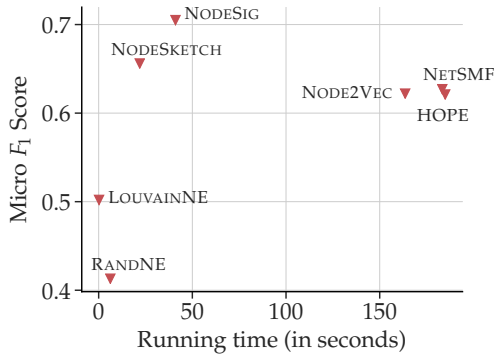


Figure 6.1: Comparison of models on the *DBLP* network. NodeSig balances between good accuracy and low running time.

A few recent studies have suggested learning *discrete* node representations [Lia+18; Wu+18; Yan+19], in which *Hamming* distance is leveraged to compute the pairwise similarity of embedding vectors very fast. The basic idea builds upon fast sketching techniques for scalable similarity search, mainly based on data-independent or data-dependent hashing techniques [Wan+18]. Although binary embeddings speedup distance measure computations, the corresponding models often undergo computationally intensive learning procedures, especially in the case of learning-to-hash models [Lia+18].

In this chapter, we propose NodeSig, a scalable model for computing expressive binary node embeddings based on stable random projections. NodeSig first leverages random walk diffusions to estimate higher-order node proximity. Then, a properly defined sign random projection hashing technique is applied to obtain binary node signatures in the Hamming space, leading to an approximation of the *chi similarity* (χ) [PW10] between the proximity vectors in the original space. Since these vectors are constructed based on the occurrence frequencies of nodes within random walks, *chi* similarity emerges as a natural choice of similarity metric, frequently used to compare histograms in various areas including natural language processing and computer vision [Ye+15; Huo+09].

Each component of NodeSig has been designed to ensure the scalability of the model, while at the same time the accuracy on downstream tasks is

not compromised or even improves compared to traditional models. Figure 6.1 positions NODESIG regarding accuracy and running time, providing a comparison to different models on the *DBLP* network. As we observe, NODESIG's running time is comparable to that of models that focus solely on scalability (e.g., NODESKETCH, RANDNE, LOUVAINNE), with improved accuracy even higher than NODE2VEC or HOPE in this particular dataset. The main contributions of the chapter can be summarized as follows:

- We introduce NODESIG, a scalable and expressive model for binary node embeddings based on stable random projection hashing of random walk diffusion probabilities. NODESIG leverages higher-order proximity information among nodes, while its design allows scaling to large graphs without sacrificing accuracy.
- The distance computation between node signatures in the embedding space is provided by the Hamming distance on bit vectors, which is significantly more efficient than distance computations based on other distance measures. Besides, NODESIG can easily be extended to deal with dynamic networks since the computation of node embeddings are entirely independent of each other. In other words, the embedding of a particular node relies on local information without requiring knowledge about other nodes' representations.
- In a thorough experimental evaluation, we demonstrate that the proposed binary embeddings achieve superior performance compared to various baseline models on two downstream tasks. At the same time, the running time required to compute node signatures allows the model to scale on large graphs.

The rest of the chapter is organized as follows. Section 6.2 presents related work in the field. Section 6.3 presents the proposed approach in detail. Performance evaluation results are offered in Section 6.4, whereas Section 6.5 discusses the potential of the approach for dynamic networks. Finally, we summarize the work in Section 6.6.

SOURCE CODE. The implementation of the proposed model in C++ can be found at the address: https://abdcelikkanat.github.io/projects/nodesig/.

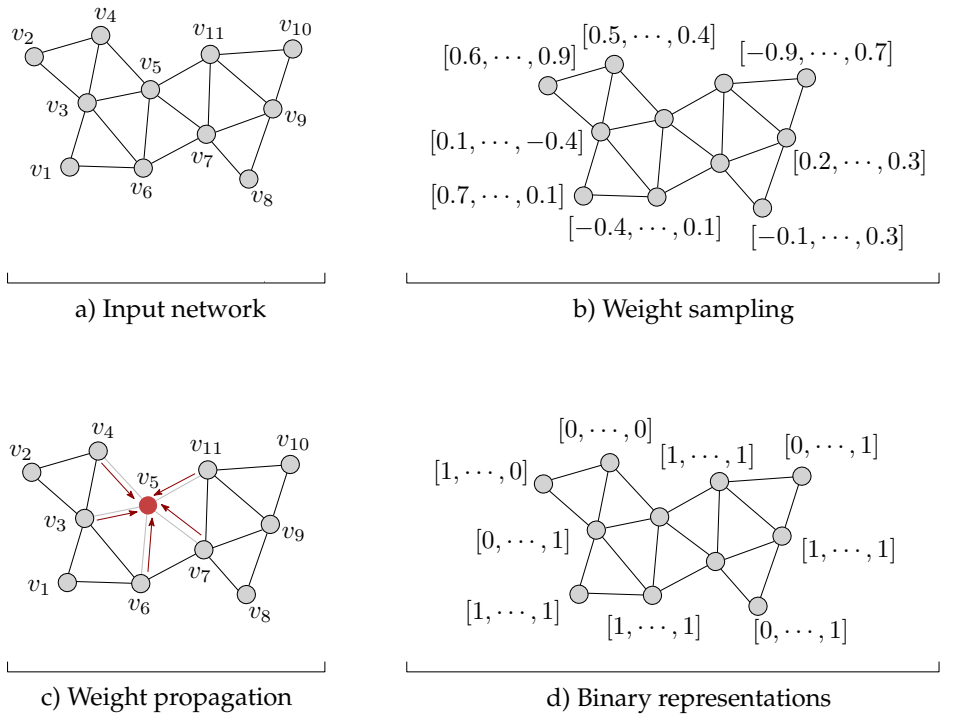
6.2 RELATED WORK

The main limitation of the conventional graph representation learning techniques is that they do not scale well for large networks. The main focus has been put on increasing the effectiveness of data mining tasks (e.g., classification, link prediction, network reconstruction) whereas the efficiency dimension has not received significant attention. To attack this problem, recent advances in GRL use random projection or hashing techniques (more specifically, variants of locality-sensitive hashing) in order to boost performance, trying to maintain effectiveness as well.

One of the first scalable approaches (RandNE) was proposed in [Zha+18]. RandNE is based on iterative Gaussian random projection, being able to adapt to any desired proximity level. In the same line, FastRP was proposed in [Che+19] which is faster than RandNE and also more accurate. LouvainNE [Bho+20] suggested learning node representations by aggregating the embeddings of nodes extracted at varying levels of the hierarchy.

Recently, embedding techniques based on hashing have emerged as a promising alternative to enable faster processing while at the same time retain good accuracy results. The NetHash algorithm, proposed in [Wu+18], expands each node of the graph into a rooted tree, and then by using a bottom-up approach encodes structural information as well as attribute values into minhash signatures in a recursive manner. A similar approach has been used in NodeSketch [Yan+19], where the context of every node is defined in a different way whereas the embedding vector of each node contains integer values and the weighted Jaccard similarity coefficient is being used. Based on performance evaluation results reported in [Yan+19], NodeSketch is extremely efficient managing to reduce the embedding cost by orders of magnitude in comparison to baseline approaches. Moreover, NodeSketch achieves comparable (or even better) F1 score in the node classification task as well as comparable or better precision in the link prediction task with respect to the baseline techniques.

Our proposed model, NodeSig, takes a different approach. First, it samples weights from the Cauchy distribution in order to construct the projection matrix. Then, it obtains the projected values by performing recursive update



a) Input network b) Weight sampling

c) Weight propagation d) Binary representations

Figure 6.2: Schematic representation of the NODESIG model. Firstly, the weights of the random projection matrix are sampled and then the projection of the proximity matrix is performed via the weight propagation step. Finally, binary node representations are obtained by combining the signs of the projected values.

rules instead of explicitly realizing the data matrix. Finally, a bit-vector is generated, corresponding to the signature of the projected data. As we will show shortly, NODESIG shows comparable or even better accuracy results compared to NODESKETCH, while at the same time, it can easily be adapted to more demanding settings like dynamic or streaming environments.

6.3 PROPOSED APPROACH

In this section, we introduce the proposed approach, referred to as NODESIG, which aims at representing the nodes of the network as fixed-length binary codes. The model mainly relies on sign stable random projections of a properly designed matrix that captures higher order proximity among nodes. Initially, we describe how we construct this target matrix by using random

walk diffusion probabilities, and then we demonstrate how the random projections can efficiently be performed by propagating the sampled weights without the need for the explicit realization of the target matrix. Finally, we obtain the binary representations by incorporating a nonlinear mapping through a simple sign function. An overview of the basic steps of NodeSig is given in Figure 1.1.

6.3.1 *Random Walk Diffusion for Node Proximity Estimation*

In most cases, direct links among nodes are not sufficient to grasp various inherent properties of the network that are related to node proximity. It is highly probable that the network might have missing or noisy connections, thus relying solely on first-order proximity can reduce the expressiveness of the model. Random walk diffusions constitute an interesting way to leverage higher-order information while computing embeddings. The underlying idea relies on the co-occurrence frequencies of nodes up to a certain distance in the random walks; nodes appearing more frequently close to each other within the random walks share similar characteristics, and therefore should be placed close to each other in the embedding space. This idea has been exploited by various representation learning models [PARS14; GL16] and our works [ÇM20; ÇM19a; ÇM21] described in the previous chapters. Nevertheless, sampling multiple random walks, as used in various models significantly increases the training time causing scalability issues.

To overcome this problem, in this chapter we directly leverage random walk diffusions, adopting a uniform random walking strategy to extract information describing the structural roles of nodes in the network. Let $\mathbf{P}$ used to denote the right stochastic matrix associated with the adjacency matrix $\mathbf{W}$ of the graph, which is obtained by normalizing the rows of the matrix. More formally, $\mathbf{P}$ can be written as $\mathbf{P}_{(v,u)} := \mathbf{W}_{(v,u)} / \sum_{x \in \mathcal{V}} \mathbf{W}_{(v,x)}$, defining the transition probabilities of the uniform random walk strategy. We use a slightly modified version of the transition matrix by adding a self-loop on each node, in case it does not exist.

Note that the probability of visiting the next node depends only on the current node that the random walk resides; therefore, node u can be visited

starting from v by taking l steps with probability $\mathbf{P}^{(l)}_{(v,u)}$, if there is a path connecting them. For a given walk length $\mathcal{L}$, we define the matrix $\mathbf{M}$ as

$$\mathbf{M} := \mathbf{P} + \cdots + \mathbf{P}^{(l)} + \cdots + \mathbf{P}^{(\mathcal{L})},$$

where $\mathbf{P}^{(l)}$ indicates the l-order proximity matrix and each entry $\mathbf{M}_{(v,u)}$ in fact specifies the expectation of visiting u starting from node v within $\mathcal{L}$ steps. By introducing an additional parameter α, $\mathbf{M}(\alpha)$ can be rewritten by:

$$\mathbf{M}(\alpha) := \alpha\mathbf{P} + \cdots + \alpha^{(l)}\mathbf{P}^{(l)} + \cdots + \alpha^{(\mathcal{L})}\mathbf{P}^{(\mathcal{L})}.$$

Higher order node proximities can be captured using longer walk lengths, where the impact of the walk at different steps is controlled by the *importance factor* α. As we will present in the next paragraph, matrix $\mathbf{M}(\alpha)$ is properly exploited by a random projection hashing strategy to efficiently compute binary node representations.

6.3.2 *Learning Binary Embeddings*

Random projection methods [Vem01] have been widely used in a wide range of machine learning applications that are dealing with large scale data. They mainly target to represent data points into a lower dimensional space by preserving the similarity in the original space. Here, we aim at encoding each node of the network in a *Hamming* space $\mathbb{H}\big(d_{\mathcal{H}}, \{0,1\}^d\big)$; we consider the normalized Hamming distance $d_{\mathcal{H}}$ as the distance metric [YCP15]. The benefit of binary representations is twofold: first, they will allow us to perform efficient distance computation using bitwise operations, and secondly reduce the required disk space to store the data.

Random projections are linear mappings; the binary embeddings though require nonlinear functions to perform the discretization step, and a natural choice is to consider the signs of the values obtained by the Johnson-Lindenstrauss (JL) [JLS86] transform. More formally, it can be written that

$$h_{\mathbf{W}}(\mathbf{x}) := \text{sign}(\mathbf{x}^\top \mathbf{W}),$$

where $\mathbf{W}$ is the projection matrix whose entries $\mathbf{W}_{(i,j)}$ are independently drawn from normal distribution and $\text{sign}(\mathbf{x})_j$ is equal to 1 if $\mathbf{x}_j > 0$ and 0 otherwise. The approach was first introduced by Goemans and Williamson [GW95] for a rounding scheme in approximation algorithms, demonstrating that the probability of obtaining different values for a single bit quantization is proportional to the angle between vectors, as it is shown in Theorem 6.1. The main idea relies on sampling uniformly distributed random hyperplanes in $\mathbb{R}^d$. Each column of the projection matrix, in fact, defines a hyperplane and the arc between vectors $\mathbf{x}$ and $\mathbf{y}$ on the unit sphere is intersected if $h_{\mathbf{W}}(\mathbf{x})_j$ and $h_{\mathbf{W}}(\mathbf{y})_j$ take different values.

Theorem 6.1 ([GW95]). *For a given pair of vectors* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$,

$$\mathbb{P}\left[h_{\mathbf{W}}(\mathbf{x})_j \neq h_{\mathbf{W}}(\mathbf{y})_j\right] = \frac{1}{\pi}\cos^{-1}\left(\frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2}\right),$$

where $\mathbf{W}_{(i,j)} \sim \mathcal{N}(0,1)$ *for* $1 \leq i, j \leq N$.

Although the signs of JL random projections allow us to approximate the angle between the vectors in the original space, in our settings, we would prefer to preserve a distance metric that can fit better the input data $\mathbf{M}(\alpha)$. Note that, the node proximity matrix $\mathbf{M}(\alpha)$ contains non-negative elements that are computed based on the occurrence frequencies of nodes within random walks. Hence, we will focus on estimating distance metrics capable of comparing histogram-type data by properly redesigning the projection matrix. The *stable* random projections approach [LSH13] generalizes the aforementioned idea by using a symmetric α-stable distribution with unit scale in order to sample the elements of the projection matrix, for $0 < \alpha \leq 2$. Li et al. [LSH13] proposed the following upper bound

$$\mathbb{P}\left[h_{\mathbf{W}}(\mathbf{x})_j \neq h_{\mathbf{W}}(\mathbf{y})_j\right] \leq \frac{1}{\pi}\cos^{-1}\rho_\alpha \tag{6.1}$$

for non-negative vectors ($\mathbf{x}_i \geq 0, \mathbf{y}_i \geq 0$, $1 \leq i \leq N$), where ρ_α is defined by

$$\rho_\alpha := \left(\frac{\sum_{m=1} x_m^{\alpha/2} y_m^{\alpha/2}}{\sqrt{\sum_{m=1} x_i^\alpha}\sqrt{\sum_{m=1} y_m^\alpha}}\right)^{2/\alpha}.$$

It is well known that the bound is exact for $\alpha = 2$, which also corresponds to the special case in which normal random projections are performed. When the vectors are chosen from the $\ell^1(\mathbb{R}^+)$ space (i.e., $\sum_{i=1} x_i = 1$, $\sum_{i=1} y_i = 1$), it is easy to see that the χ^2 similarity ρ_{χ^2} defined as $\sum_{i=1}(2x_iy_i)/(x_i + y_i)$ is always greater or equal to ρ_1, as suggested by Lemma 6.1.

Lemma 6.1. *For given $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ satisfying $x_i, y_i \geq 0$ for all $1 \leq i \leq d$ and $\sum_{i=1} x_i = \sum_{i=1} y_i = 1$, then*

$$\rho_1 \leq \rho_{\chi^2}.$$

Proof.

$$\begin{aligned}\rho_1 = \left(\frac{\sum_{i=1} x_i^{1/2} y_i^{1/2}}{\sqrt{\sum_{i=1} x_i}\sqrt{\sum_{i=1} y_i}}\right)^2 = \left(\sum_{i=1} \sqrt{x_i y_i}\right)^2 &= \left(\sum_{i=1} \frac{\sqrt{2x_iy_i}}{\sqrt{x_i + y_i}} \frac{\sqrt{x_i + y_i}}{\sqrt{2}}\right)^2 \\ &\leq \sum_{i=1} \frac{2x_iy_i}{x_i + y_i} \sum_{i=1} \frac{x_i + y_i}{2} \\ &= \sum_{i=1} \frac{2x_iy_i}{x_i + y_i} = \rho_{\chi^2},\end{aligned}$$

where the inequality follows from the *Cauchy-Schwarz* inequality. □

Besides, it has been empirically shown [LSH13] that the collision probability for *Cauchy* random projections with unit scale can be well estimated, especially for sparse data:

$$\mathbb{P}\left[h_{\mathbf{W}}(\mathbf{x})_j \neq h_{\mathbf{W}}(\mathbf{y})_j\right] \approx \frac{1}{\pi}\cos^{-1}\rho_{\chi^2} \leq \frac{1}{\pi}\cos^{-1}\rho_1. \tag{6.2}$$

Note that, the matrix $\mathbf{M}(\alpha)$ described in the previous paragraph consists of non-negative values; its row sums are equal to $\sum_{l=1}^{\mathcal{L}} \alpha^{(l)}$ and $\mathbf{M}(\alpha)$ is sparse enough for small walk lengths. Therefore, we design the projection matrix by sampling its entries from the *Cauchy* distribution, aiming to learn binary representations preserving the *chi-square* similarity. The *chi-square* distance is one of the measures used for histogram-based data, commonly used in the fields of computer vision and natural language processing [Ye+15; Huo+09].

As it is shown in Figure 6.2, the last step of NodeSig for obtaining binary node representations is to utilize the signs of the projected data. In other words, the embedding vector $\mathbf{E}[v]$ for each node $v \in V$ is computed as follows:

$$\mathbf{E}[v] := \left[\text{sign}\left(\mathbf{M}_{(v,:)}(\alpha)\ \mathbf{W}_{(:,1)}\right), \ldots, \text{sign}\left(\mathbf{M}_{(v,:)}(\alpha)\ \mathbf{W}_{(:,d)}\right)\right].$$

Note that, the projection of the exact realization of $\mathbf{M}(\alpha)$ can be computationally intensive, especially for large walks. Instead, it can be computed by propagating the weights $\mathbf{W}_{(u,j)}$ for each dimension j ($1 \leq j \leq d$), using the following recursive update rule:

$$\mathcal{R}^{(l+1)}_{(v,j)}(\alpha) \leftarrow \alpha \sum_{u \in \mathcal{N}(v)} \mathbf{P}_{(v,u)} \times \left(\mathbf{W}_{(u,j)} + \mathcal{R}^{(l)}_{(u,j)}(\alpha)\right), \tag{6.3}$$

where $\mathcal{N}(v)$ refers to the set of neighbors of node $v \in V$ and $\mathcal{R}^{(l)}_{(v,j)}$ is equal to the projected data, $\mathbf{M}_{(v,:)}(\alpha) \cdot \mathbf{W}_{(:,j)}$ for the walk length l, and $\mathcal{R}^{(0)}_{(v,j)}$ is initialized to zero. By Lemma 6.2, it can be seen that the projection of $\mathbf{M}(\alpha)$ can be computed with the recursive update rules defined in Equation 6.3.

Lemma 6.2. *Let $\mathbf{P}$ be $N \times N$ a right stochastic matrix and $\mathbf{M}^{(\mathcal{L})}(\alpha)$ be the matrix defined by $\alpha\mathbf{P} + \cdots + \alpha^{(l)}\mathbf{P}^{(l)} + \cdots + \alpha^{(\mathcal{L})}\mathbf{P}^{(\mathcal{L})}$. For a given $\mathbf{W} \in \mathbb{R}^{N \times \mathcal{D}}$, we have*

$$\mathbf{M}^{(\mathcal{L})}(\alpha)\mathbf{W} := \left(\alpha\mathbf{P} + \cdots + \alpha^{(l)}\mathbf{P}^{(l)} + \cdots + \alpha^{(\mathcal{L})}\mathbf{P}^{(\mathcal{L})}\right)\mathbf{W} = \mathcal{R}^{(\mathcal{L})}, \tag{6.4}$$

where each $R^{(l)}_{(i,j)}$ is recursively defined by $\alpha\mathbf{P}_{(i,:)}\left(\mathbf{W}_{(:,j)} + \mathcal{R}^{(l-1)}_{(:,j)}\right)$ for all $l \in \{1, \ldots, \mathcal{L}\}$, $i \in \{1, \ldots, N\}$ and $\mathcal{R}^{(0)}$ is set to 0.

Proof. For $l = 1$, we have that

$$\mathcal{R}^{(1)}_{(i,j)} = \alpha\mathbf{P}_{(i,:)}\left(\mathbf{W}_{(:,j)} + \mathcal{R}^{(0)}_{(:,j)}\right) = \alpha\mathbf{P}_{(i,:)}\left(\mathbf{W}_{(:,j)} + 0\right) = \alpha\mathbf{P}_{(i,:)}\mathbf{W}_{(:,j)},$$

for all $j \in \{1, \ldots, d\}$, and $i \in \{1, \ldots, N\}$ so the claim in Equation (6.4) holds. Let us assume that it is true for $n = l \geq 1$. Then,

$$\mathcal{R}^{(n+1)}_{(i,j)} = \alpha\mathbf{P}_{(i,:)}\left(\mathbf{W}_{(:,j)} + \mathcal{R}^{(n)}_{(:,j)}\right)$$

$$
\begin{aligned}
&= \alpha \mathbf{P}_{(i,:)} \left(\mathbf{W}_{(:,j)} + \alpha \mathbf{P} \mathbf{W}_{(:,j)} + \cdots + \alpha^{(n)} \mathbf{P}^{(n)} \mathbf{W}_{(:,j)} \right) \\
&= \alpha \mathbf{P}_{(i,:)} \mathbf{W}_{(:,j)} + \cdots + \alpha^{(i+1)} \mathbf{P}^{(n+1)}_{(i,:)} \mathbf{W}_{(:,j)} \\
&= \left(\alpha \mathbf{P}_{(i,:)} + \cdots + \alpha^{(i+1)} \mathbf{P}^{(n+1)}_{(i,:)} \right) \mathbf{W}_{(:,j)} \\
&= \mathbf{M}^{(n+1)}(\alpha) \mathbf{W}_{(:,j)}.
\end{aligned}
$$

Thus, the claim holds for $n+1=l$. By the principle of induction, it satisfies for all $l \in \{1, \ldots, \mathcal{L}\}$. □

Algorithm 6.1 provides the pseudocode of NodeSig. Firstly, we generate the projection matrix by sampling the weights from the *Cauchy* distribution with unit scale. The samples are further divided by $\sum_{l=1} \alpha^{(l)}$, because the row sums of $\mathbf{M}(\alpha)$ must be equal to 1. Then, we compute the terms $\mathcal{R}^{(l)}_{(v,j)}$ by propagating the weights in Line 9 at each walk iteration $l < \mathcal{L}$. Note that, the term R in the pseudocode is a vector of length d, thus we obtain the final representation of each node using the signs of $\mathcal{R}^{(\mathcal{L})}_{(v,j)}$.

6.3.3 *Time and Space Complexity*

At the beginning of the algorithm, we need to sample a weight matrix of size $d \cdot |\mathcal{V}|$, and it can be formed in the order of $\mathcal{O}(d \cdot |\mathcal{V}|)$. As we observe in Algorithm 6.1, the main cumbersome point of NodeSig is caused by the update rule defined in Equation 6.3, which corresponds to Line 9 of the pseudocode. The update rule must be repeated $|\mathcal{N}(v)|$ times for each node $v \in \mathcal{V}$, thus it requires $2 \cdot m \cdot d$ multiplication operations at the walk step l ($1 \leq l \leq \mathcal{L}$) for a network consisting of m edges and for embedding vectors of dimension d. Hence, the overall time complexity of the algorithm is $\mathcal{O}\left((d \cdot |\mathcal{V}| + d \cdot m \cdot \mathcal{L}\right)$. During the running course of the algorithm, we need to store a vector of size $|\mathcal{V}|$ in memory for the computation of each dimension. Assuming, in the worst case, that we aim to retain the whole projection matrix $\mathbf{W}$ in memory, we need $\mathcal{O}(d \cdot |\mathcal{V}|)$ space in total, since each node requires d space for storing the $\mathcal{R}^{(l)}_{(v,j)}$ values in the update rule of Equation 6.3. Note that, the performance of the algorithm can be boosted by using parallel processing for each dimension of embedding vectors or

Algorithm 6.1 NODESIG

Input: Graph: $G = (\mathcal{V}, \mathcal{E})$ with the transition matrix $\mathbf{P}$
Representation size: d
Walk length: $\mathcal{L}$
Importance factor: α
Output: Embedding matrix: $\mathbf{E}$

/* Initialization */
1: **for each** node $v \in V$ **do**
2: $\mathcal{R}[v] \leftarrow \mathbf{0}_d = (0, \ldots, 0)$
3: $\mathbf{W}[v] \sim Cauchy(0,1)^d \;/\; \sum_{l=1}^{\mathcal{L}} \alpha^l$
4: **end for**
/* Project the matrix with recursive update rules */
5: **for** $l \leftarrow 1$ **to** $\mathcal{L}$ **do**
6: **for each** node $v \in V$ **do**
7: $temp[v] \leftarrow \mathbf{0}_d = (0, \ldots, 0)$
8: **for each** neighbour node $u \in \mathcal{N}(v)$ **do**
9: $temp[v] \leftarrow temp[v] + \mathbf{P}[v,u] \times (\mathbf{W}[u] + \mathcal{R}[u])$
10: **end for**
11: **end for**
12: **for each** node $v \in V$ **do**
13: $\mathcal{R}[v] \leftarrow \alpha \times temp[v]$
14: **end for**
15: **end for**
/* Get the sign of each value */
16: **for each** node $v \in V$ **do**
17: $\mathbf{E}[v] \leftarrow \text{sign}(\mathcal{R}[v])$
18: **end for**

for Line 6, since the required computation for each node is completely independent.

6.3.4 *Discussion for Dynamic Networks*

The majority of existing network representation learning models have been developed for static networks. Nevertheless, most real-world networks undergo structural changes and evolve over time with the addition and removal of links and nodes [Kaz+20]. Therefore, being able to design models that properly adapt to dynamic networks is an important point to investigate.

As we discuss here, the proposed method allows for efficient updates of the embeddings, without requiring any costly learning procedures.

More precisely, the key point in the dynamic case, is that the learned embedding vectors should be efficiently updated instead of being recalculated from scratch. If an edge is added or removed for a pair of nodes $(u, v) \in V \times V$, the terms $\mathcal{R}^{(l)}_{(w,:)}$ in Equation (6.3) for node $w \in V$ are affected, for all $l > k := \min\{dist(w, u), dist(w, v)\}$—thus, it suffices to update only these affected terms. The transition probabilities for nodes u and v also change even though the remaining nodes are not affected, so all the terms $\mathbf{P}_{(v,:)}$ must be divided by $\sum_{w \in \mathcal{N}(v)} P_{(v,w)}$ in order to normalize the transition probabilities and similarly the same procedure must also be applied to node u after each edge insertion and deletion operation.

Evidently, more research is required towards this direction in order to provide a method that scales well when the input data changes rapidly (i.e., in the form of a stream).

6.4 EXPERIMENTAL EVALUATION

In this section, we report empirical evaluation results demonstrating the effectiveness and efficiency of NODESIG compared to baseline algorithms. All the experiments have been performed on an Intel Xeon 2.4GHz CPU server (32 Cores) with 60GB of memory.

6.4.1 *Datasets and Baseline Models*

DATASETS. We perform experiments on networks of different scales and types. In addition to the networks that we have described in Chapter 2, we also use *Blogcatalog* [TL09a], which is a social network constructed by using the relationships among bloggers, where node labels indicate the blog categories specified by the blogger. The network contains 10,312 nodes, 333,893 edges and 39 labels. *Youtube* [TL09b] which has been crawled from the corresponding video sharing platform. Node labels indicate categories of

videos among 47 possible options and it exactly consists of $1,138,499$ nodes and $2,990,443$ edges.

BASELINE MODELS. We have considered six representative baseline methods in the evaluation. In particular, two of these methods (NODE2VEC, LINE) correspond to widely used node embedding models that we have also employed in previous chapters. The remaining ones constitute more recent models aiming to address the scalability challenge, which we provide brief information about them below. For all methods, we learn embedding vectors of size 128.

- NETSMF [Qiu+19] is a sparse matrix factorization method, recently proposed to deal with the scalability constraints of NETMF [Qiu+18]—a model that relies on the pointwise mutual information of node co-occurrences. In our experiments, we set the rank parameter to 512 and the number of rounds to $10,000$ for all networks except *PPI* and *Youtube* in which the model was unable to run. In these cases, the rank parameter is set to 256, while the number of rounds to $1,000$ and 50 for *PPI* and *Youtube*, respectively.

- RANDNE [Zha+18] is an efficient method based on Gaussian random projections, and one of the most widely applied scalable models. The experiments were conducted by setting the parameters suggested by the authors. In the case of node classification, the transition matrix with parameter values $q = 3$ and $weights = [1, 10^2, 10^4, 10^5]$ was used, while the adjacency matrix with $q = 2$ and $weights = [1, 1, 10^{-2}]$ was considered in the link prediction experiments.

- LOUVAINNE [Bho+20] constructs a hierarchical subgraph structure and aggregates the node representations learned at each different level to obtain the final embeddings. In our experiments, we have used the recommended parameter settings and α was set to 0.01.

- NODESKETCH [Yan+19] learns embeddings in the *Hamming* space, using MINHASH signatures. For the *Cora* network, we have performed parameter tuning for the values of α and order k, which are set to 0.1

and 20, respectively. For the remaining datasets, we have used the best-performing settings recommended by the authors.

For simplicity, we set the importance factor α to 1 in all the experiments of NodeSig, as we have observed that the algorithm shows comparable performance for values close to 1; a detailed analysis of the behaviour of NodeSig with respect to the importance factor is given in Section 6.4.4. The walk length is set to 3 for *Cora*, and to 5 for all the other networks in the classification experiment. For the link prediction task, the walk length is chosen as 15 for all networks. We set the dimension size of the embedding vectors to $8,192$ bits in order to be consistent with the experiments with the baseline methods, since modern computer architectures use 8 Bytes for storing floating point data types.

6.4.2 *Multi-label Node Classification*

The networks described previously consist of nodes having at least one or more labels. In the classification task, our goal is to correctly infer the labels of nodes chosen for the testing set, using the learned representations and the labels of nodes in the rest of the network, namely the nodes in the training set. The evaluation follows a strategy similar to the one used by baseline models [Yan+19].

EXPERIMENTAL SET-UP. The experiments are carried out by training an one-vs-rest *SVM* classifier with a pre-computed kernel, which is designed by computing the similarities of node embeddings. The similarity measure is chosen depending on the algorithm that we use to learn representations. More specifically, the *Hamming* similarity for NodeSketch and the *Cosine* similarity for the rest baselines methods are chosen in order to build the kernels for the classifier. For NodeSig, we use the *chi* similarity χ, defined as $1 - \sqrt{d_{\chi^2}}$, where d_{χ^2} is equal to

$$d_{\chi^2} := \sum_{i=1}^{d} \frac{(x_j - y_j)^2}{x_j + y_j} = \sum_{j=1}^{d} (x_j + y_j) - \sum_{m=1}^{d} \frac{4x_j y_j}{x_j + y_j} = 2 - 2\rho_{\chi^2},$$

	2%	4%	6%	8%	10%	30%	50%	70%	90%
HOPE	0.239	0.273	0.289	0.300	0.303	0.314	0.314	0.319	0.328
	0.085	0.100	0.107	0.113	0.117	0.116	0.117	0.120	0.125
NODE2VEC	0.285	0.322	0.332	0.335	0.342	0.348	0.354	0.355	0.350
	0.109	0.136	0.146	0.149	0.157	0.158	0.170	0.173	0.170
NETSMF	**0.326**	**0.351**	**0.366**	**0.376**	**0.380**	0.392	0.399	0.402	0.412
	0.152	**0.174**	**0.188**	**0.201**	**0.203**	0.216	0.226	0.226	0.226
LOUVAINNE	0.041	0.032	0.030	0.034	0.046	0.107	0.155	0.161	0.172
	0.022	0.020	0.020	0.019	0.023	0.031	0.039	0.040	0.041
RANDNE	0.258	0.289	0.310	0.312	0.322	0.340	0.339	0.340	0.349
	0.097	0.117	0.137	0.138	0.148	0.166	0.164	0.171	0.177
NODESKETCH	0.220	0.253	0.276	0.295	0.304	0.359	0.380	0.387	0.400
	0.074	0.101	0.115	0.130	0.144	0.211	0.237	0.248	0.259
NODESIG	0.266	0.319	0.338	0.354	0.361	**0.398**	**0.410**	**0.419**	**0.430**
	0.108	0.144	0.165	0.182	0.196	**0.249**	**0.270**	**0.281**	**0.287**

Table 6.1: Node classification for varying training sizes on *Blogcatalog*. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

for the vectors satisfying $\sum_j x_j = \sum_j y_j = 1$ and $x_j \geq 0$, $y_j \geq 0$ for all $1 \leq j \leq d$. Hence, we apply a small transformation while constructing the kernel matrix of the SVM in order to approximate the *chi* similarity, instead of using $\cos^{-1}\rho_{\chi^2}/\pi$ in Equation (6.2), which is estimated directly via the Hamming distance.

EXPERIMENTAL RESULTS. For the multi-label node classification task, Tables 6.1 to 6.5 report the average Micro-F_1 and Macro-F_1 scores over 10 runs, where the experiments are performed on different training set sizes. The symbol "-" is used to indicate that the corresponding algorithm is unable to run due to excessive memory usage or because it requires more than one day to complete. The best and second best performing models for each training ratio (10%, 50%, and 90%) are indicated with bold and underlined text, respectively.

As we observe, NODESIG consistently outperforms the baselines for higher training ratios on the *Blogcatalog* and *Cora* networks, while the obtained Macro-F_1 score is very close to the performance of NETSMF for 10% training ratio on *Blogcatalog*. In the case of the *Cora* network which corresponds to the smallest one used in our study, NODE2VEC shows better performance for small training ratio of 10%. For the *Youtube* and *DBLP* networks, the proposed

	2%	4%	6%	8%	10%	30%	50%	70%	90%
HOPE	0.474	0.560	0.632	0.654	0.687	0.763	0.780	0.787	0.803
	0.395	0.494	0.611	0.620	0.672	0.757	0.772	0.780	0.798
Node2Vec	0.641	0.707	0.745	0.754	0.757	0.798	0.805	0.813	0.833
	0.581	0.663	0.723	0.736	0.742	0.786	0.791	0.798	0.821
NetSMF	**0.670**	**0.732**	**0.753**	**0.773**	**0.778**	0.813	0.826	0.839	0.830
	0.636	**0.701**	**0.743**	**0.763**	**0.769**	0.806	0.817	0.832	0.817
LouvainNE	0.577	0.656	0.675	0.685	0.691	0.710	0.717	0.724	0.745
	0.497	0.615	0.632	0.642	0.660	0.673	0.683	0.690	0.705
RandNE	0.421	0.494	0.529	0.576	0.581	0.664	0.684	0.693	0.702
	0.312	0.443	0.486	0.552	0.558	0.659	0.683	0.690	0.694
NodeSketch	0.474	0.604	0.654	0.696	0.710	0.807	0.842	0.860	0.882
	0.340	0.549	0.621	0.670	0.690	0.795	0.832	0.852	0.873
NodeSig	0.514	0.659	0.699	0.727	0.746	**0.823**	**0.850**	**0.869**	**0.886**
	0.433	0.622	0.679	0.712	0.732	**0.813**	**0.841**	**0.861**	**0.878**

Table 6.2: Node classification for varying training sizes on *Cora*. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

	2%	4%	6%	8%	10%	30%	50%	70%	90%
HOPE	0.568	0.597	0.610	0.619	0.621	0.629	0.634	0.632	0.633
	0.464	0.501	0.514	0.524	0.524	0.533	0.540	0.538	0.539
Node2Vec	0.606	0.615	0.612	0.615	0.617	0.629	0.629	0.626	0.632
	0.503	0.522	0.491	0.493	0.495	0.535	0.528	0.524	0.532
NetSMF	0.597	0.618	0.629	0.628	0.626	0.644	0.648	0.647	0.650
	0.511	0.536	0.548	0.543	0.521	0.565	0.578	0.578	0.580
LouvainNE	0.490	0.511	0.502	0.508	0.497	0.504	0.510	0.511	0.515
	0.348	0.393	0.365	0.376	0.343	0.359	0.374	0.376	0.395
RandNE	0.399	0.415	0.412	0.421	0.425	0.438	0.435	0.436	0.438
	0.201	0.232	0.224	0.242	0.241	0.257	0.254	0.255	0.253
NodeSketch	0.516	0.584	0.615	0.645	0.666	0.791	**0.848**	**0.881**	**0.902**
	0.393	0.501	0.544	0.584	0.614	0.767	**0.831**	**0.868**	**0.890**
NodeSig	**0.613**	**0.648**	**0.672**	**0.692**	**0.705**	**0.797**	0.844	0.873	0.896
	0.523	**0.581**	**0.616**	**0.641**	**0.661**	**0.772**	0.825	0.858	0.882

Table 6.3: Node classification for varying training sizes on *DBLP*. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

	2%	4%	6%	8%	10%	30%	50%	70%	90%
HOPE	0.084	0.100	0.107	0.122	0.133	0.162	0.152	0.142	0.151
	0.042	0.057	0.063	0.072	0.083	0.101	0.084	0.082	0.081
NODE2VEC	0.093	0.114	0.123	0.128	0.138	0.168	0.164	0.150	0.140
	0.048	0.066	0.074	0.078	0.082	0.100	0.089	0.079	0.069
NETSMF	0.063	0.064	0.064	0.065	0.067	0.065	0.063	0.058	0.053
	0.016	0.016	0.016	0.016	0.018	0.016	0.016	0.015	0.014
LOUVAINNE	0.059	0.046	0.049	0.046	0.041	0.048	0.060	0.052	0.060
	0.028	0.025	0.027	0.026	0.022	0.023	0.025	0.022	0.023
RANDNE	0.096	0.114	0.124	0.137	0.144	0.169	0.160	0.135	0.140
	0.053	0.064	0.075	0.080	0.088	0.100	0.090	0.081	0.080
NODESKETCH	0.072	0.098	0.122	0.131	0.153	0.205	0.224	0.234	0.237
	0.036	0.055	0.076	0.082	0.101	0.158	0.180	0.192	0.187
NODESIG	**0.097**	**0.125**	**0.153**	**0.166**	**0.177**	**0.220**	**0.233**	**0.242**	**0.254**
	0.051	**0.073**	**0.095**	**0.111**	**0.118**	**0.167**	**0.183**	**0.191**	**0.194**

Table 6.4: Node classification for varying training sizes on *PPI*. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

	2%	4%	6%	8%	10%	30%	50%	70%	90%
HOPE	0.346	0.353	0.348	0.344	0.341	0.341	0.344	0.344	0.345
	0.213	0.222	0.208	0.202	0.197	0.199	0.205	0.202	0.203
NODE2VEC	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-
NETSMF	0.371	0.384	0.387	0.381	0.386	0.373	0.373	0.360	0.364
	0.254	0.271	0.275	0.266	0.265	0.248	0.249	0.239	0.236
LOUVAINNE	0.179	0.247	0.236	0.249	0.249	0.252	0.249	0.249	0.254
	0.067	0.066	0.063	0.062	0.063	0.062	0.063	0.065	0.071
RANDNE	0.320	0.334	0.335	0.337	0.335	0.342	0.343	0.339	0.343
	0.196	0.207	0.212	0.212	0.204	0.218	0.222	0.218	0.221
NODESKETCH	0.399	0.416	0.427	0.433	0.440	0.459	**0.467**	**0.471**	**0.472**
	0.294	0.327	0.346	0.355	0.367	0.399	**0.412**	**0.421**	**0.422**
NODESIG	**0.429**	**0.445**	**0.450**	**0.454**	**0.456**	**0.464**	0.466	0.469	0.467
	0.332	**0.362**	**0.372**	**0.380**	**0.388**	**0.402**	0.408	0.413	0.411

Table 6.5: Node classification for varying training sizes on *Youtube*. For each method, the rows indicates the Micro-F_1 and Macro-F_1 scores, respectively.

NodeSig model along with NodeSketch perform equally well. This is quite surprising, since both these methods that correspond to data-independent hashing techniques offer a clear performance gain over traditional models, such as Node2Vec and HOPE. Lastly, for the *PPI* dataset, NodeSig obtains consistently the highest scores for Micro-F_1, while its main competitor, NodeSketch, has close performance for the Macro-F_1 score.

6.4.3 *Link Prediction*

The second downstream task used to assess the quality of node embeddings is the one of link prediction.

EXPERIMENTAL SET-UP. As we have described in the previous chapter, half of the edges of a given network are removed by still keeping the residual network connected. Node embeddings are learned on the rest of the graph. As it has been described in Section 6.4.2, we build the features corresponding to the node pair samples using the similarities between embedding vectors; the similarity measure is chosen depending on the algorithm that we use to extract the representations. Since the *Youtube* dataset is relatively larger than the rest of the networks, we work on 7% of its initial size. We predict edges by constructing the similarity list of edges, and we provide the *Area Under Curve* (AUC) scores in Table 6.6.

EXPERIMENTAL RESULTS. For the link prediction task, NodeSig acquires the highest AUC scores on three datasets, while it is also the second best performing model for the remaining two. In the case of the *Youtube* dataset, all baselines demonstrate comparable results. Although Node2Vec shows good performance across most datasets in the link prediction task, it does not perform well on the *Blogcatalog* networks, mainly because of its high density. On the other hand, NodeSig reaches the highest score on this dataset, with a clear difference to its main competitor, NodeSketch.

	Blogcatalog	*Cora*	*DBLP*	*PPI*	*Youtube*
HOPE	0.517	0.662	0.769	0.524	0.514
NODE2VEC	0.595	**0.747**	0.844	0.615	0.531
NETSMF	0.570	0.717	0.832	0.549	**0.541**
LOUVAINNE	0.501	0.698	0.785	0.575	0.529
RANDNE	0.622	0.590	0.697	0.504	0.511
NODESKETCH	0.703	0.710	0.714	0.512	0.510
NODESIG	**0.822**	0.740	**0.860**	**0.655**	0.536

Table 6.6: Area Under Curve (AUC) scores for link prediction.

6.4.4 *Parameter Sensitivity*

Next, we analyze how the behavior of the proposed NODESIG algorithm is affected by the parameter setting. More specifically, we concentrate on the influence of three parameters, namely walk length $\mathcal{L}$, importance factor α and dimension size d, examining the impact on the *Cora* network.

EFFECT OF WALK LENGTH. In order to examine the influence of the walk length on the performance, we perform experiments for varying lengths by fixing the importance factor α to 1.0. Figure 6.3 depicts the Micro-F_1 scores for different training ratios. We observe a significant increase in performance when the walk length increases, particularly for small training ratios and walk lengths. Although it shows a wavy behavior for the largest training ratio, there is a logarithmic improvement depending on the walk length. NODESIG better captures the structural properties of the network in longer walks, thus the low performance observed on small training ratios can be compensated with longer walks.

EFFECT OF IMPORTANCE FACTOR. The importance factor is another parameter of NODESIG, which controls the impact of walks of different lengths: the importance of the higher levels is increasing for $\alpha > 1$, while it can be diminished choosing $\alpha < 1$. Figure 6.3 depicts the performance of NODESIG on the *Cora* network, fixing the walk length value to 3. Although we do not observe a steady behavior for the large training set, higher values

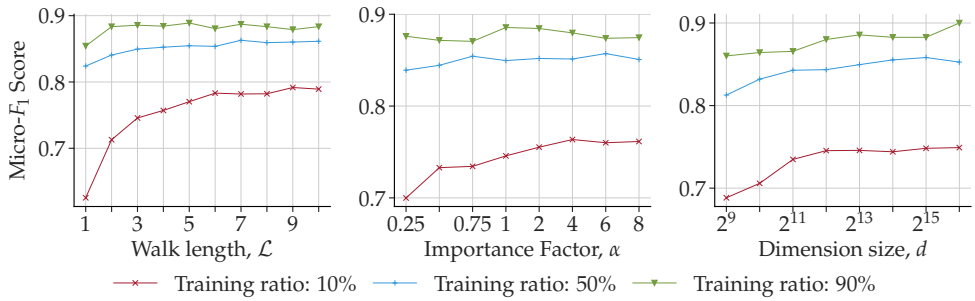


Figure 6.3: Influence of various parameters of the NODESIG model in terms of Micro-F_1 score on the *Cora* network for varying training set.

of α, especially around 4, positively contribute to the performance; values smaller than 1 have negative impact on the performance.

EFFECT OF DIMENSION SIZE. The dimension size is a crucial parameter affecting the performance of the algorithm, since a better approximation to the *chi* similarity measure can be obtained for larger dimension sizes, following *Hoeffding's* inequality [Hoe63]. Therefore, we perform experiments for varying dimension sizes, by fixing the walk length to 3. Figure 6.3 depicts the Micro-F_1 scores of the classification experiment for different dimension sizes ranging from 2^9 to 2^{17}. Although we have fluctuating scores on the large training set due to the randomized behavior of the approach, the impact of the dimension size can be observed clearly on the small training set size. On the other hand, we observe an almost stable behavior for the training ratio of 50%, encouraging the use of small embedding sizes towards reducing storage requirements.

6.4.5 *Running Time Comparison*

The running time is of great importance, since the embedding models are expected to scale and run in reasonable time on large graphs. We have recorded the elapsed real (wall clock) time of all baseline models including the one of NODESIG, and the results are provided in Table 6.7. The *Random* network indicates the $\mathcal{G}_{n,p}$ Erdös-Renyi random graph model [ER60], using

	Blogcatalog	*Cora*	*DBLP*	*PPI*	*Youtube*	*Random*	*Speedup*
HOPE	89	26	185	32	9183	1117	1.8x
NODE2VEC	1187	15	164	62	-	749	5.6x
NETSMF	2241	15	183	13	5399	454	1.4x
LOUVAINNE	0.29	0.06	0.21	0.13	5.70	1.2	0.001x
RANDNE	3.18	1.50	6.22	2.02	161.8	23	0.03x
NODESKETCH	106.84	13.46	21.93	9.26	2300	190	0.45x
NODESIG	120	4.38	41	14	5508	209	1.0x

Table 6.7: Running time (in seconds) and average speedup.

$n = 100,000$ and $p = 0.0001$. All the experiments have been conducted on the server whose specifications have been given in the beginning of Section 6.4. We use 32 threads for each algorithm, when it is applicable. If a model cannot run due to excessive memory usage with the parameter settings described in Section 6.4.1, the parameters are set to values closest to its default parameters, which enables the models to run in reasonable time. For NODESIG, the walk length is set to 5, and α to 1 in all experiments.

As we observe, NODESIG runs faster than HOPE, NODE2VEC as well as NETSMF. This is happening because HOPE requires an expensive matrix factorization, while NODE2VEC needs to simulate random walks to obtain their exact realizations. Although NETSMF has been proposed as the scalable extension of another matrix factorization model [Qiu+18], we have observed that it requires high memory footprint; therefore, we could not run it with the default parameters specified by the authors of the corresponding paper. Furthermore, although the remaining baseline methods run faster compared to NODESIG, as we have already presented, the proposed model generally outperforms them both in classification and link prediction tasks. These experiments further support the intuition about designing NODESIG, as an expressive model that balances accuracy and running time.

6.5 DISCUSSION FOR DYNAMIC NETWORKS

The majority of existing network representation learning models have been developed for static networks. Nevertheless, most real-world networks undergo structural changes and evolve over time with the addition and removal of links and nodes [Kaz+20]. Therefore, being able to design models that properly adapt to dynamic networks is an important point to investigate. As we discuss here, the proposed method allows for efficient updates of the embeddings, without requiring any costly learning procedures.

More precisely, the key point in the dynamic case, is that the learned embedding vectors should be efficiently updated instead of being recalculated from scratch. If an edge is added or removed for a pair of nodes $(u, v) \in V \times V$, the terms $\mathcal{R}^{(l)}_{(w,:)}$ in Equation (6.3) for node $w \in V$ are affected, for all $l > k := \min\{dist(w, u), dist(w, v)\}$—thus, it suffices to update only these affected terms. The transition probabilities for nodes u and v also change even though the remaining nodes are not affected, so all the terms $\mathbf{P}_{(v,:)}$ must be divided by $\sum_{w \in \mathcal{N}(v)} \mathbf{P}_{(v,w)}$ in order to normalize the transition probabilities and similarly the same procedure must also be applied to node u after each edge insertion and deletion operation.

Evidently, more research is required towards this direction in order to provide a method that scales well when the input data changes rapidly (i.e., in the form of a stream).

6.6 CONCLUSIONS AND FUTURE WORK

In this chapter, we have introduced NodeSig, an efficient binary node embedding model. Each component of model has properly been designed to improve scalability without sacrificing effectiveness on downstream tasks. NodeSig exploits random walk diffusion probabilities via stable random projection hashing, towards efficiently computing representations in the Hamming space that approximate the *chi* similarity. The experimental results have demonstrated that NodeSig outperformed in accuracy recent highly-scalable models, being able to run within reasonable time duration, while at the same time it shows comparable or even better accuracy with respect

to widely used baseline methods in multi-label node classification and link prediction. As future work, we plan to further study the properties of the model for attributed and dynamic networks and also study the performance of parallel/distributed alternatives.

7

CONCLUSION

The appearance of graph-structured data in many application domains has stirred the development of graph learning tools. In recent years, the focus of research studies has shifted towards techniques that automatically extract feature vectors, thereby encouraging the interest in graph representation learning. These features are further used to perform various downstream tasks, so numerous approaches have been developed targeting different aspects of the problem. In this dissertation, we have considered methods leveraging random walk diffusions to improve the effectiveness and efficiency of node embedding methods.

7.1 SUMMARY OF CONTRIBUTIONS

We have introduced several algorithms addressing various challenges problems in the field. The main contributions of the thesis can be compiled as follows.

Explicit Integration of Community Information

The community structure is one of the key aspects of complex networks, distinguishing them from random graph structures. A community is a group of nodes having higher connections among them than the rest of the network, where these nodes can share common characteristics. In light of this fact, in Chapter 3 we have proposed a novel framework, Topical Node Embedding (TNE), which explicitly incorporates the underlying community structure in the representation learning process. TNE extracts the latent node community assignments with various community detection methods and statistical models relying on graphs. Then, it learns node and community embeddings independently by means of the generated node sequences, leveraging the

SkipGram model. The final representation of nodes is obtained by concatenating their embedding vectors with the community representations. The extensive experimental evaluation has showed that the integration of community information in learning embeddings enhances the node representations' performance in downstream tasks.

Modeling Node Interactions with Exponential Family Distributions

Most of the random walk-based models presented in the literature, rely on [Mik+13a] to model the interactions between nodes appearing in the generated node sequences. Nevertheless, this could cause limitations concerning the expressiveness of the model. To address this point, we have introduced the EFGE model in Chapter 4, which leverages exponential family conditional distributions. In particular, we have employed three instances of exponential family distributions to represent the interactions among nodes and learn their representations. We have further shown the connection between BigClam, a well-known overlapping community detection method, and the variants of the EFGE model under appropriate parameter settings. The experimental evaluation has demonstrated the benefit of exponential family distributions on the performance of downstream tasks.

A Kernelized Matrix Factorization Framework Based on Random Walks

Kernel functions are generally applied on linear models to enhance their modeling capacity, allowing them to capture complex non-linear structures existing in data. In Chapter 5, we have adopted universal kernels under a matrix factorization framework to learn node representations. We have proposed KernelNE, a model which leverages random walks to mitigate the computational burden of the optimization step and the exact realization of the target matrix. To further strengthen the predictive capabilities, we have introduced MKernelNE, a multiple kernel learning formulation of the model. The experiments have shown that we could learn more expressive

embedding vectors grasping the complex co-occurrence information patterns of nodes within the random walks by means of kernel functions.

Scalable Graph Representation Learning

With the growing size of networks, many learning-based graph representation learning models become inapplicable to large-scale networks. Therefore, in recent years, the research community has concentrated on developing approaches that take advantage of sketching techniques to overcome this problem. In Chapter 6, we have introduced a novel approach, NodeSig, which is capable of handling networks consisting of millions of nodes and edges. NodeSig learns binary representations in the Hamming space utilizing sign α-stable random projections, which allows to approximate the *chi* similarity between random walk-based node representations. We have further developed an approach to carry out projections of the features in a recursive way without the need of realizing the exact matrix. We have shown that the proposed approach balances the trade-off between the performance in downstream tasks and the computational burden for large-scale networks.

7.2 FUTURE DIRECTIONS

Although we have observed an immense increase in the number of research studies in graph representation learning over the past few years, there are still various subjects worthy of discussion. In this section, we will briefly describe ongoing and possible research topics for future work.

Dynamic Networks

Most graph representation learning approaches have primarily focused on static networks. Nevertheless, many real-world networks naturally undergo changes over time. For instance, each time a new user joins a social network platform, the connections that she makes to other users cause a change in the network structure and contribute to the evolution of the network. Therefore,

developing efficient representation learning techniques for dynamic networks is of great significance. Classical representation learning methods can be used for learning the embeddings of a dynamic network at each particular state; the embedding vectors though should be updated concerning the modifications occurring in the network. For instance, in the case of random walk-based methods, which this dissertation also relies on, can be adapted for dynamic networks. A possible idea might be to perform random walks only for the nodes which have been affected by the change that happened in the network, within a certain time interval. If a node is removed from the network, then we can also remove the existing walks containing it, or we can perform new random walks for an inserted node [Ngu+18; Kaz+20]. Then, the node representations can be updated based on this new set of walks. Dynamic networks might also possess additional rich semantics, including node and edge features, thereby taking this additional information into account while updating the embeddings could constitute another future research direction.

Scalable Algorithms

Networks consisting of millions of nodes and edges have recently become ubiquitous in various disciplines, such as biology, recommender systems, and social sciences—making scalability a core property in the algorithm design process. As we have discussed in Chapter 2 and Chapter 6, approaches relying on sketching techniques have recently become prominent in the field, as they constitute a quite effective framework in computing embeddings. The idea mainly relies on first constructing feature vectors corresponding to the nodes of the graph; these vectors are then given as input to a hashing algorithm to compute signatures, which are used as embeddings. The input data for the sketching algorithms should also be carefully designed. Choosing simple vectors, such as the rows of the adjacency matrix, might not convey sufficient information about node proximity, thus causing limitations on the predictions on downstream tasks. On the other hand, designing higher-order proximity matrices requires high computational cost for large-scale networks. Therefore, building informative node features in an efficient

way that can be used as input to hashing algorithms, is still a crucial problem to investigate. It can be tackled with recursive techniques, provided that it is allowed by the sketching scheme, as we have adopted in the NodeSig method proposed in Chapter 6. Nevertheless, the chosen sketching technique should also be compatible with the assumptions of the feature data, including the distance measure.

There are mainly two different hashing techniques: locality sensitive hashing (LSH) and learning to hash [Wan+18]. LSH algorithms are data-independent, and have extensively been employed in various domains, ranging from natural language processing to computer vision. A typical example is the sign random projections technique, which learns binary representations preserving the angular similarity of the input vectors [LSH13]. Although graph representation learning methods relying on LSH propose fast sketching schemes, their performance is often limited compared to more traditional learning-based models. Learning to hash approaches can be an alternative to overcome this limitation, constituting a promising future research direction. They aim to learn hash functions for a given dataset by minimizing the discrepancy between the similarities in the original space and the embedding space. Learning to hash enables to adapt a wide range of various hash functions. For instance, one can incorporate kernel functions [HLC10] with the data vectors of nodes in order to learn the projection matrix, leading to better performance on downstream tasks. Moreover, as we have discussed in the previous section, adapting such methods to compute embedding in large-scale dynamic networks constitutes another interesting direction.

Graph Neural Networks

Graph Neural Networks (GNNs) [HYL17b; Ham20] provide effective tools for graph analysis tasks by learning node or network embeddings in a lower-dimensional space. They constitute powerful models enabling to incorporate side information such as node features with the network structure in the representation learning process. Nevertheless, GNNs require expensive computational resources, especially for large networks—making scalability a vital property to investigate.

The early GNN models [KW17; Vel+18] have required the whole network structure and the features in the memory during the optimization step. Therefore, they were not very suitable especially for large-scale networks. Furthermore, graph sampling strategies [HYL17a; Ham20] have become the leading technique to cope with this problem. The idea basically relies on sampling a fixed number of nodes from the k-hop neighborhood of each node. It also acts as an edge dropout process and thus, enhancing performance [Ron+20]. Nevertheless, on large-scale graphs, even the sampling process itself is challenging. The number of nodes within the k-hop neighborhood drastically increases even for small k values, since the graph structure is non-Euclidean for many real-world networks. Node sampling might also result in the loss of particular patterns of the network and the deprivation of influential nodes. Therefore, as future work, it is interesting to examine how to properly leverage random walks to address the scalability limitation of GNN models [Yin+18], while at the same time, preserving key structural properties of the underlying graph that could impact the accuracy on downstream tasks.

Although many GNNs relying on deep architectures [Wu+21] have been developed, shallow models also demonstrate comparable performance with less computational complexity [Wu+19]. Besides, higher number of layers also leads to over-smoothing. Therefore, a promising direction is to investigate how to adapt shallow models with rich diffusion functions towards designing scalable GNN models, without sacrificing the predictive performance [KWG19; Fra+20].

BIBLIOGRAPHY

[Ahm+13] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. Distributed Large-Scale Natural Graph Factorization. In *WWW*, Association for Computing Machinery, 2013, 37–48 (cit. on p. 15).

[AB02] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.* 74 (2002), pp. 47–97 (cit. on p. 2).

[Alp10] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2010 (cit. on p. 25).

[AS10] C. Alzate and J. A. K. Suykens. Multiway Spectral Clustering with Out-of-Sample Extensions through Weighted Kernel PCA. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32:2 (2010), pp. 335–347 (cit. on p. 74).

[AO04] L. A. N. Amaral and J. M. Ottino. Complex networks. *The European Physical Journal B* 38:2 (2004), pp. 147–162 (cit. on p. 1).

[AYC11] S. An, J. Yun, and S. Choi. Multiple kernel nonnegative matrix factorization. In *ICASSP*, 2011, pp. 1976–1979 (cit. on p. 74).

[And70] Erling Andersen. Sufficiency and Exponential Families for Discrete Sample Spaces. *Journal of the American Statistical Association.* 65:331 (1970), pp. 1248–1255 (cit. on p. 54).

[Arc96] Dan Archdeacon. Topological Graph Theory – A Survey. *CONG. NUM* 115 (1996), pp. 115–5 (cit. on p. 13).

[BLJ04] Francis R. Bach, Gert R. G. Lanckriet, and Michael I. Jordan. Multiple Kernel Learning, Conic Duality, and the SMO Algorithm. In *ICML*, 2004, p. 6 (cit. on p. 74).

[BN01] Mikhail Belkin and Partha Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *NIPS*, Cambridge, MA, USA, 2001, pp. 585–591 (cit. on p. 71).

[BCV13] Y. Bengio, A. Courville, and P. Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35:8 (2013), pp. 1798–1828 (cit. on p. 2).

[BG19] Dimitris Berberidis and Georgios B. Giannakis. Node Embedding with Adaptive Similarities for Scalable Learning over Graphs. In *IEEE Transactions on Knowledge and Data Engineering*, 2019, pp. 1307–1321 (cit. on p. 47).

[Bho+20] Ayan Kumar Bhowmick, Koushik Meneni, Maximilien Danisch, Jean-Loup Guillaume, and Bivas Mitra. LouvainNE: Hierarchical Louvain Method for High Quality and Scalable Network Embedding. In *WSDM*, 2020, pp. 43–51 (cit. on pp. 102, 112).

[BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research.* 3:1 (2003), pp. 993–1022 (cit. on p. 36).

[Blo+08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *J. Stat. Mech.* 2008:10 (2008), P10008 (cit. on pp. 31, 38, 93, 99).

[Boj+20] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling Graph Neural Networks with Approximate PageRank. In *KDD*, 2020, 2464–2473 (cit. on p. 20).

[Bot91] Leon Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nimes. EC2*, 1991 (cit. on pp. 62, 79).

[Bro97] A. Broder. On the Resemblance and Containment of Documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, 1997, p. 21 (cit. on p. 21).

[CZC18] HongYun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Trans. Knowl. Data Eng.* 30:9 (2018), pp. 1616–1637 (cit. on pp. 2, 9).

[CLX16] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep Neural Networks for Learning Graph Representations. In *AAAI*, 2016 (cit. on p. 19).

[CLX15] Shaosheng Cao, Wei Lu, and Qiongkai Xu. GraRep: Learning Graph Representations with Global Structural Information. In *CIKM*, 2015, pp. 891–900 (cit. on pp. 16, 71, 75).

[Cav+17] Sandro Cavallari, Vincent W. Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. Learning Community Embedding with Community Detection and Node Embedding on Graphs. In *CIKM*, 2017, pp. 377–386 (cit. on p. 32).

[ÇM20] Abdulkadir Çelikkanat and Fragkiskos D. Malliaros. Exponential Family Graph Embeddings. In *AAAI*, 2020, pp. 3357–3364 (cit. on pp. vii, 5, 71, 104).

[ÇM19a] Abdulkadir Çelikkanat and Fragkiskos D. Malliaros. Kernel Node Embeddings. In *GlobalSIP*, 2019, pp. 1–5 (cit. on pp. vii, 5, 104).

[ÇM19b] Abdulkadir Çelikkanat and Fragkiskos D. Malliaros. Learning Node Embeddings with Exponential Family Distributions. In *NeurIPS Graph Representation Learning Workshop (NeurIPS-GRL) Workshop*, 2019 (cit. on pp. vii, 5).

[ÇM18] Abdulkadir Çelikkanat and Fragkiskos D. Malliaros. TNE: A Latent Model for Representation Learning on Networks. In *NeurIPS Relational Representation Learning (NeurIPS-R2L) Workshop*, 2018 (cit. on pp. vii, 4).

[ÇM21] Abdulkadir Çelikkanat and Fragkiskos D. Malliaros. Topic-Aware Latent Models for Representation Learning on Networks. *Pattern Recognition Letters* 144 (2021), pp. 89–96 (cit. on pp. vii, 4, 104).

[ÇPM21] Abdulkadir Çelikkanat, Apostolos N. Papadopoulos, and Fragkiskos D. Malliaros. NodeSig: Random Walk Diffusion meets Hashing for Scalable Graph Embeddings. *Manuscript* (2021) (cit. on pp. vii, 6).

[ÇSM21] Abdulkadir Çelikkanat, Yanning Shen, and Fragkiskos D. Malliaros. Multiple Kernel Representation Learning on Networks. *Manuscript* (2021) (cit. on pp. vii, 5).

[CM20] Sudhanshu Chanpuriya and Cameron Musco. InfiniteWalk: Deep Network Embeddings as Laplacian Embeddings with a Nonlinearity. In *KDD*, 2020, 1325–1333 (cit. on p. 71).

[Che+18] Haochen Chen, Bryan Perozzi, Yifan Hu, and S. Skiena. HARP: Hierarchical Representation Learning for Networks. In *AAAI*, 2018 (cit. on p. 22).

[Che+19] Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. Fast and Accurate Network Embeddings via Very Sparse Random Projection. In *CIKM*, 2019, pp. 399–408 (cit. on pp. 21, 102).

[DeV07] C.L. DeVito. *Harmonic analysis: a gentle introduction*. Jones and Bartlett Publishers, 2007 (cit. on p. 36).

[DGK04] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel K-means: Spectral Clustering and Normalized Cuts. In *KDD*, 2004, pp. 551–556 (cit. on pp. 72, 74).

[EY36] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika* 1:3 (1936), pp. 211–218 (cit. on p. 76).

[ER60] P. Erdős and A Rényi. On the Evolution of Random Graphs. In *Publication of the Mathematical Institute of The Hungarian Academy of Sciences*, 1960, pp. 17–61 (cit. on pp. 46, 91, 119).

[FHK14] Fei Zhu, P. Honeine, and M. Kallas. Kernel nonnegative matrix factorization without the pre-image problem. In *MLSP*, 2014, pp. 1–6 (cit. on p. 74).

[Fen+18] Rui Feng, Yang Yang, Wenjie Hu, Fei Wu, and Yueting Zhuang. Representation Learning for Scale-free Networks. In *AAAI*, 2018 (cit. on p. 2).

[For10] Santo Fortunato. Community detection in graphs. *Physics Reports* 486:3 (2010), pp. 75 –174 (cit. on p. 1).

[Fra+20] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Benjamin Chamberlain, Michael Bronstein, and Federico Monti. SIGN: Scalable Inception Graph Neural Networks. In *ICML GRL+ Workshop*, 2020 (cit. on p. 128).

[GN02] M. Girvan and M. E. J. Newman. Community Structure in Social and Biological Networks. *PNAS* 99:12 (2002), pp. 7821–7826 (cit. on p. 32).

[GBB18] Vladimir Gligorijević, Meet Barot, and Richard Bonneau. deepNF: deep network fusion for protein function prediction. *Bioinformatics* 34:22 (2018), pp. 3873–3881 (cit. on p. 1).

[GW95] Michel X. Goemans and David P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. ACM* 42 (1995) (cit. on p. 106).

[GA11] Mehmet Gönen and Ethem Alpaydın. Multiple Kernel Learning Algorithms. *J. Mach. Learn. Res.* 12 (2011) (cit. on pp. 72, 74, 80, 82).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016 (cit. on p. 19).

[GF18] Palash Goyal and Emilio Ferrara. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge-Based Systems* 151 (2018), pp. 78–94 (cit. on pp. 2, 9, 20).

[GS04] Thomas L. Griffiths and Mark Steyvers. Finding scientific topics. *PNAS* 101 (2004) (cit. on p. 42).

[GL16] Aditya Grover and Jure Leskovec. Node2Vec: Scalable Feature Learning for Networks. In *KDD*, 2016, pp. 855–864 (cit. on pp. 18, 22, 25, 27, 40, 57, 71, 83, 104).

[Ham20] William L. Hamilton. *Graph Representation Learning*. Morgan and Claypool Publishers, 2020 (cit. on pp. 2, 9, 14, 20, 127, 128).

[HYL17a] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *NIPS*, 2017 (cit. on pp. 20, 128).

[HYL17b] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.* 40:3 (2017), pp. 52–74 (cit. on pp. 9, 14, 20, 99, 127).

[HLC10] Junfeng He, Wei Liu, and Shih-Fu Chang. Scalable similarity search with optimized kernel hashing. In *SIGKDD*, ACM, 2010, pp. 1129–1138 (cit. on p. 127).

[HER09] Keith Henderson and Tina Eliassi-Rad. Applying Latent Dirichlet Allocation to Group Discovery in Large Graphs. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, 2009, 1456–1461 (cit. on p. 33).

[Hen+] Keith Henderson, Tina Eliassi-Rad, Spiros Papadimitriou, and Christos Faloutsos. „HCDF: A Hybrid Community Discovery Framework.“ In. *Proceedings of the 2010 SIAM International Conference on Data Mining (SDM)*, pp. 754–765 (cit. on p. 33).

[Hoc09] Michiel Hochstenbach. A Jacobi–Davidson type method for the generalized singular value problem. *Linear Algebra and its Applications* 431 (2009), pp. 471–487 (cit. on p. 16).

[Hoe63] Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association* 58 (1963), pp. 13–30 (cit. on p. 119).

[HSS08] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel Methods in Machine Learning. *The Annals of Statistics* 36:3 (2008), pp. 1171–1220 (cit. on pp. 15, 72, 74).

[HR11] P. Honeine and C. Richard. Preimage Problem in Kernel-Based Machine Learning. *IEEE Signal Processing Magazine* 28:2 (2011), pp. 77–88 (cit. on p. 78).

[Huo+09] V. T. L. Huong, D. Park, D. Woo, and Yunsik Lee. Centroid neural network with Chi square distance measure for texture classification. In *IJCNN*, 2009, pp. 1310–1315 (cit. on pp. 100, 107).

[JB20] Junteng Jia and Austion R. Benson. Residual Correlation in Graph Neural Network Regression. In *SIGKDD*, 2020, 588–598 (cit. on p. 20).

[JLS86] William B. Johnson, Joram Lindenstrauss, and Gideon Schechtman. Extensions of lipschitz maps into Banach spaces. *Israel Journal of Mathematics* 54 (1986), pp. 129–138 (cit. on p. 105).

[JC16] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Phil. Trans. R. Soc.* 374 (2065 2016) (cit. on p. 14).

[Kaz+20] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation Learning for Dynamic Graphs: A Survey. *Journal of Machine Learning Research* 21:70 (2020), pp. 1–73 (cit. on pp. 94, 110, 121, 126).

[KW17] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*, 2017 (cit. on pp. 20, 128).

[KT05] Jon Kleinberg and Eva Tardos. *Algorithm Design*. USA: Addison-Wesley Longman Publishing Co., Inc., 2005 (cit. on pp. 9, 11).

[KBG19] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*, 2019 (cit. on p. 20).

[KWG19] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion Improves Graph Learning. In *NIPS*, 2019, pp. 13333–13345 (cit. on p. 128).

[KJM20] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Appl. Netw. Sci.* 5:1 (2020) (cit. on pp. 72, 74).

[KW78] J.B. Kruskal and M. Wish. *Multidimensional Scaling*. Quantitative Applications in the Social Sciences. SAGE Publications, 1978 (cit. on p. 14).

[LFK09] Andrea Lancichinetti, Santo Fortunato, and János Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics* 11:3 (2009), p. 033015 (cit. on pp. 31, 32).

[LKF07] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *ACM Trans. Knowl. Discov. Data* 1:1 (2007) (cit. on p. 23).

[LM12] Jure Leskovec and Julian J. Mcauley. Learning to Discover Social Circles in Ego Networks. In *NIPS*, 2012, pp. 539–547 (cit. on p. 23).

[LG14] Omer Levy and Yoav Goldberg. Neural Word Embedding as Implicit Matrix Factorization. In *NIPS*, Montreal, Canada, 2014, 2177–2185 (cit. on p. 19).

[Li15] Ping Li. o-Bit Consistent Weighted Sampling. In *KDD*, 2015, 665–674 (cit. on p. 21).

[LSH13] Ping Li, Gennady Samorodnitsky, and John Hopcroft. Sign Cauchy Projections and Chi-Square Kernel. In *NIPS*, 2013 (cit. on pp. 106, 107, 127).

[Lia+18] Defu Lian, Kai Zheng, Vincent W. Zheng, Yong Ge, Longbing Cao, Ivor W. Tsang, and Xing Xie. High-Order Proximity Preserving Information Network Hashing. In *KDD*, 2018, pp. 1744–1753 (cit. on p. 100).

[LLF11] Y. Lin, T. Liu, and C. Fuh. Multiple Kernel Learning for Dimensionality Reduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2011), pp. 1147–1160 (cit. on p. 74).

[Liu+17] Li-Ping Liu, Francisco J. R. Ruiz, Susan Athey, and David M. Blei. Context Selection for Embedding Models. In *NIPS*, 2017, pp. 4816–4825 (cit. on p. 55).

[Liu+16] Xinyue Liu, Chara Aggarwal, Yu-Feng Li, Xiaugnan Kong, Xinyuan Sun, and Saket Sathe. Kernelized Matrix Factorization for Collaborative Filtering. In *SDM*, 2016, pp. 378–386 (cit. on pp. 72, 74).

[Liu+15] Yang Liu et al. Topical Word Embeddings. In *AAAI*, 2015, pp. 2418–2424 (cit. on p. 28).

[Lu+15] Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. Recommender system application developments: A survey. *Decision Support Systems* 74 (2015), pp. 12 –32 (cit. on p. 1).

[LZ11] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications* 390:6 (2011), pp. 1150–1170 (cit. on p. 2).

[MH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9:86 (2008), pp. 2579–2605 (cit. on p. 93).

[MV13] D. Fragkiskos Malliaros and Michalis Vazirgiannis. Clustering and Community Detection in Directed Networks: A Survey. *Physics Reports* (2013) (cit. on pp. 31, 33, 94).

[Mal+20] Fragkiskos D. Malliaros, Christos Giatsidis, Apostolos N. Papadopoulos, and Michalis Vazirgiannis. The core decomposition of networks: theory, algorithms and applications. *The VLDB Journal* 29:1 (2020), pp. 61–92 (cit. on p. 1).

[Mat+19] G. Mateos, S. Segarra, A. G. Marques, and A. Ribeiro. Connecting the Dots: Identifying Network Structure via Graph Signal Processing. *IEEE Signal Processing Magazine* 36:3 (2019), pp. 16–43 (cit. on p. 74).

[MXZ06] Charles A. Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal Kernels. *J. Mach. Learn. Res.* 7 (Dec. 2006) (cit. on p. 79).

[Mik+13a] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *ICLR*, 2013 (cit. on pp. 16, 29, 40, 52, 56, 84, 124).

[Mik+13b] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and Their Compositionality. In *NIPS*, 2013, pp. 3111–3119 (cit. on pp. 16, 17, 20, 27, 29, 38, 40, 41, 51, 58, 62, 71, 80).

[MH09] Andriy Mnih and Geoffrey E Hinton. A Scalable Hierarchical Distributed Language Model. In *NIPS*, vol. 21. 2009, pp. 1081–1088 (cit. on p. 17).

[MK13] Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *NIPS*, vol. 26. 2013, pp. 2265–2273 (cit. on pp. 17, 58).

[MS08] Andriy Mnih and Russ R Salakhutdinov. Probabilistic Matrix Factorization. In *NIPS*, vol. 20. 2008, pp. 1257–1264 (cit. on p. 94).

[MT12] Andriy Mnih and Yee Whye Teh. A Fast and Simple Algorithm for Training Neural Probabilistic Language Models. In *ICML*, 2012, 419–426 (cit. on p. 17).

[New06] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences* 103:23 (2006), pp. 8577–8582 (cit. on p. 30).

[New03] M. E. J. Newman. The Structure and Function of Complex Networks. *SIAM Review* 45:2 (2003), pp. 167–256 (cit. on p. 1).

[New10] Mark Newman. *Networks: An Introduction*. 2010 (cit. on p. 93).

[NM18] Duong Nguyen and Fragkiskos D. Malliaros. BiasedWalk: Biased Sampling for Representation Learning on Graphs. In *Big Data*, 2018, pp. 4045–4053 (cit. on pp. 18, 27, 57).

[Ngu+18] G. H. Nguyen, J. Boaz Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim. Dynamic Network Embeddings: From Random Walks to Temporal Random Walks. In *Big Data*, 2018 (cit. on p. 126).

[Ou+16] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric Transitivity Preserving Graph Embedding. In *KDD*, 2016, pp. 1105–1114 (cit. on pp. 15, 41, 71, 75, 76).

[Pal+05] Gergely Palla et al. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435 (2005), p. 814 (cit. on p. 35).

[PJA10] Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg. Locality sensitive hashing: a comparison of hash function types and querying mechanisms. *Pattern Recognition Letters* 31:11 (2010), pp. 1348–1358 (cit. on p. 21).

[Ped+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 23, 85, 93).

[PW10] Ofir Pele and Michael Werman. The Quadratic-Chi Histogram Distance Family. In *ECCV*, 2010, pp. 749–762 (cit. on p. 100).

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online Learning of Social Representations. In *KDD*, 2014, pp. 701–710 (cit. on pp. 18, 27, 36, 40, 57, 71, 104).

[Per+17] Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. Don'T Walk, Skip!: Online Learning of Multi-scale Network Embeddings. In *ASONAM*, 2017, pp. 258–265 (cit. on p. 22).

[Por+08] Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. Fast Collapsed Gibbs Sampling for Latent Dirichlet Allocation. In *KDD*, 2008, 569–577 (cit. on p. 40).

[Qiu+19] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization. In *WWW*, 2019, pp. 1509–1520 (cit. on pp. 76, 99, 112).

[Qiu+18] Jiezhong Qiu et al. Network Embedding As Matrix Factorization: Unifying DeepWalk, LINE, PTE, and Node2Vec. In *WSDM*, 2018, pp. 459–467 (cit. on pp. 16, 41, 76, 112, 120).

[Ray13] Santanu Saha Ray. *Graph Theory with Algorithms and its Applications*. Springer India, 2013 (cit. on pp. 9, 11).

[RSF17] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. Struc2vec: Learning Node Representations from Structural Identity. In *KDD*, 2017, 385–394 (cit. on p. 22).

[RIF02] Matei Ripeanu, Adriana Iamnitchi, and Ian Foster. Mapping the Gnutella Network. *IEEE Internet Computing* 6:1 (2002), pp. 50–57 (cit. on p. 23).

[RMG17] D. Romero, M. Ma, and G. B. Giannakis. Kernel-Based Reconstruction of Graph Signals. *IEEE Transactions on Signal Processing* 65:3 (2017), pp. 764–778 (cit. on p. 74).

[RIG17] Daniel Romero, Vassilis N. Ioannidis, and Georgios B. Giannakis. Kernel-Based Reconstruction of Space-Time Functions on Dynamic Graphs. *IEEE J. Sel. Top. Signal Process.* 11:6 (2017), pp. 856–869 (cit. on p. 74).

[Ron+20] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *ICLR*, 2020 (cit. on p. 128).

[RSooa] Sam T. Roweis and Lawrence K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science* 290:5500 (2000), pp. 2323–2326 (cit. on p. 15).

[RSoob] Sam T. Roweis and Lawrence K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science* 290:5500 (2000), pp. 2323–2326 (cit. on p. 71).

[Roz+19] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. GEMSEC: Graph Embedding with Self Clustering. In *ASONAM*, 2019, pp. 65–72 (cit. on pp. 32, 41).

[Rud+17] Maja Rudolph, Francisco Ruiz, Susan Athey, and David Blei. Structured Embedding Models for Grouped Data. In *NIPS*, 2017, pp. 251–261 (cit. on p. 55).

[Rud+16] Maja Rudolph, Francisco Ruiz, Stephan Mandt, and David Blei. Exponential Family Embeddings. In *NIPS*, 2016, pp. 478–486 (cit. on p. 55).

[SSM97] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *ICANN*, 1997, pp. 583–588 (cit. on pp. 72, 74).

[SS01] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001 (cit. on pp. 72, 74).

[Sen+08] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine* (2008) (cit. on p. 22).

[SBG17] Y. Shen, B. Baingana, and G. B. Giannakis. Kernel-Based Structural Equation Models for Topology Identification of Directed Networks. *IEEE Transactions on Signal Processing* 65:10 (2017), pp. 2503–2516 (cit. on p. 74).

[SJ03] Nathan Srebro and Tommi Jaakkola. Weighted Low-rank Approximations. In *ICML*, 2003, pp. 720–727 (cit. on p. 75).

[Ste02] Ingo Steinwart. On the Influence of the Kernel on the Consistency of Support Vector Machines. *J. Mach. Learn. Res.* 2 (2002), pp. 67–93 (cit. on pp. 77–79).

[Sur21] Surabhi Jagtap, Abdulkadir Çelikkanat, Aurélie Pirayre, Frederique Bidard, Laurent Duval and Fragkiskos D. Malliaros. Multiomics Data Integration for Gene Regulatory Network Inference with Exponential Family Embeddings. In *EUSIPCO*, 2021 (cit. on p. vii).

[Tan+15] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale Information Network Embedding. In *WWW*, 2015, pp. 1067–1077 (cit. on pp. 20, 41).

[TL09a] Lei Tang and Huan Liu. Relational Learning via Latent Social Dimensions. In *KDD*, 2009, pp. 817–826 (cit. on p. 111).

[TL09b] Lei Tang and Huan Liu. Scalable Learning of Collective Behavior Based on Sparse Social Dimensions. In *CIKM*, 2009, pp. 1107–1116 (cit. on p. 111).

[Tao13] T. Tao. *An Introduction to Measure Theory*. Graduate studies in mathematics. American Mathematical Society, 2013 (cit. on pp. 9, 12).

[TSL01] Joshua Tenenbaum, Vin Silva, and John Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science (New York, N.Y.)* 290 (Jan. 2001), pp. 2319–23 (cit. on p. 15).

[Tha+17] Alaa Tharwat, Tarek Gaber, Abdelhameed Ibrahim, and Aboul Ella Hassanien. Linear discriminant analysis: A detailed tutorial. *AI Communications* 30 (2017), pp. 169–190 (cit. on p. 14).

[Tia+19] Yu Tian, Long Zhao, Xi Peng, and Dimitris Metaxas. Rethinking Kernel Methods for Node Representation Learning on Graphs. In *NIPS*, 2019, pp. 11686–11697 (cit. on p. 74).

[VG+08] Jurgen Van Gael et al. Beam Sampling for the Infinite Hidden Markov Model. In *ICML*, Association for Computing Machinery, 2008, 1088–1095 (cit. on pp. 37, 40, 42).

[Vel+18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations* (2018) (cit. on p. 128).

[Vem01] Santosh Vempala. *The random projection method*. American Mathematical Soc., 2001 (cit. on p. 105).

[Vic79] B.C. Vickery. Reviews : van Rijsbergen, C. J. Information retrieval. 2nd edn. London, Butterworths, 1978. 208pp. *Journal of librarianship* 11:3 (1979), pp. 237–237 (cit. on p. 24).

[Vin+10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. 11 (2010), 3371–3408 (cit. on p. 19).

[WCZ16] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural Deep Network Embedding. In *KDD*, 2016, 1225–1234 (cit. on p. 19).

[Wan+18] Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. A Survey on Learning to Hash. *IEEE Trans. Pattern Anal. Mach. Intell.* 40:4 (2018), pp. 769–790 (cit. on pp. 100, 127).

[Wan+12] S. Wang, Q. Huang, S. Jiang, and Q. Tian. S^3MKL: Scalable Semi-Supervised Multiple Kernel Learning for Real-World Image Applications. *IEEE Transactions on Multimedia* 14 (2012), pp. 1259–1274 (cit. on p. 72).

[WZH21] Tinghua Wang, Lin Zhang, and Wenyu Hu. Bridging deep and multiple kernel learning: A review. *Information Fusion* 67 (2021), pp. 3 –13 (cit. on p. 74).

[Wan+17] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community Preserving Network Embedding. In *AAAI*, 2017, pp. 203–209 (cit. on pp. 2, 32, 41).

[Wu+19] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying Graph Convolutional Networks. In *Proceedings of the 36th International Conference on Machine Learning*, vol. 97. 2019, pp. 6861–6871 (cit. on p. 128).

[Wu+18] Wei Wu, Bin Li, Ling Chen, and Chengqi Zhang. Efficient Attributed Network Embedding via Recursive Randomized Hashing. In *IJCAI*, 2018, pp. 2861–2867 (cit. on pp. 21, 100, 102).

[Wu+21] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32:1 (2021), pp. 4–24 (cit. on p. 128).

[Yan+15] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. Network Representation Learning with Rich Text Information. In *IJCAI*, 2015, 2111–2117 (cit. on p. 15).

[Yan+19] Dingqi Yang, Paolo Rosso, Bin Li, and Philippe Cudre-Mauroux. NodeSketch: Highly-Efficient Graph Embeddings via Recursive Sketching. In *KDD*, 2019, pp. 1162–1172 (cit. on pp. 21, 100, 102, 112, 113).

[YL13] Jaewon Yang and Jure Leskovec. Overlapping Community Detection at Scale: A Nonnegative Matrix Factorization Approach. In *WSDM*, 2013, pp. 587–596 (cit. on pp. 5, 31, 38, 52, 60).

[Ye+15] Chen Ye, Jian Wu, Victor S. Sheng, Shiquan Zhao, Pengpeng Zhao, and Zhiming Cui. Multi-Label Active Learning with Chi-Square Statistics for Image Classification. In *ICMR*, 2015, pp. 583–586 (cit. on pp. 100, 107).

[YCP15] Xinyang Yi, Constantine Caramanis, and Eric Price. Binary Embedding: Fundamental Limits and Fast Algorithm. In *ICML*, 2015, pp. 2162–2170 (cit. on p. 105).

[Yin+18] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*, 2018, pp. 974–983 (cit. on p. 128).

[Zha+20] D. Zhang, J. Yin, X. Zhu, and C. Zhang. Network Representation Learning: A Survey. *IEEE Transactions on Big Data* 6:1 (2020), pp. 3–28 (cit. on p. 9).

[ZLZ18] Yuan Zhang, Tianshu Lyu, and Yan Zhang. COSINE: Community-Preserving Social Network Embedding From Information Diffusion Cascades. In *AAAI*, 2018, pp. 620–2627 (cit. on p. 32).

[Zha+18] Z. Zhang, P. Cui, H. Li, X. Wang, and W. Zhu. Billion-Scale Network Embedding with Iterative Random Projection. In *ICDM*, 2018, pp. 787–796 (cit. on pp. 21, 102, 112).