Aristotle University of Thessaloniki

Master's Thesis

Distributed Algorithms for Graph Embeddings

Author:
Dimitrios Tourgaidis

Supervisor:
Apostolos Papadopoulos

A dissertation submitted to the School of Informatics of Aristotle University of Thessaloniki in partial fulfillment of the requirements for the degree of Master of Science

THESSALONIKI, MARCH 2022

Contents

List of Figures

List of Tables

Abstract

Due to the continuously tendency to model domains in a network format, network representation learning (NRL) is a field where a lot of research has been carried out the recent years. Programmatically, a network is preserved in an adjacency matrix that preserves the neighbors for each network's node. The size of the adjacency matrix is $N \times N$, where N the number of nodes in a network. For large networks, utilizing the vectors of the network's nodes preserved by the adjacency matrix is unfeasible in order to train machine learning models to solve various network analytic tasks. The reason is the size of the vectors, which for large networks is so big that makes practically the training of the machine learning models unfeasible time-wise. The objective of NRL is to map the original representation space of a network (adjacency matrix) to a latent low-dimensional representation. The characteristics of the new node vectors (node embeddings) are **1)** the vectors' size is d, where $d \ll N$, making the training of machine learning models time-wise feasible and **2)** the vectors try to capture as much as possible characteristics of the original network to make the node embeddings as efficient as possible in regard to the machine learning models training. A lot of NRL methods have been developed and presented in the literature, which doesn't respect the scaling fact though, making them inapplicable in large networks. In 2018, Zhang and his colleagues presented RandNE, "A novel and simple billion-scale network embedding method based on high-order proximity preserved random projection" [1]. RandNE showed exceptional time-wise performance, but the provided implementation by [1] is **1)** centralized and **2)** was tested on small networks. Consequently, the objective of this thesis is the develop a distributed version of the RandNE method utilizing the Apache Spark framework, and examine its time-wise behavior. The implementation is available at

the following repository[1]. The results showed that the provided implementation produces feasible execution times on large networks. In case of a network with 334.863 nodes and 925.872 edges, a few minutes were enough in order the implementation to be executed. Except from the main objective of this thesis, this thesis is also a great entry-point for someone who is unfamiliar with the NRL field, since it covers fundamental concepts, notations and definitions of the field, as well as important NRL methods that have been developed in the literature.

[1]Distributed RandNE implementation repository

Περίληψη

Λόγω της συνεχούς τάσης για μοντελοποίηση τομέων σε μορφή δικτύου, η εκμάθηση αναπαράστασης δικτύου είναι ένα πεδίο πάνω στο οποίο έχει πραγματοποιηθεί πολλή έρευνα τα τελευταία χρόνια. Προγραμματιστικά, ένα δίκτυο διατηρείται σε έναν πίνακα γειτνίασης που διατηρεί τους γείτονες για κάθε κόμβο του δικτύου. Το μέγεθος του πίνακα γειτνίασης είναι $N \times N$, όπου N ο αριθμός των κόμβων σε ένα δίκτυο. Για μεγάλα δίκτυα, η χρήση των διανυσμάτων των κόμβων του δικτύου που διατηρούνται από τον πίνακα γειτνίασης, δεν είναι πρακτικά εφικτό να χρησιμοποιηθούν για την εκπαίδευση μοντέλων μηχανικής μάθησης, για την επίλυση διαφόρων αναλυτικών εργασιών δικτύου. Ο λόγος είναι το μέγεθος των διανυσμάτων, το οποίο για μεγάλα δίκτυα είναι τόσο μεγάλο που καθιστά πρακτικά ανέφικτη την εκπαίδευση των μοντέλων μηχανικής μάθησης, χρονικά. Ο στόχος της εκμάθηση αναπαράστασης δικτύου είναι να μετασχηματίσει τον αρχικό χώρο αναπαράστασης ενός δικτύου (πίνακας γειτνίασης) σε μια λανθάνων αναπαράσταση χαμηλών διαστάσεων. Τα χαρακτηριστικά των νέων διανυσμάτων των κόμβων είναι **1)** το μέγεθος τους είναι d, όπου $d \ll N$, καθιστώντας την εκπαίδευση των μοντέλων μηχανικής μάθησης χρονικά εφικτή και **2)** προσπαθούν να συλλάβουν όσο το δυνατόν περισσότερα στοιχεία του αρχικού δικτύου, ώστε να είναι όσο το δυνατόν πιο αποτελεσματικά σε σχέση με την εκπαίδευση μοντέλων μηχανικής μάθησης. Πολλές μέθοδοι έχουν αναπτυχθεί και παρουσιαστεί στη βιβλιογραφία για το πεδίο αυτό, οι οποίες όμως δεν αντιμετωπίζουν επιτυχώς το θέμα της κλιμάκωσης, δηλαδή να μπορούν να εφαρμοστούν σε μεγάλα δίκτυα, καθιστώντας τις ανεφάρμοστες σε τέτοιου τύπου δίκτυα. Το 2018, ο Zhang και οι συνεργάτες του παρουσίασαν την μέθοδο RandNE, "a novel and simple billion-scale network embedding method based on high-order proximity preserved random projection" [1]. Η μέθοδος RandNE έδειξε εξαιρετική απόδοση από άποψη χρόνου, αλλά η παρεχόμενη υλοποίηση από το [1] είναι **1)** υλοποιημένη ώστε να εκτελείται μόνο χρησιμοποιώντας

έναν πυρήνα και **2)** εξετάστηκε σε μικρά δίκτυα. Κατά συνέπεια, στόχος αυτής της διπλωματικής εργασίας είναι η ανάπτυξη μιας κατανεμημένης έκδοσης της μεθόδου RandNE, χρησιμοποιώντας το λογισμικό Apache Spark και η εξέταση της συμπεριφοράς της από άποψη χρόνου και κλιμάκωσης. Η υλοποίηση είναι διαθέσιμη στον ακόλουθο σύνδεσμο[2]. Τα αποτελέσματα έδειξαν ότι η παρεχόμενη υλοποίηση παράγει αποδεκτούς χρόνους εκτέλεσης σε μεγάλα δίκτυα. Σε περίπτωση δικτύου με 334.863 κόμβους και 925.872 ακμές, λίγα λεπτά ήταν αρκετά για να εκτελεστεί η υλοποίηση. Εκτός από τον κύριο στόχο αυτής της διπλωματικής εργασίας, επιπλέον αποτελεί ένα εξαιρετικό σημείο εισόδου για κάποιον που δεν είναι εξοικειωμένος με το συγκεκριμένο πεδίο, καθώς καλύπτει θεμελιώδεις έννοιες και ορισμούς, και παρουσιάζει σημαντικές μεθόδους που έχουν αναπτυχθεί μέχρι στιγμής στη βιβλιογραφία στο πεδίο αυτό.

[2]Distributed RandNE implementation repository

Acknowledgements

I would like to express my gratitude to my family for the support they provided to me the whole time, as well my supervisor Associate Professor Apostolos Papadopoulos who helped me with the completion of my thesis.

Chapter 1

Introduction

Many domains due to their nature provide the option to be modeled as a network (graph) that leads to a sophisticated representation of the domains' features. Typical examples of such networks are; social networks, telecommunication networks, biological networks, citation networks. The number of nodes (vertices) and edges in these networks depends on their nature and can be from hundreds to millions or billions.

Analyzing the network that represents a specific domain can lead to the resolution of important tasks, making the procedure of decision-making more efficient. For example, in social networks, more efficient targeted advertising and recommendations can be achieved by clustering the users based on their characteristics preserved in the network. In a biological network, analyzing the interaction among the proteins can result to new treatment for diseases.

Traditionally, a networks is represented using an adjacency matrix which preserves the neighboring relationships between the nodes. The problems with this type of representation are **1)** it does not preserve more complex relationships (higher-order structure relationships), consequently such type of relationships are computationally expensive to be extracted, and **2)** it requires a lot of memory to preserve the network, making it an insufficient data-structure. This two reasons make the usage of an adjacency matrix an insufficient option to store and handle large-scale networks.

In order to overcome the aforementioned problems, a lot of research has been carried out in the field of Network Representation Learning (Graph Embedding or Network Embedding), whose objective is to learn latent, low-dimensional representations of network nodes, while preserving the network topology structure, node content, and other side information. In NRL, each node is mapped from the vector representation of size $|V|$ (number of nodes in the network), stored in the adjacency matrix, to a vector representation of size $d \ll |V|$, resulting to a new representation space of the entire network. These new node vector-based representations (node embeddings) are then used to handle the network analytic tasks. Since the size of the each node's representation is now $d \ll |V|$ and they preserve more complex relationships, analytical tasks are more efficiently handled compared to the adjacency matrix case.

Unfortunately, most of the methods developed in the literature can be applied only to small networks, because of their execution time that is very high in large networks. Due to that, the experiments presented in those papers are not executed in large networks, but only in small networks. Inspired by this problem, this thesis tries to conduct a research on the field of scalable NRL. In 2018, RandNE [1] was presented, that is "a novel and simple billion-scale network embedding method based on high-order proximity preserved random projection" [1]. RandNE, due to its implementation nature, showed a great potential regarding its feasibility to be applied to large networks.

The implementation of RandNE that Zhang and his colleagues provide in [1] is centralized, so its does not utilize the benefits of distributed computing. Consequently, the objective of this thesis is **1)** to provide a distributed version of the RandNE algorithm and **2)** to examine the time-wise behavior of the provided implementation through experimentation, on a number of large networks. Since one of the objectives of the thesis is to develop RandNE in a distributed manner, the implementation is developed utilizing the Apache Spark[1] platform. Apache Spark is a software system that provides the capacity of developing distributed applications and constitutes one of the most efficient frameworks in this area.

The results of the performance evaluation showed that the implementation provided by the thesis is time-wise feasible to be executed in real case scenarios, since on a network that has 334.863 nodes and 925.872 edges, it needed only a few minutes in order to be executed.

Contributions. Next, the contributions of the thesis are presented briefly.

[1]Apache Spark website

1. It provides a distributed implementation of the RandNE method [1], developed utilizing the Apache Spark framework. The implementation code is available at the following repository[2]. An analysis of the implementation is provided too.

2. An analysis of the RandNE method is provided, as well of the Apache Spark framework.

3. An extensive experimentation is conducted that regard the implementation of the RandNE method provided by the thesis. Consequently, an analysis of the experimentation is provided too, in order understand the applicability of the implementation on large networks time-wise.

4. The thesis provides an introduction to the NRL field and discuss the applications, challenges, notations and definitions of the field too, for the substantive comprehension of the field.

5. A number of NRL methods developed in literature are presented and analyzed. The related work is presented through a structured taxonomy.

Thesis outline. This thesis consists of six chapters. Chapter 1 presents the objective of the thesis, as well an introduction of the field it concerns that is network representation learning (NRL), and the challenges, applications, note and definitions of the field. Then chapter 2 presents and analyzes the methods that have developed for the NRL field in the literature. Chapter 3 provides a detailed analyzes of the RandNE method [1]. Chapter 4, first analyzes the Apache Spark framework and then analyzes the code implementation of the RandNE methods provided by the thesis. Afterwards, chapter 5 presents the results of the experimentation for the implementation provided by the thesis. At last, chapter 6 presents a summary of the thesis, as well how the distributed RandNE implementation provided by the thesis could be utilized to examine other objectives.

[2]Distributed RandNE implementation repository

1.1 Challenges

Network Representation Learning is a field that is inherently difficult, since producing a new vector representation for each node in an efficient manner has many challenges and key-points that must be taken under consideration, and are discussed below.

Choice of features. Probably the most important concern for NRL methods is which features they are going to choice to preserve in the new representation space, through the produced vectors. The two feature types that can be preserved are; *structure-context* and *attribute-context* (if attributes are available). For the former, nodes that are close in the original network representation space or structurally similar should be also close in the new representation space taking under consideration local and global structure characteristics. As for the latter, except from structural similarity, two nodes can be similar in a content-level provided by their attributes. Therefore, in cases where two nodes are structurally distant from each other in the original representation space, but similar in attribute-context, after learning their new representations their must come closer in the new representation space.

Scalability. Most of real-world networks, like social networks, are enormous and contain millions or even billions nodes and edges. This event makes the appliance of methods almost unfeasible in cases where the methods haven't been developed taking under consideration the largeness of the networks, due to their time complexity. Consequently, a key-point in NRL is to develop methods that are sufficient even in large networks, regarding the execution time.

Dimensionality of vectors (embeddings). Finding which is the optimal size of the vectors in the new representation space is a hard task. On the one hand, deciding for a high number of dimension can be more efficient regarding the reconstruction precision, but will result in high time and space complexity. On the other hand, deciding for a low number of dimension can lead to low time and space complexity, but to low reconstruction precision too. Another key-point is that the decision concerning the dimension size, is application-depended. Consequently, a lot of considerations and try-error effort must be carried out in order to find the correct dimensionality.

1.2 Applications

Once the node embeddings are constructed, they can be used in a variate of vector-based algorithms to solve network analytic tasks. Below, fundamental network analytic tasks are analyzed, in which the produced node embeddings could be utilized; *node classification*, *link prediction*, *node clustering*, *network visualization.*

1.2.1 Node Classification

Probably the most important task to solve in a network is the one of node classification. In many cases, a network's node is associated with a semantic label indicating a specific characteristic of it; In a citation network, a label attached to a publication can indicate its topics or research areas. In a social network, a label attached to a user may indicate they religious, political believes or they general interests. An important feature in networks is that their nodes are partial labeled, due to the high labeling cost or lack of effort from the users to provide some information. Consequently, node classification is the task of predicting the missing labels of nodes, utilizing the networks structure and labels of other nodes. Furthermore, classification cannot only be applied to node-level, but also to subgraph-level and graph-level.

1.2.2 Link Prediction

Link prediction is one of the most fundamentals problems on network analysis, and a field where a lot research has been done to examine the effectiveness of NRL. Its objective is to predict missing edges between nodes that are likely to appear in the future, based on the observed network structure. In general, recommendation systems are using link prediction to propose products to users, like movies. Social networks are utilizing link prediction to recommend friends to users or to identify suspicious relationships. In biological networks, using link prediction can reduce the cost of experimental approaches by identifying previously unknown relationships between proteins.

1.2.3 Node Clustering

Node clustering regard the operation of grouping similar nodes, utilizing the network's structure and nodes' attributes, into a cluster, so the network is partitioned into a set of clusters (communities). Inside a cluster, the nodes are densely connected to each other, and some of them can have connections to nodes of other clusters. A cluster in biological networks can contain proteins with similar functions, while in a document network it can contain documents with the same topic. Clustering has been heavily used in social networks, to create communities of users with similar interests.

1.2.4 Network Visualization

Visualizing a network is a task that has long history and comes with an obstacle concerning the visualization of larger networks. An objective of this field is to find a network's representation as small as possible in order to be able to handle it, consequently NRL is a great match, because it aims to map a network's representation to a smaller representation space.

1.3 Notations And Definitions

Next, formal definitions of fundamentals concepts that are used in the NRL field are presented. Their comprehension is crucial in order to be able understand the analyzes of methods that have been developed for this area in the literature. Table 1.1 shows common notations used below, and their descriptions.

1.3.1 Undirected Network

A Graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ denotes a set that contains all the nodes of the Graph G and $E = \{e_{ij}\}_{i,j=1}^{n}$ a set that contains the corresponding edges, is undirected if each edge has no direction. When G is preserved in an adjacency matrix A, $\forall A_{ij}$ element of A shows if there is an edge from v_i to v_j, or not. In an undirected graph, when to nodes are connected, then $A_{ij} = A_{ji} = 1$, otherwise $A_{ij} = A_{ji} = 0$.

G	The Graph (Network)
V	The set of vertices of a graph
E	The set of edges of a graph
$\|V\|$	The number of nodes in a graph
$\|E\|$	The number of edges in a graph
d	Dimension of learned nodes representation
A	The adjacency matrix that preserves a graph
A_{ij}	The i-th row and j-th column in adjacency matrix A

TABLE 1.1: Summary of common notations

1.3.2 Directed Network

A Graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ denotes a set that contains all the nodes of the Graph G and $E = \{e_{ij}\}_{i,j=1}^{n}$ a set that contains the corresponding edges, is directed when all edges have direction. When G is preserved in an adjacency matrix A, $\forall A_{ij}$ element of A shows if there is an edge from v_i to v_j, or not. In a directed graph, when there is a connection from v_i to v_j, then $A_{ij} = 1$, but not automatically $A_{ji} = 1$ also, as the undirected case. There must be an additional edge from v_j to v_i in order to $A_{ji} = 1$.

1.3.3 First-order Proximity

First-order proximity is the local proximity between two nodes that are connected through an edge. If there is an edge that connects two nodes, then the edge indicates the first-order proximity between the two nodes, otherwise the first-proximity order between the nodes equals to 0. Essentially, first-order proximity captures the similarity of two nodes based on their neighbor relationship.

1.3.4 Second-order Proximity

Second-order proximity analyzes the 2-level proximity between two nodes by examining their neighborhood similarity. More particular, second-order proximity is captured by the number of common neighbors between two nodes.

1.3.5 High-order Proximity

High-order proximity can be used to calculate the k-order ($k \geq 3$) proximity between two nodes, which can capture similarity between pairs of nodes that are not directly connected through an edge (like second-order proximity), but emphasises on more global structure context compared to first-order and second-order proximity.

1.3.6 Structural-Role Proximity

Structural-role proximity captures the similarity between two nodes, based on their similar role in their neighborhood; a bridge between two communities, edge of a chain, center of star and etc. Compared to first-order, second-order and high-order proximity, that are trying to define similarities between nodes that are nearly located in the graph, structural-role proximity defines similarity for nodes that are distant.

1.3.7 Network Representation learning (NRL)

Given a Graph $G = (V, E)$, the objective of Network Representation Learning (NRL), is to apply a mapping function $f : v_i \longmapsto y_i \in \mathbb{R}^d$, for each $v \in V$. Consequently, each node is mapped to a low dimensional vector of size $d \ll |V|$, where similarity strength between nodes is tried to be preserved, so that nodes that were similar in the previous representation space, are similar in the new representation space also.

Chapter 2

Related work

Network representation learning is a continuously growing area, where rich literature is available in order to study it. [2], [3], [4], [5], [6] provide great surveys regarding the field, and constitute a great entry-point. Based on these surveys, this section provides a taxonomy concerning the methods that have been developed in regard to the field, as well a number of important NRL methods existing in the literature.

Outline. As for the outline of this chapter, section 2.1 presents the first-level taxonomy (information-based) applied in NRL methods that regard which type of information the methods are utilizing. Next, section 2.2 discuss the second-level taxonomy (algorithmic-based) concerning the methods that regard the algorithmic approach that they apply. Finally, sections 2.3 and 2.4 present in a sequence the network representation methods existing in the literature, based on the taxonomy provided in section 2.1.

2.1 Information based Taxonomy

At a firs-level, the two categories in which the methods can be categorized are; *unsupervised network representation learning* and *semi-supervised network representation learning*. The difference between the two categories lies in the fact that labels are provided for the network nodes in the former case, in order to learn the new representations, where in the latter case there are not. Consequently, in the semi-supervised network representation learning case, extra information (labels) is utilized to learn the embeddings. Figure 2.1

shows the aforementioned taxonomy and table 2.1 shows the NRL methods presented and analyzed in the chapter.

Next, the two categories, unsupervised network representation learning and semi-supervised network representation learning, are analyzed in 2.1.1 and 2.1.2 correspondingly.

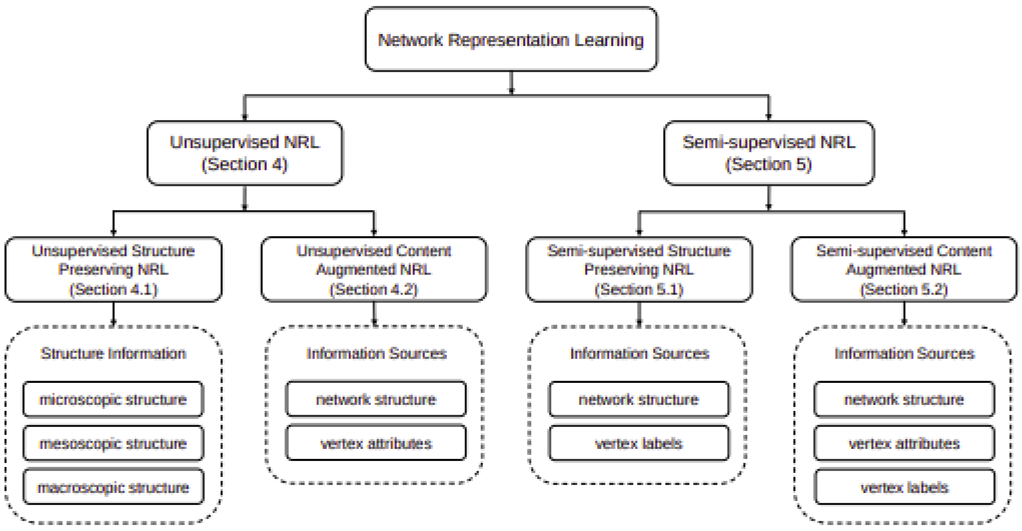

Network Representation Learning
Unsupervised NRL (Section 4)
Semi-supervised NRL (Section 5)
Unsupervised Structure Preserving NRL (Section 4.1)
Unsupervised Content Augmented NRL (Section 4.2)
Semi-supervised Structure Preserving NRL (Section 5.1)
Semi-supervised Content Augmented NRL (Section 5.2)
Structure Information
microscopic structure
mesoscopic structure
macroscopic structure
Information Sources
network structure
vertex attributes
Information Sources
network structure
vertex labels
Information Sources
network structure
vertex attributes
vertex labels

FIGURE 2.1: The taxonomy that summarizes network representation learning methods. Network representation learning is categorized into two groups, unsupervised network representation learning and semi-supervised network representation learning, depending on whether node labels are available for learning. For each group, each methods is categorized further into two subgroups, depending on whether the representation learning is based on network topology structure only, or augmented with information from node content [2]

2.1.1 Unsupervised Network Representation Learning

The majority of the methods that have been developed are categorized into the *unsupervised network representation learning* category, where no labels are provided for the network nodes. These type of methods can be divided even further into two groups, based on the type of information the utilize to produce the new vector representations, that can then used in any vector-based algorithm as inputs to solve various learning tasks. The first group

Category	Algorithms	Microscopic	Mesoscopic	Macroscopic	Attributes	Labels
Unsupervised	DeepWalk [7]	✓				
	LINE [8]	✓				
	node2vec [9]	✓				
	struct2vec [10]	✓	✓			
	GraphWave [11]		✓			
	DP [12]	✓		✓		
	HARP [13]	✓		✓		
	TADW [14]	✓			✓	
	HSCA [15]	✓			✓	
	PPNE [16]	✓			✓	
Semi-supervised	DDRW [17]	✓				✓
	MMDW [18]	✓				✓
	DMF [19]	✓				
	TriDNR [20]	✓			✓	✓

TABLE 2.1: A summary of NRL methods according to the information sources they use for learning

utilizes only network structure information (*unsupervised structure preserving NRL*), where the second group utilizes both network structure and node attributes (content) information to produce joint embeddings (*unsupervised content augmented NRL*).

2.1.2 Semi-supervised Network Representation Learning

In this case of *semi-supervised network representation learning*, some network nodes are associated with labels, which are playing a crucial role in categorizing and finding the similarity of nodes with high structural and content similarities. Techniques classified in this category, can be categorized even further with the same logic applied in the unsupervised network representation learning case; *semi-supervised structure preserving NRL* and *semi-supervised content augmented NRL*. These type of methods are linking with supervised learning tasks (e.g node classification).

2.2 Algorithmic based Taxonomy

The type of taxonomy discussed in 2.1, concerns what kind of information the NRL methods are utilizing. On the other hand, a second-level categorization derives from the algorithmic approach a method uses to produce the new network representation space; *matrix factorization*, *random walk*, *edge modeling*, *deep learning* and *hybrid* based methods.

Next, the five aforementioned algorithmic approaches are analyzed in 2.2.1, 2.2.2, 2.2.3, 2.2.4 and 2.2.5 correspondingly.

2.2.1 Matrix Factorization Methods

Most early approaches in NRL used matrix-factorization to produce the node embeddings. In matrix-factorization approaches, the connection between the nodes are represented in a matrix, which then is factorized to construct the embeddings. Some common matrices used in matrix-factorization are: node adjacency matrix, Laplacian matrix, modularity matrix, node transition probability matrix, and Katz similarity matrix. What type of factorization strategy is used, depends on the matrix properties and objective. Matrix-factorization based methods have been proven to be effective, but comes with a huge bottleneck regarding scalability since carrying out factorization in large matrices is very expensive and some times inapplicable.

2.2.2 Random Walk Methods

Random walk is an approach to capture structural relationships between nodes in a network. The intuitive idea behind random walks is that nodes that tend to co-exist in short random walks in a network, their produced embeddings should be close in the new representation space. Consequently, performing a number of random walks maps the original network to a collection of node sequences from which the structure similarity between nodes is calculated.

2.2.3 Edge Modeling Methods

Methods that apply edge modeling logic, use the node to node connections in a network to learn the new representation space. In contrast to random walk and matrix-factorization methods, edge modeling methods do not utilize global network structure information.

2.2.4 Deep Learning Methods

Deep learning is a field that has been showed that can be applied in a variety of fields, where one of them is NRL. The advantaged of deep learning methods derives from the fact that they can capture non-linearity in networks, that comes with a high computational cost though. Either entirely new deep learning models have been constructed or models applied in other fields have been adopted in NRL.

2.2.5 Hybrid Methods

In some cases, methods have been developed that combine the above methods to produce the new representation space of a network.

2.3 Unsupervised Network Representation Learning Methods

This section presents NRL methods of the literature that are categorized in the unsupervised network representation learning category (2.1.1). Methods that belong to the unsupervised structure preserving NRL category (2.3.1) are presented at first, and then those that belong to the unsupervised content augmented NRL category (2.3.2).

2.3.1 Unsupervised Structure Preserving NRL Methods

As showed in Figure 2.1, unsupervised structure preserving NRL category is split to three types; *microscopic structure*, *mesoscopic structure* and *macroscopic structure*.

Microscopic structure. Methods that utilize local structure information to produce the new representation space through first-order (1.3.3), second-order (1.3.4) or high-order (1.3.5) proximity are capturing microscopic structure.

Mesoscopic structure. Methods that utilize structural (1.3.6) proximity to produce the new representation space are capturing mesoscopic structure.

Macroscopic structure. Methods that capture macroscopic structure utilize global network properties like small world or scale-free property.

Next, the methods that are categorized in the unsupervised structure preserving NRL category are presented, where those that use microscopic structure are analyzed at first, then the ones that use mesoscopic structure, and finally the ones that use macroscopic structure.

DeepWalk. DeepWalk [7] extends the Skip-Gram model [21] in order to produce the embeddings. It is a random walk based method, that preserves second-order and high-order proximity, so nodes that have similar neighborhood structure in the original network are close in the new representation space. DeepWalk generates a set of random walks sequences of length L to learn the representation of a node, by using the sequences to predict its context nodes. In order to accomplish the aforementioned, DeepWalk solves the below optimization problem.

$$\min -\log \Pr(\{v_{i-t}, \dots, v_{i+t}\} \setminus v_i | f(v_i))$$

(2.1)

Large-scale Information Network Embedding (LINE). LINE [8] learns the node embeddings by modeling directly the first-order and second-order proximity. Equation (2.2) shows the objective that LINE minimize for the first-order proximity, where equation (2.3) shows the objective that LINE minimize for the second-order proximity. LINE concatenates the two produced latent embeddings from the two proximities in order to create the final representation space.

$$O_1 = d(\hat{p}_1(\cdot,\cdot), p_1(\cdot,\cdot)). \tag{2.2}$$

$$O_2 = \sum_{v_i \in V} \lambda_i d(\hat{p}_2(\cdot|v_i), p_2(\cdot|v_i)). \tag{2.3}$$

As for equation (2.2), $d(\cdot,\cdot)$ is the distance between two distributions. Then, $\hat{p_1}(\cdot,\cdot)$ is the empirical distribution between two nodes that are connected with an edge, and is $p_1(\cdot,\cdot)$ the joint distribution modeled by their latent embeddings.

As for equation (2.3), λ_i is the prestige of node v_i. Then, $p_2(\cdot|v_i)$ is the context conditional distribution for a node modeled by node embeddings, and $\hat{p_2}(\cdot|v_i)$ is the empirical conditional distribution.

node2vec. Similar to DeepWalk [7], node2vec [9] applies the random walk model to capture second-order and high-order proximity. The difference of node2vec is that it uses biased random walks, where a trade-of between breadth-first (BFS) and depth-first (DFS) strategy is applied. Consequently, the biased random walk provides a set of neighbor nodes $N(v_i)$ that is used to learn the vector representation $f(v_i)$ of node v_i by optimizing the occurrence probability of neighbor vertices $N(v_i)$ conditioned on the representation of vertex v_i.

$$\max_f \sum_{v_i \in V} \log \Pr\left(N\left(v_i\right) \mid f\left(v_i\right)\right) \tag{2.4}$$

struct2vec. struct2vec [10] exploits a multilayer logic, where node structural role similarity is mapped to a multilayer graph. At each layer, the edge weights are determined by the structural role difference at the corresponding scale. After the construction of the multilayer graph, DeepWalk [7] is executed to learn the new representation space for each vertex pair (v_i, v_j). Considering their k-hop neighborhood formed by their neighbors within k steps, their structural distance at scale k, $D_k(v_i, v_j)$, is defined as

$$D_k\left(v_i, v_j\right) = D_{k-1}\left(v_i, v_j\right) + g\left(s\left(R_k\left(v_i\right)\right), s\left(R_k\left(v_j\right)\right)\right) \tag{2.5}$$

where $R_k(v_i)$ is the set of vertices in v_i's k-hop neighborhood, $s(R_k(v_i))$ is the ordered degree sequence of the vertices in $R_k(v_i)$, and $g(s(R_k(v_i))$, $s(R_k(vj)))$ is the distance between the ordered degree sequences $s(R_k(vi))$ and $s(R_k(vj))$. When $k = 0$, $D_0(v_i, v_j)$ is the degree difference between vertex v_i and v_j.

GraphWave. GraphWave [11] maps node neighborhood structure to latent vector by making use of the spectral graph wavelet diffusion patterns. For node v_k, its spectral graph wavelet coefficients Ψ_k is defined as equation (2.6).

$$\Psi_k = U \operatorname{Diag}\left(g_s\left(\lambda_1\right), \cdots, g_s\left(\lambda_{|V|}\right)\right) U^{\mathrm{T}} \delta_k \tag{2.6}$$

In (2.6), U is the eigenvector matrix of the graph Laplacian L and $\lambda_1, \cdots, \lambda_{|V|}$ are the eigenvalues, $g_s(\lambda) = exp(\lambda_s)$ is the heat kernel, and δ_k is the one-hot vector for k.

Degree penalty principle (DP). The scale-free property, which portrays the event that node degree follows a long-tailed distribution, is used by DP [12] to learn the new representation space. In the long-tailed distribution case, nodes are sparsely connected at most and a few are densely connected. DP proposes the degree penalty principle, that is, punishing the proximity between nodes with high-degree. In order to learn then the new representation space, DP combines the penalty principle with the DeepWalk[7] and spectral embedding algorithms [22].

Hierarchical Representation Learning for Networks (HARP). HARP [13] aims to capture global patterns in a network, by sampling smaller networks from the original network. HARP samples the smaller networks sequentially, and learn latent embeddings for each, and the uses these individual embeddings to infer the global representation. In order to learn the embeddings, HARP utilizes DeepWalk [7] and LINE [8].

2.3.2 Unsupervised Content Augmented NRL Methods

In case where network nodes are attached with attributes, network representation learning methods can use this content to achieve more efficient results, than just utilizing network structure. This field has been not researched as much as the field concerning non-attributed networks, so they aren't as many methods in this category in the literature as in the unsupervised structure preserving category. Next, methods that belong in the unsupervised content augmented category are presented.

Text-Associated DeepWalk (TADW). TADW [14] proves that DeepWalk [7] and the matrix factorization presented in equation (2.7) are equal, where W and H are learned latent embeddings and M is the vertex-context matrix carrying transition probability between each vertex pair within k steps.

$$\min_{W,H} \left\| M - W^{\mathrm{T}} H \right\|_F^2 + \frac{\lambda}{2} \left(\|W\|_F^2 + \|H\|_F^2 \right) \tag{2.7}$$

Next, textual features are imported through inductive matrix factorization [23] through equation 2.8, where T is vertex textual feature matrix. The final latent node representations are produced by concatenating W and HT.

$$\min_{W,H} \left\| M - W^{\mathrm{T}} HT \right\|_F^2 + \frac{\lambda}{2} \left(\|W\|_F^2 + \|H\|_F^2 \right) \tag{2.8}$$

Homophily, Structure, and Content Augmented Network Representation Learning (HSCA). HSCA [15] utilizes homophily (first-order proximity), structural context (second-order and high-order proximity), and vertex content (attributes) to produce the new representation space, by extending TADW [14] that does not utilizes first-order proximity. To insert the first-order proximity, HSCA applies a regularization term presented in equation (2.9), where S is the adjacent matrix.

$$\mathcal{R}(W,H) = \frac{1}{4} \sum_{i,j=1}^{|V|} S_{ij} \left\| \begin{bmatrix} W_{:i} \\ HT_{:i} \end{bmatrix} - \begin{bmatrix} W_{:j} \\ HT_{:j} \end{bmatrix} \right\|_2^2 \tag{2.9}$$

Then, the objective function of HSCA is equation 2.10, where λ and μ are the trade-off parameters.

$$\min_{W,H} \left\| M - W^{1} HT \right\|_F^2 + \frac{1}{2} \left(\|W\|_F^2 + \|H\|_F^2 \right) + \mu \mathcal{R}(W,H) \tag{2.10}$$

After solving the above optimization problem, the final latent node representations are produced by concatenating W and HT

Property Preserving Network Embedding (PPNE). To produce the latent representation, PPNE [16] optimizes a structure-based objective function and an attributed-based objective function. For the former, PPNE tries to make nodes close with similar structure, where given random walk sequence S, equation (2.11) constitutes the structure-based objective function.

$$\min D_T = \prod_{v \in \mathcal{S}} \prod_{u \in context(v)} \Pr(u \mid v) \tag{2.11}$$

Then, attribute-based objective function tries to make the node representations learned by the structure-based objective function to take under consideration the node attribute similarity. Equation (2.12) constitutes the attribute-based objective function.

$$\min D_N = \sum_{v \in S} \sum_{u \in \mathrm{pos}(v) \cup \mathrm{neg}(v)} P(v,u) d(v,u) \tag{2.12}$$

In (2.12), $d(u,v)$ is the distance between u and v in the new representation space, $P(u,v)$ is the attribute similarity between u and v, and $\mathrm{pos}(v)$ and $\mathrm{neg}(v)$ are the set of top-k similar and dissimilar vertices respectively, according to $P(u,v)$.

2.4 Semi-Supervised Network Representation Learning Methods

This section presents NRL methods of the literature that are categorized in the semi-supervised network representation learning category (2.1.2). At first, methods that belong to the semi-supervised structure preserving NRL category (2.4.1) are presented, and then those that belong to the semi-supervised content augmented NRL category (2.4.2)

2.4.1 Semi-supervised Structure Preserving NRL Methods

In this group of methods (semi-supervised structure preserving NRL), two methods are analyzed that aim to learn the new representations space with the help of node labels, capturing network structure and discriminative learning. Next, the two methods are analyzed.

Discriminative Deep Random Walk (DDRW). DDRW [17] learns discriminative node embeddings [24], [25] by optimizing the objective of DeepWalk [7] together with the following L2-loss Support Vector Classification objective of equation (2.13). In (2.13), $\sigma(x) = x$, if $x > 0$ and otherwise $\sigma(x) = 0$.

$$\mathcal{L}_c = C\sum_{i=1}^{|V|}\left(\sigma\left(1 - Y_{ik}\beta^{\mathrm{T}} r_{v_i}\right)\right)^2 + \frac{1}{2}\beta^{\mathrm{T}}\beta \tag{2.13}$$

Consequently, the final objective function of DDRW is (2.14), where $\mathcal{L}_{DW}$ is the objective function of DeepWalk.

$$\mathcal{L} = \eta\mathcal{L}_{DW} + \mathcal{L}_c \tag{2.14}$$

Max-Margin DeepWalk (MMDW). Based on the same logic as DDRW, MMDW [18] uses the objective of DeepWalk [7] with the multiclass Support Vector machine objective presented in equation (2.15), with $\{(r_{v1}, Y_1 :), \ldots, (r_{vT}, Y_T :)\}$ training set.

$$\begin{aligned} &\min_{W,\xi} \mathcal{L}_{SVM} = \min_{W,\xi} \frac{1}{2}\|W\|_2^2 + C\sum_{i=1}^{T}\xi_i, \\ &\text{s.t. } \mathrm{w}_{l_i}^{\mathrm{T}} r_{v_i} - \mathrm{w}_j^{\mathrm{T}} r_{v_i} \geq e_i^j - \xi_i, \forall i, j, \end{aligned} \tag{2.15}$$

In (2.15), $l_i = k$ with $Y_{ik} = 1$, $e_i^j = 1$ for $Y_{ij} = -1$, and $e_i^j = 0$ for $Y_{ij} = 1$. Consequently, the final objective function of MMDW is (2.16), where $\mathcal{L}_{DW}$ is the objective function of DeepWalk.

$$\begin{aligned} &\min_{U,H,W,\xi} \mathcal{L} = \min_{U,H,W,\xi} \mathcal{L}_{DW} + \frac{1}{2}\|W\|_2^2 + C\sum_{i=1}^{T}\xi_i, \\ &\text{s.t. } \mathrm{w}_{l_i}^{\mathrm{T}} r_{v_i} - \mathrm{w}_j^{\mathrm{T}} r_{v_i} \geq e_i^j - \xi_i, \forall i, j. \end{aligned} \tag{2.16}$$

2.4.2 Semi-supervised Content Augmented NRL Methods

Methods that utilize node attributes and labels have been developed too, in the semi-supervised network representation learning category In this case, the learned latent representation are expected to be more informative. Two methods belonging to the semi-supervised content augmented NRL category are presented next.

Discriminative Matrix Factorization (DMF). DMF [19] enforces the objective of TADW (2.3.2) with an empirical loss minimization for a linear classifier trained on labeled nodes, presented in equation (2.17). In (2.17), wi is the i-th column of vertex representation matrix W and t_j is j-th column of vertex textual feature matrix T, and L is the set of indices of labeled vertices. DMF considers binary-class classification, i.e. $Y = \{+1, 1\}$. Hence, Y_{n1} is used to denote the class label of vertex v_n.

$$\begin{aligned} \min_{W,H,\eta} \frac{1}{2} \sum_{i,j=1}^{|V|} \left(M_{ij} - \boldsymbol{w}_i^{\mathrm{T}} H \boldsymbol{t}_j \right)^2 + \frac{\mu}{2} \sum_{n \in \mathcal{L}} \left(Y_{n1} - \boldsymbol{\eta}^{\mathrm{T}} \boldsymbol{x}_n \right)^2 \\ + \frac{\lambda_1}{2} \left(\|H\|_F^2 + \|\boldsymbol{\eta}\|_2^2 \right) + \frac{\lambda_2}{2} \|W\|_F^2 \end{aligned} \tag{2.17}$$

The optimization problem (2.17) is solved by optimizing W, H and $\boldsymbol{\eta}$ alternately. When the optimization function is resolved, the discriminative and informative node embeddings together with the classifier are capable of classifying unlabeled nodes in the network.

Tri-Party Deep Network Representation (TriDNR). TriDNR [20] gets information from network structure, node content (attributes) and node labels, in order to learn the new representations space by utilizing a coupled neural network framework. A Paragraph Vector model [26] is used to analyze the vertex-word correlation and the label-word correspondence by maximizing the objective function presented in equation (2.18).

$$\mathcal{L}_{PV} = \sum_{i \in L} \log \Pr \left(w_{-b} : w_b \mid c_i \right) + \sum_{i=1}^{|V|} \log \Pr \left(w_{-b} : w_b \mid v_i \right) \tag{2.18}$$

In (2.18), $\{w_{-b} : w_b\}$ is a sequence of words inside a contextual window of length $2b$, c_i is the class label of node v_i, and L is the set of indices of labeled nodes. Finally, the Paragraph Vector objective is combined with DeepWalk [7] objective, leading to equation (2.19), where $\mathcal{L}_{DW}$ is the objective function of DeepWalk and $\boldsymbol{\alpha}$ is the trade-off parameter.

$$\max(1-\alpha)\mathcal{L}_{DW} + \alpha\mathcal{L}_{PV} \tag{2.19}$$

Chapter 3

The RandNE Algorithm

As mentioned in section 1.1, *scalability*, that is developing methods that are applicable in enormous networks, is one of the most difficult challenges in network representation learning. The vast majority of the methods in the literature are unfeasible to be executed in large networks, due to time and space complexity.

In 2018, Zhang and his colleagues introduced RandNE (Iterative Random Projection Network Embedding) [1]. The main objective of this thesis is to provide a distributed version of the RandNE method, which could be utilized to produce node embeddings for large network. As [1] mentions, RandNE is ”a novel and simple billion-scale network embedding method based on high-order proximity preserved random projection”. Random projection is an efficient technique to learn low-dimensional vector-based representation of the network nodes, with objective to capture the structure of the original network. One advantage of random projection is that it can also be utilized in a distribute manner, consequently it can be used in large-scale data cases [27].

In order to to minimize the matrix factorization objective function that captures the high-order proximity, RandNE uses the Gaussian random projection. It executes an iterative projection procedure that enables to capture arbitrary high-order proximity preserved random projection with a linear time complexity. In that manner, RandNE avoids to calculate explicitly high-order proximities that is highly computational expensive.

In summary, the contributions of [1] are presented below.

- A novel and simple random projection based network representation learning method is introduced, named RandNE, that allows billion-scale network representation learning.

- RandNE provides an iterative projection procedure to realize high-order proximities preserved random projection efficiently without explicitly calculating the high-order proximities.

- RandNE can be developed in a distributed manner (3.2.4), without communication cost and can also efficiently deal with dynamic networks (3.2.5) without error aggregation. These two aforementioned facts, are theoretically and empirically proved in [1].

Outline. As for the outline of this chapter, section 3.1 presents the notations used in [1] and the problem formulation. Then section 3.2 presents the analyses of RandNE and its important individual characteristics. Finally, section 3.3 presents the results of RandNE presented in [1].

3.1 Notation And Problem Formulation

Since an introduction of RandNE has been presented above, this section presents the notations used throughout [1] and the analyzes of the problem formulation. At first, (3.1.1) presents the notations, whose comprehension is important in order to understand the problem formulation and the analyzes of the proposed method (RandNE) presented at (3.1.2) and (3.2) correspondingly.

3.1.1 Notations

The adjacency matrix of a network G with N nodes and M edges, is donated as $\mathbf{A}$. RandNE as presented in [1], concerns undirected networks, so $\mathbf{A}$ is symmetric, and $\mathbf{A}^T$ donates its transpose. $\mathbf{A}(i, :)$ represents the i^{th} row of $\mathbf{A}$, where $\mathbf{A}(:, j)$ represents the j^{th} column of $\mathbf{A}$. In general, matrices are denoted with bold uppercase characters (e.g. $\mathbf{X}$), and vectors with bold lowercase characters (e.g. $\mathbf{x}$). As for the product of two matrices, a dot notation is used (e.g. $\mathbf{X} \cdot \mathbf{Y}$), and functions are marked by curlicue (e.g. $\mathcal{F}(\cdot)$).

3.1.2 Problem Formulation

One of the most commonly used techniques, regarding the learning process of low-dimensional node embeddings, is matrix-factorization. In RandNE method, a similarity function of the adjacency matrix $\mathcal{F}(A) \in \mathbb{R}^{N \times N}$ is broke down to the product of two low-dimensional matrices $\mathbf{U}$ and $\mathbf{V} \in \mathbb{R}^{N \times d}$, with the objective function presented in equation (3.1), where p is the norm of the embedding.

$$\min_{\mathbf{U},\mathbf{V}} \left\| \mathcal{F}(\mathbf{A}) - \mathbf{U} \cdot \mathbf{V}^T \right\|_p \tag{3.1}$$

[1] focuses only on the spectral norm (i.e $p = 2$) [28], and since it takes under consideration only undirected networks, $\mathbf{U} = \mathbf{V}$. Furthermore, [1] estimates that $\mathcal{F}(A)$ is a positive semi-definite function, so it can be formulated $\mathcal{F}(A) = \mathbf{S} \cdot \mathbf{S}^T$. Consequently, equation (3.1) can be modified as:

$$\begin{gathered} \min_{\mathbf{U}} \left\| \mathbf{S} \cdot \mathbf{S}^T - \mathbf{U} \cdot \mathbf{U}^T \right\|_2 \\ \mathbf{S} = \alpha_0 \mathbf{I} + \alpha_1 \mathbf{A} + \alpha_2 \mathbf{A}^2 + \ldots + \alpha_q \mathbf{A}^q \end{gathered} \tag{3.2}$$

In (3.2), $\mathbf{S}$ is the high-order proximity matrix, q is the order and $a_0, a_1, \ldots, a_q$ are predefined weights.

3.2 Proposed Method (RandNE) Analyzes

Now that the notations used throughout [1] and the problem formulation have been presented in section 3.1, this section analyzes RandNE and its important individual characteristics in detail.

At first, 3.2.1 presents how Guassian random projection is utilized in the problem formulation to minimize the objective function. Next, 3.2.2 shows how a time-wise issue that arises from using Guassian random projection is handled, and presents finally the pseudocode of RandNE. Then 3.2.3 presents the time-complexity analyses of RandNE. At last, 3.2.4 and 3.2.5 analyzes how RandNE can be executed in a distributed environment and in a dynamic network, correspondingly.

3.2.1 Gaussian Random Projection Embedding

RandNE uses random projection, more specifically Guassian random projection [27], in order to minimize the objective function of equation (3.2). So, let assume that $\mathbf{R} \in \mathbb{R}^{N \times d}$ and each element of $\mathbf{R}$ follows an i.i.d Gaussian distribution $\mathbf{R}(i,j) \sim \mathcal{N}\left(0, \frac{1}{d}\right)$, then the embeddings $\mathbf{U}$ can be obtained by performing a matrix product, presented in equation (3.3).

$$\mathbf{U} = \mathbf{S} \cdot \mathbf{R} = \left(\alpha_0 \mathbf{I} + \alpha_1 \mathbf{A} + \alpha_2 \mathbf{A}^2 + \ldots + \alpha_q \mathbf{A}^q\right) \mathbf{R} \tag{3.3}$$

What equation (3.2) accomplishes, is that it randomly projects the proximity matrix $\mathbf{S}$ into a low-dimensional subspace. In order the aforementioned to be true, the following theorem is true about Guassian random projection that provides a theoretical guarantee.

Theorem 1. For any similarity matrix $\mathbf{S}$, denote its rank as r_S. Then, for any $\epsilon \in \left(0, \frac{1}{2}\right)$, the following equation hold:

$$P\left[\left\|\mathbf{S} \cdot \mathbf{S}^T - \mathbf{U} \cdot \mathbf{U}^T\right\|_2 > \epsilon \left\|\mathbf{S}^T \cdot \mathbf{S}\right\|_2\right] \leq 2rse^{-\frac{\left(c^2 - \epsilon^3\right)d}{4}} \tag{3.4}$$

where $\mathbf{U} = \mathbf{S} \cdot \mathbf{R}$ and R is a Gaussian random matrix.

What the theorem shows is that the residual of our projection $\mathbf{S} \cdot \mathbf{S}^T - \mathbf{U} \cdot \mathbf{U}^T$ has a much smaller spectral radius than the spectral radius of the original high-order proximity $\mathbf{S} \cdot \mathbf{S}^T$. Consequently, Gaussian random projection can be applied to the objective function presented in equation (3.2).

3.2.2 Iterative Projection

Despite the applicability of Gaussian random projection, in case where S is dense, executing the projection is still time expensive and arises problems about the scalability issue in large networks.

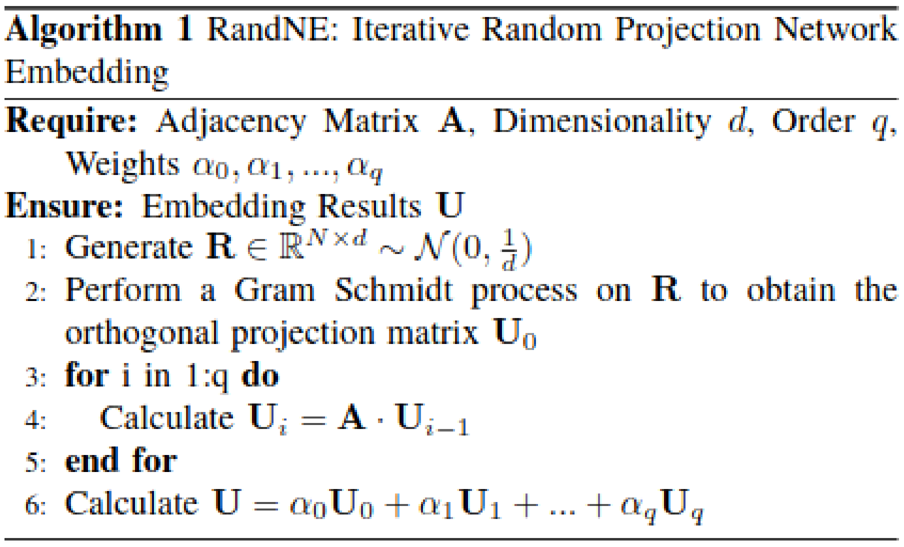
To handle this problem, [1] proposes an iterative projection procedure. In equation (3.3), matrix $\mathbf{U}$ is decomposed into matrices of different orders, as showed in equation (3.5), where $\mathbf{U}_i = \mathbf{A}^i \cdot \mathbf{R}, 0 \leq i \leq q$.

$$\mathbf{U} = \alpha_0 \mathbf{U}_0 + \alpha_1 \mathbf{U}_1 + \ldots + \alpha_q \mathbf{U}_q \tag{3.5}$$

As for the $\mathbf{U}_1 \ldots \mathbf{U}_q$, they can be calculated recursively, using equation (3.6).

$$\mathbf{U}_i = \mathbf{A} \cdot \mathbf{U}_{i-1}, \forall 1 \leq i \leq q \tag{3.6}$$

Based on (3.6), only the dot product of a sparse adjacency matrix and low-dimensional matrix is needed to be calculated, which using sparse matrix can be highly scalable. So, figure 3.1 presents pseudocode implementation of the RandNE method.



Algorithm 1 RandNE: Iterative Random Projection Network Embedding

```
Require: Adjacency Matrix $\mathbf{A}$, Dimensionality $d$, Order $q$,
    Weights $\alpha_0, \alpha_1, ..., \alpha_q$
Ensure: Embedding Results $\mathbf{U}$
1: Generate $\mathbf{R} \in \mathbb{R}^{N \times d} \sim \mathcal{N}(0, \frac{1}{d})$
2: Perform a Gram Schmidt process on $\mathbf{R}$ to obtain the
   orthogonal projection matrix $\mathbf{U}_0$
3: for i in 1:q do
4:     Calculate $\mathbf{U}_i = \mathbf{A} \cdot \mathbf{U}_{i-1}$
5: end for
6: Calculate $\mathbf{U} = \alpha_0 \mathbf{U}_0 + \alpha_1 \mathbf{U}_1 + ... + \alpha_q \mathbf{U}_q$
```

FIGURE 3.1: The pseudocode implementation of the RandNE algorithm [1]

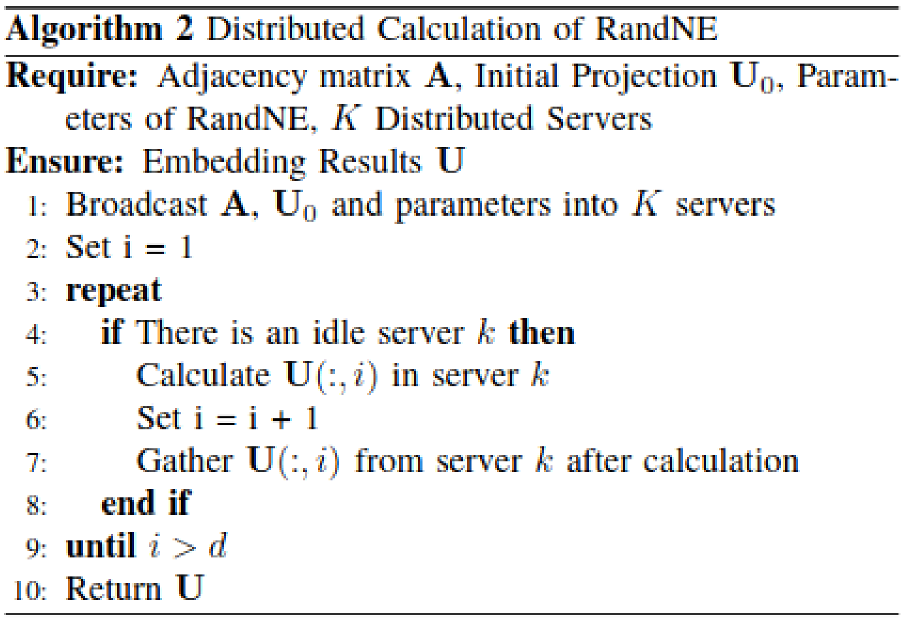
3.2.3 Time complexity

RandNE time complexity is $O\left(N \cdot d^2 + M \cdot q \cdot d\right)$. It is linear with respect to the network size. This time complexity derives from the analyses of the pseudocode in figure 3.1. The complexity of line 1 and 2 are $O(N \cdot d)$ and $O(N \cdot d^2)$ correspondingly. Then, the complexity of line 3 to 5 for each iteration is $O(M \cdot d)$ and for line 6 it is $O(q \cdot N \cdot d)$. In the above analysis, N is the number of nodes in the network, M the number of edges in the network, q the proximity order and d the dimensionality of the new representation space.

RandNE is extremely efficient because it executes only q iterations, and in each iteration, just a simple matrix product is calculated.

3.2.4 Distributed Computing

Because matrix product and matrix sum calculations are a well studied problems in distributed computing and have been solved, RandNE can be executed in a distributed manner, due to the fact that the lines 3 to 6 in figure 3.1 can be executed in parallel. Figure 3.2 shows a protocol that can be utilized to implement RandNE in a distributed manner.



```
Algorithm 2 Distributed Calculation of RandNE
Require: Adjacency matrix $\mathbf{A}$, Initial Projection $\mathbf{U}_0$, Parameters of RandNE, $K$ Distributed Servers
Ensure: Embedding Results $\mathbf{U}$
 1: Broadcast $\mathbf{A}$, $\mathbf{U}_0$ and parameters into $K$ servers
 2: Set i = 1
 3: repeat
 4:     if There is an idle server $k$ then
 5:         Calculate $\mathbf{U}(:, i)$ in server $k$
 6:         Set i = i + 1
 7:         Gather $\mathbf{U}(:, i)$ from server $k$ after calculation
 8:     end if
 9: until $i > d$
10: Return $\mathbf{U}$
```

FIGURE 3.2: The pseudocode of the distributed framework implementation of the RandNE algorithm [1]

Other methods in the NRL literature can be implement utilizing multiple server too, but RandNE separates itself from the other methods because of the low-communication cost between the servers that is needed, fact that makes it a very promising method for large networks.

3.2.5 Dynamic Networks

RandNE has also the characteristic that it can be applied for dynamic networks. A dynamic network can have three types of changes. **1)** changes of edges **2)** nodes added **3)** nodes deleted.

For **1)**, only the corresponding $\mathbf{U}_i, 1 \leq i \leq q$ are needed to be updated in order to update the node embeddings. Donating the changes in the adjacency matrix as ΔA and the changes in $\mathbf{U}_i$ as $\Delta \mathbf{U}_i, 1 \leq i \leq q$, based on equation (3.6), we can recursively calculate $\Delta \mathbf{U}_i$ with equation (3.7).

$$\begin{aligned} \mathbf{U}_i + \Delta \mathbf{U}_i &= (\mathbf{A} + \Delta \mathbf{A}) \cdot (\mathbf{U}_{i-1} + \Delta \mathbf{U}_{i-1}) \\ \Rightarrow \Delta \mathbf{U}_i &= \mathbf{A} \cdot \Delta \mathbf{U}_{i-1} + \Delta \mathbf{A} \cdot \mathbf{U}_{i-1} + \Delta \mathbf{A} \cdot \Delta \mathbf{U}_{i-1} \end{aligned} \tag{3.7}$$

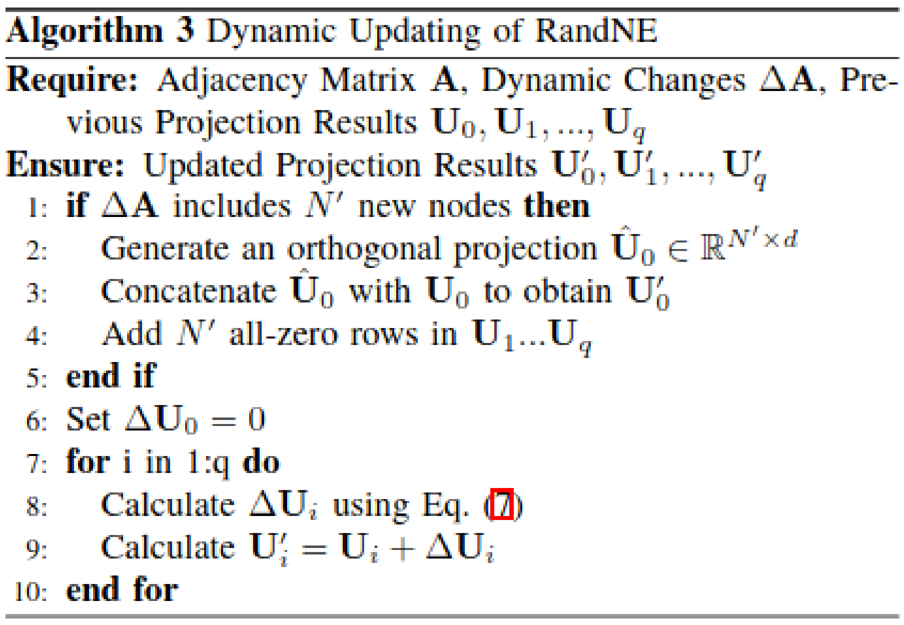
As for **2)**, the same logic applies as **1)**, where the edges of deleted nodes are just deleted.

In case **3)**, at first, the corresponding nodes are added without attached edges, then an additional orthogonal Gaussian random matrix $\hat{\mathbf{U}}_0 \in \mathbb{R}^{N' \times d}$, where N' the number of nodes added, is concatenated with the current projection matrix $\mathbf{U}_0$ to form the new projection matrix $\mathbf{U}_0'$. Then, the other $\mathbf{U}_i, 1 \leq i \leq q$ have all-zero elements for the nodes added, so N' all-zero rows are concatenated to them to match the dimensionality. Finally, edges attached to the added nodes can be added using equation (3.7). Figure 3.3 shows the framework regarding the dynamic updating for RandNE.

3.3 Experimentation Results

In order to evaluate the efficiency of RandNE, Zhang and his colleagues conducted experiments on the datasets presented in table 3.1. In [1], RandNE is evaluated in regard to its time efficiency, as well its efficiency in network analytic tasks, in comparison to some baseline methods; DeepWalk, Line, node2vec, SDNE. All the results concerning RandNE's time efficiency and its efficiency in regard to various network analytic tasks are presented below.

Time efficiency. Figure 3.4 shows the running time results of all the methods. As figure 3.4 shows, RandNE is more efficient by more than 24 times over all the baselines methods, on all networks.



```
Algorithm 3 Dynamic Updating of RandNE
Require: Adjacency Matrix $\mathbf{A}$, Dynamic Changes $\Delta\mathbf{A}$, Previous Projection Results $\mathbf{U}_0, \mathbf{U}_1, ..., \mathbf{U}_q$
Ensure: Updated Projection Results $\mathbf{U}'_0, \mathbf{U}'_1, ..., \mathbf{U}'_q$
 1: if $\Delta\mathbf{A}$ includes $N'$ new nodes then
 2:     Generate an orthogonal projection $\hat{\mathbf{U}}_0 \in \mathbb{R}^{N' \times d}$
 3:     Concatenate $\hat{\mathbf{U}}_0$ with $\mathbf{U}_0$ to obtain $\mathbf{U}'_0$
 4:     Add $N'$ all-zero rows in $\mathbf{U}_1 ... \mathbf{U}_q$
 5: end if
 6: Set $\Delta\mathbf{U}_0 = 0$
 7: for i in 1:q do
 8:     Calculate $\Delta\mathbf{U}_i$ using Eq. (7)
 9:     Calculate $\mathbf{U}'_i = \mathbf{U}_i + \Delta\mathbf{U}_i$
10: end for
```

FIGURE 3.3: The pseudocode of the dynamic framework implementation of the RandNE algorithm [1]

Dataset	Nodes	Edges
BlogCatalog	10.312	667.966
Flickr	80.513	11.799.764
Youtube	1.138.499	5.980.886

TABLE 3.1: Summary of datasets that were used to evaluate the RandNE method [1]

Network reconstruction. In [1], RandNE is examined in regard to its efficiency to reconstruct networks, by ranking pairs of nodes according to their inner product similarities. For the evaluation metrics, Area Under Curve (AUC) [29] and Precision@K [30] are utilized.

Figures 3.5 and 3.6 present the correspondingly results. On AUC, RandNE achieves the best performance on BlogCatalog and Flickr. On the other hand, RandNE has comparable performance on Youtube, compared to the other baseline methods. As for the the Precision@K metric, RandNE outperforms all the baseline methods.

Link prediction. In order to evaluate RandNE's efficiency in link prediction, the 30% of the edges were randomly hidden for testing. RandNE's

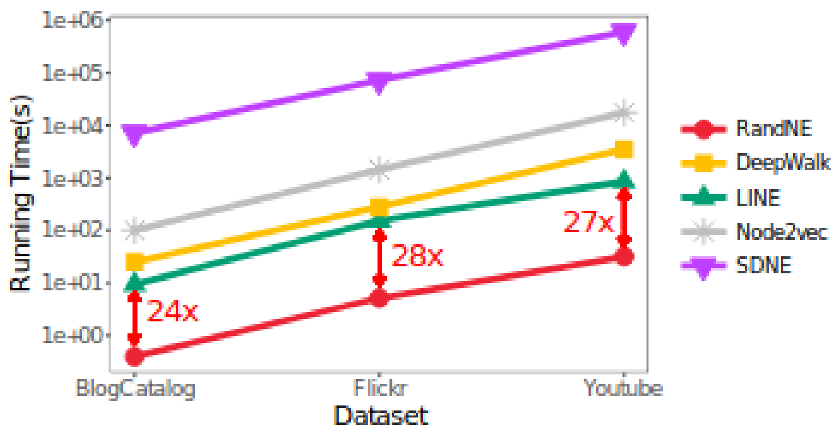
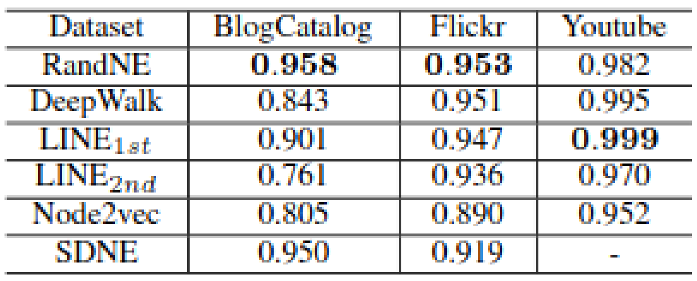


FIGURE 3.4: The running time comparison of RandNE to other methods [1]



Dataset	BlogCatalog	Flickr	Youtube
RandNE	**0.958**	**0.953**	0.982
DeepWalk	0.843	0.951	0.995
LINE$_{1st}$	0.901	0.947	**0.999**
LINE$_{2nd}$	0.761	0.936	0.970
Node2vec	0.805	0.890	0.952
SDNE	0.950	0.919	-

FIGURE 3.5: AUC scores of network reconstruction[1]

efficiency to link prediction is examined the same way as network reconstruction.

Figures 3.7 and 3.8 present the correspondingly results. Except the AUC score on Youtube, RandNE outperforms the all other methods.

Node classification. Regarding RandNE's examination on the node classification task, an one-vs-all logistic regression with L2 regularization [31] is trained using the embeddings on the training set, and tested on the testing set. Training and testing set were randomly split. Macro-F1 and Micro-F1 were used to evaluate the performance.

Different results are produced on different network in regard to the methods efficiency, as showed in figure 3.9. RandNE achieves the best result on Flickr. On the other networks, RandNE has comparable results with the other methods.

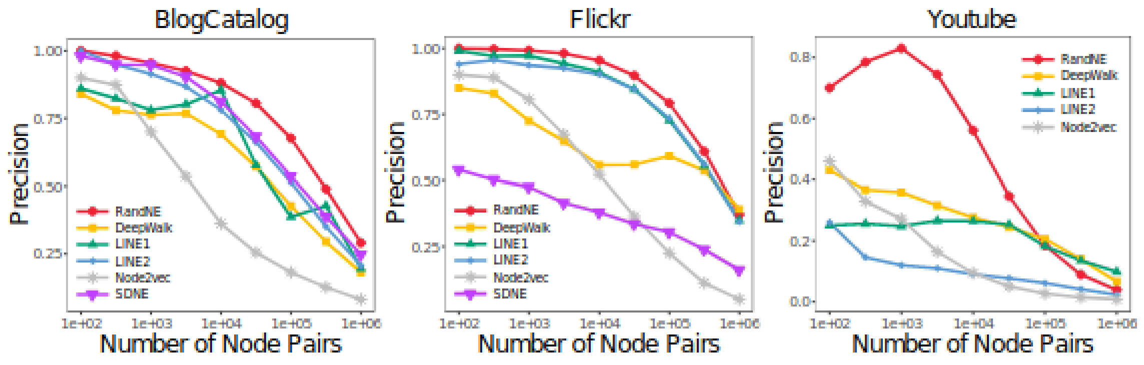


FIGURE 3.6: The Precision@K of network reconstruction for the RandNE method compared to other methods, on moderate-scale networks [1]

Dataset	BlogCatalog	Flickr	Youtube
RandNE	**0.944**	**0.940**	0.887
DeepWalk	0.760	0.938	0.909
LINE$_{1st}$	0.667	0.909	0.847
LINE$_{2nd}$	0.762	0.932	**0.959**
Node2vec	0.650	0.865	0.778
SDNE	0.940	0.926	-

FIGURE 3.7: AUC scores of link prediction [1]

Structural role classification. Capturing the structural role of nodes in network embedding has important applications such as influence maximization and measuring node centrality. For this case, different datasets were utilized from those presented in table 3.1; three air-traffic networks[1] from Brazilian, European and American. The networks have 131 nodes and 2.006 edges, 399 nodes and 11.986 edges, 1.190 nodes and 27.198 edges respectively.

Except from the performance metric utilized, which in this case is accuracy, the experimental setup is similar as the node classification case. In all network cases, the embedding sizes are 16. From the results presented in figure 3.10, we see that RandNE outperforms all the other methods on the European Flights networks, and comes second in the other two network cases.

Impact of proximity order (q). [1] examined the impact of the q

[1]repository that contains the three network

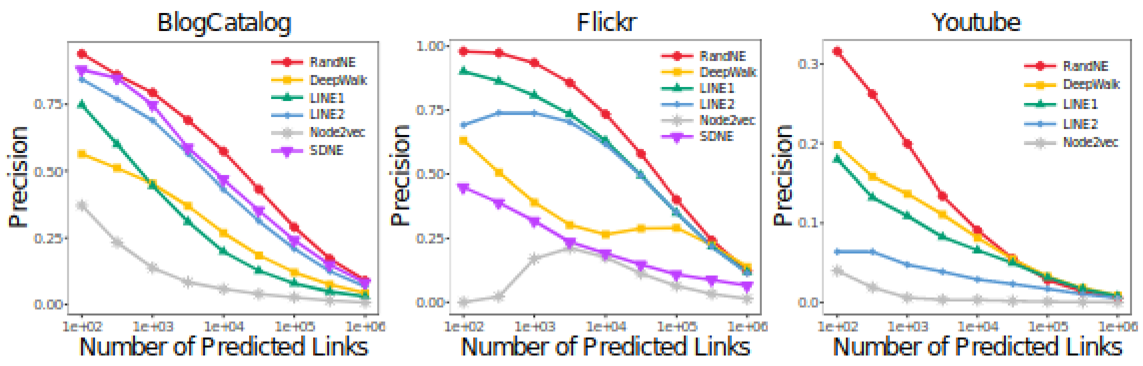


FIGURE 3.8: The Precision@K of link prediction for the RandNE method compared to other methods, on moderate-scale networks [1]

(proximity-order) parameter. For brevity, [1] presents the results for different q, that is 1 to 3, only for AUC scores of link prediction and the accuracy of structural role classification on American Flights, using the he same experimental setup. Figure 3.11 shows the correspondingly results. For $q1$, RandNE outperforms the $q = 1$ case. Consequently, preserving high-order proximities is important in NRL. On the other hand, to examine the impact of q time-wise, experiments on random networks (i.e. the Erdos Renyi model [32]) were conducted. The number of nodes and edges was fixed; 1.000.000 and 10.000.000 respectively. The results presented in figure 3.12 show that the execution time grows linearly with respect to the number of nodes and number of edges correspondingly.

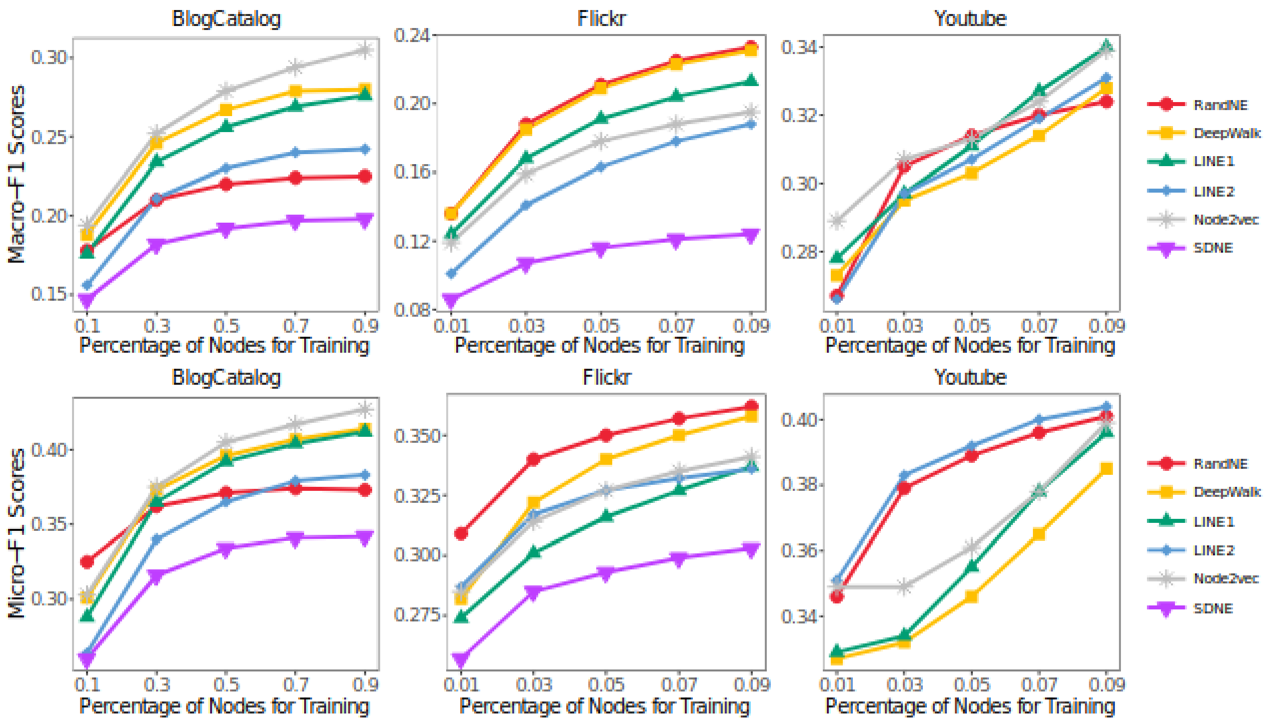


FIGURE 3.9: The results of node classification for the RandNE method compared to other methods, on moderate-scale networks [1]

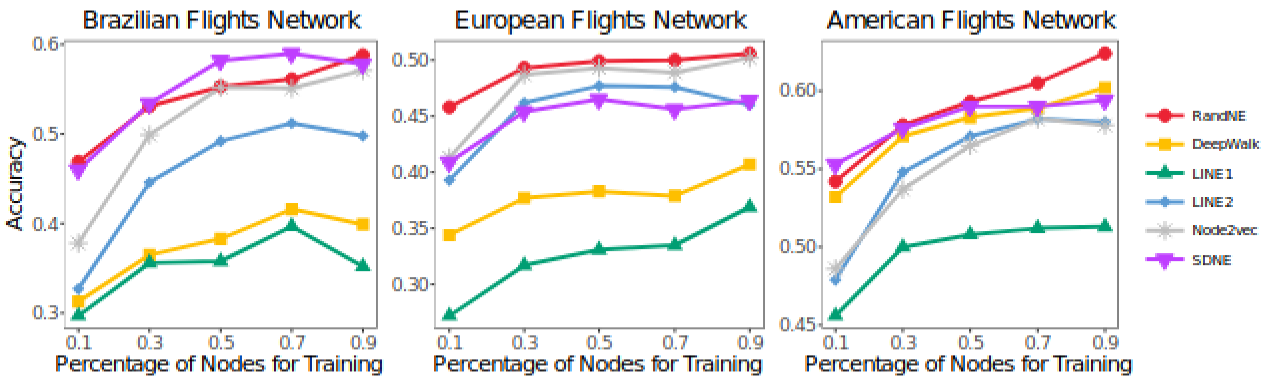


FIGURE 3.10: The accuracy of structural role classification for the RandNE method compared to other methods [1]

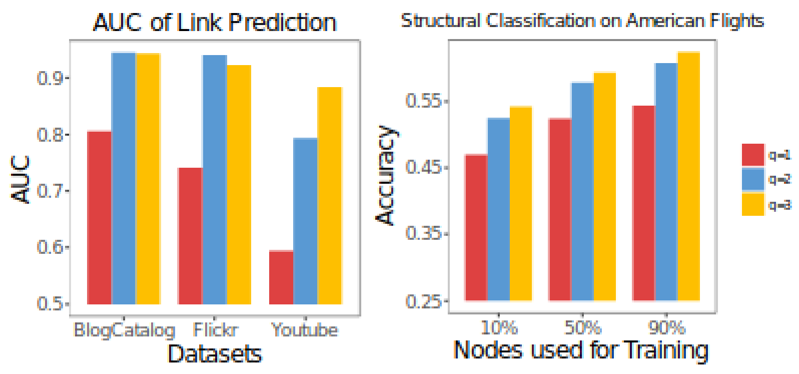


FIGURE 3.11: Parameter analysis. The high-order proximity (q > 1) greatly outperforms the simple random projection (q = 1), demonstrating the importance of preserving high-order proximities. [1]

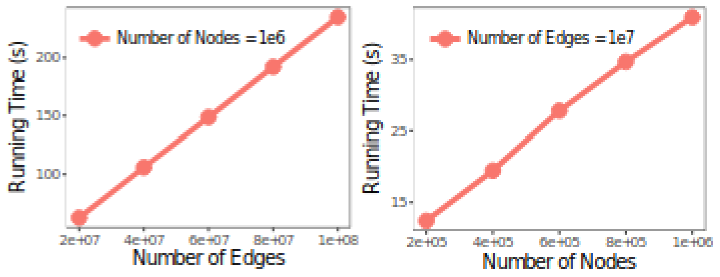


FIGURE 3.12: Scalability analysis. Figure shows the linear time complexity of our method with respect to the number of nodes and number of edges respectively. [1]

Chapter 4

Distributed Implementation Of RandNE

As mentioned in chapter 3, this thesis regard the implementation of the RandNE method [1] in a distributed manner, in order to be able to be executed on large networks. Since the objective of the thesis is to implement RandNE in a distribute manner, the Apache Spark[1] framework is utilized that is most used and efficient framework to develop distributed applications nowadays.

Outline. This chapter presents the analyzes of the RandNE implementation provided by the thesis. At first, section 4.1 analyzes the Apache Spark framework and all its features needed to be understood to comprehend the code implementation. Then, section 4.2 provides the analyses of the code implementation of the RandNE algorithm provided by the thesis, which is accessible at the following repository[2].

4.1 Apache Spark

Apache Spark is open source software that provides the capacity of distributed computing on a cluster of computers. It is mainly used for processing large volumes of data. It has its own ecosystem (figure 4.1) that contains a number of libraries and APIs, which provide the following functions; libraries

[1]Apache Spark website
[2]Distributed RandNE implementation repository

with ready to use parallel machine learning algorithms and the capacity of developing new ones (MLlib), the ability to process data streams (Spark Streaming), static data (Spark SQL and DataFrames) and graphs (GraphX), and a number of third-party libraries that can be connected to Apache Spark and provide new functionalities. The aforementioned functionalities can be used with one of the following programming languages: scala, java, python, R.

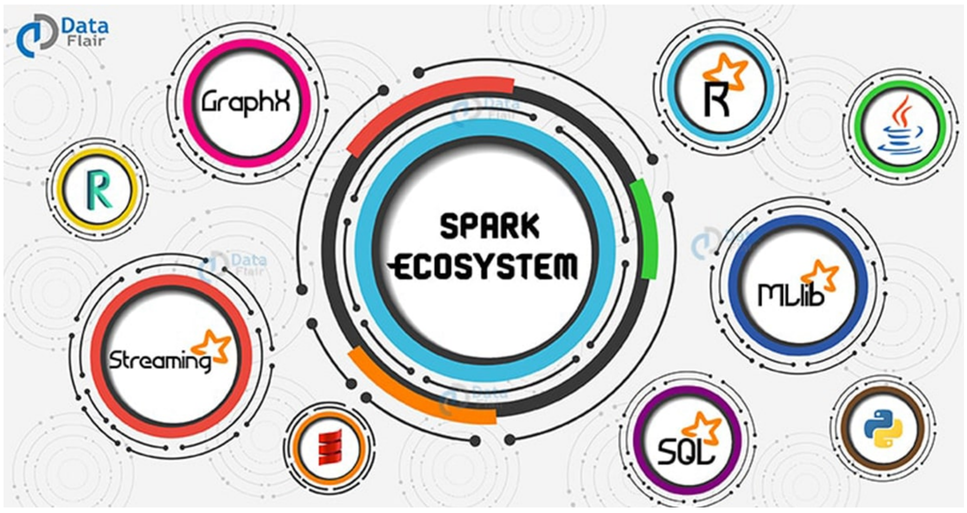


FIGURE 4.1: The ecosystem of Apache Spark [33]

Apache Spark aims to offer a centralized platform for creating large-scale data processing applications. Creating such applications is complex, as it requires the combination of many different libraries or even programs. Apache Spark simplifies this process by offering APIs that make it easy to combine programs or ready-made libraries. Furthermore, it is easier to create our own libraries or programs, by evolving existing.

Spark confines itself to a 'data processing' tool. It does not support the ability to function as a database. What it provides is the ability to load data from a variety of data sources. For example from databases that store data in key-value format (Cassandra), relational databases (MySql, PostreSql ...), distributed file system (HDFS), cloud storage systems (Azure storage, Amazon S3).

4.1.1 RDD - Data Structure

Apache Spark has its own data structures through which we process data. RDD is Spark's most basic and low-level data structure. A RDD contains an immutable set of data that splits into a number of partitions, where each partition is stored to a different machine of the cluster. Consequently, RDD is a distributed data structure. Each partition contains a distinct subset of the RDD's data. The structure of a RDD is the same as a simple list of any programming language, that is, a collection of finite elements. The data within a RDD are objects of the programming language selected by the programmer. Therefore, in a RDD we can put what we want, as in the objects of a programming language, where we can put any type of data in the format we want. Figure 4.2 shows a RDD, which immutable data collection consists of seven elements.

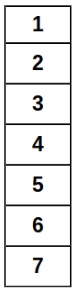


FIGURE 4.2: The structure of a RDD that contains seven elements

The data of a RDD is immutable. But in case where we want to convert its data, what we can do is to state exactly how we want the data to be converted. In order to state how we want the data of a RDD to be transformed, we utilize functions that are called transformations, which are provided by Apache Spark. After executing a transformation function to a RDD, which is executed to each individual partition of the RDD, a completely new RDD will be created. The data of the new RDD is going to be the data of the previous one, but having applied to it the conversions we wanted. The original RDD still exists, so we can make multiple different conversions to the same RDD. Figure 4.3 shows a RDD in which a transformation function takes place, resulting to a new RDD.

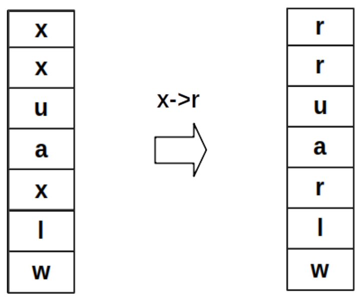


FIGURE 4.3: Two RDDs, where a transformation function is executed to the left RDD, that converts all the 'x' to 'r'. The right RDD is the new RDD created after the execution of the transformation function to the left RDD.

The two tables in Figure 4.3 present two RDDs, where their data is the data presented in the two tables respectively. To the left RDD, a transformation function is applied through which the data 'x' is mapped to 'r'. The right RDD presents the new RDD created after the execution of the transformation to the left RDD. We notice that all the 'x' are converted to 'r', and that all the rest data remain the same.

In Spark, when we want to process the data of a RDD, we first declare how we want it to be transformed and then the transformations will take place and the new RDD will be generated. However, when we declare the transformations and run the line of code in which we declared them, they will not automatically run to get the new RDD, because Spark implements lazy evaluation. This means that when Spark runs the line of code that declares the transformations, they will not take place instantly. What it does is create a transformation plan. The transformation plan contains the transformations we want to make in the RDD. To execute the transformation plan and get the new RDD, we need to call an action on the old RDD. As we declare transformations, so we declare actions through functions. Transformation and action functions are provided by Spark. The reason Spark implements lazy evaluation is to optimize the data flow of each transformation from start to finish.

4.2 Code Implementation

This section describes the distributed code implementation of RandNE, provided by the thesis, that is developed by scala[3]. Apache Spark is utilized in order to develop RandNE in a distributed manner.

At first, 4.2.1 analyzes the features of the MLlib library that are utilized in order to handle the matrices in the RandNE implementation. Next, 4.2.2 analyzes the individual transformation functions provided by Spark, that are utilized in the code implementation. Last, 4.2.3 describes how the pseudocode presented in Figure 3.1 is implemented, and how the features described in 4.2.1 and 4.2.2 are used in the implementation.

4.2.1 MLlib - Linear Algebra

RandNE includes a lot of linear algebra operations (e.g matrices product). Apache Spark provides the MLlib library, which is a scalable machine learning library. One of the features that MLlib provides is linear algebra. It provides local and distributed matrices and vectors that come with built-in methods like the dot product calculation of two vectors or transposing a matrix, and so on.

Next, the linear algebra features that are used in the code implementation (4.2.3) are presented. All the features presented below, are implemented as objects.

Dense Matrix. A dense matrix is a local matrix that preserves all the elements of the matrix. The object that implements the dense matrix has the two following signatures:

```
new DenseMatrix(numRows: Int, numCols: Int, values: Array[Double])
```

```
new DenseMatrix(numRows: Int, numCols: Int, values: Array[Double],
                isTransposed: Boolean)
```

[3]scala website

numRows and *numCols* indicate the number of rows and columns in the matrix, correspondingly. *values* indicates the values of the matrix by column-based order. In the second signature, if *isTransposed* is true, then the values are stored by row-based order, otherwise by column-based order. In the first signature, *isTransposed* is by default false.

Dense Vector. A dense vector is a local vector where all the elements are preserved. The signature of the object is the following:

```
new DenseVector(values: Array[Double])
```

values contains the values of the vector. *DenseVector* extends the *Vector* trait.

Sparse Vector. A sparse vector is a local vector where the elements that are equal to 0 are not preserved. The signature of the object is the following:

```
new SparseVector(size: Int, indices: Array[Int], values: Array[Double])
```

size indicates the size of the vector, *indices* the indices that have non-zero values, and *values* the corresponding values of *indices*. *indices* and *values* arrays have the same size, and the indices in *indices* are strictly in increasing order. *SparseVector* extends the *Vector* trait.

Indexed row. An indexed row is essentially a vector that has an index. The signature of the object is the following:

```
new IndexedRow(index: Long, vector: Vector)
```

index indicates the index of the vector, and *vector* the vector. *vector* is either a *DenseVector* or a *SparseVector*.

Indexed row matrix. An indexed row matrix is a distributed indexed matrix. The following object has the two following signatures:

```
new IndexedRowMatrix(rows: RDD[IndexedRow])
```

```
new IndexedRowMatrix(rows: RDD[IndexedRow], nRows: Long, nCols: Int)
```

In both, *rows* indicates the indexed vectors. The difference of the second signature lies to the fact that *nRows* and *nCols* indicate the number of rows and columns in the matrix correspondingly.

4.2.2 Transformations - Broadcast Variable

Next, all the individual transformation functions utilized in the code implementation are analyzed, as well the broadcast variable feature. How they are actually utilized in the code implementation, is presented in 4.2.3.

map. map(*func*) is the most used transformation function. It returns a new RDD that contains the result of applying *func*, which is a custom function stated by the programmer, to each element of the RDD that it is executed. Figure 4.3 shows a map(*func*) transformation, where *func* return 'r' if the input equals 'x', otherwise it returns the input as it is.

flatMap. flatMap(*func*) is similar to map(*func*), but each input element is mapped to 0 or more output elements, so *func* returns a sequence (e.g. list, array) rather than a single element. All the individual sequences are then flatten, resulting to a RDD that contains all the elements of all the individual sequences.

sortBy. sortBy(*func*, *Boolean*) returns a new RDD, that is the sorted version of the original RDD, where the order is resolved by *func*. If *Boolean* is true, then ascending order is applied, otherwise descending.

groupByKey. In case where a RDD contains elements of (K, V) pairs, then when we call grouByKey(*numPartitions*), a new RDD that contains elements of (K, Sequence<V>) pairs is created. *numPartitions* states the number of partitions, of which the new RDD will be constituted. *numPartitions* is optional, where by default the number o partitions of the original RDD are inherit by the new RDD.

zipWithIndex. zipWithIndex() produces a new RDD, which elements are the elements of the original RDD zipped with indices. The ordering of the indices is first-based on the partition index and then the ordering of the elements within each partition. So the first element in the first partition gets index 0, and the last element in the last partition receives the largest index.

Broadcast variable. An Apache Spark application is executed in a cluster of machines, where each partition of a RDD is placed in a different machine from the other. Consequently, variables that are accessible in one machine, are not accessible to the other. But the are cases where we want variables to be known and accessible across all the machines of the cluster. Apache Spark provides the *broadcast variable* feature in order to share variables to all the machines. A single element can be stated as a broadcast variable or even a sequence of elements.

4.2.3 Algorithm Implementation

The code implementation of RandNE is accessible in the following repository[4]. The programming language used for the implementation is scala. Below, an analyses of the logic behind the implementation is provided, instead of a detailed analysis of the code. So, in case where someone is familiar with scala and understands the features of Apache Spark presented in 4.2.1 and 4.2.2, they will easily understand the code implementation provided in the repository.

RandNE is implemented as a scala case class, named *RandNE*. A case class is similar to a simple scala class, with the difference that a case class has immutable data. A detailed analyses of *RandNE*'s parameters and their meaning is provided in the repository.

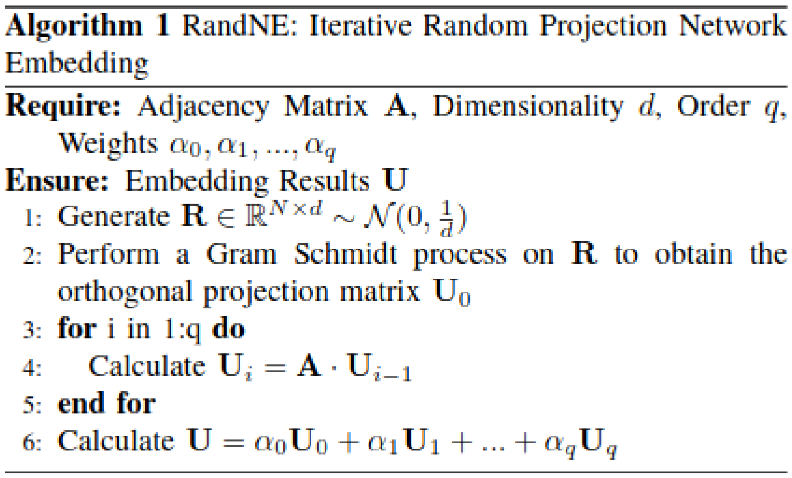
The essence behind the code implementation logic is to carry out the pseudocode of RandNE provided in [1], and presented initially in Figure 3.1. For convenience, the pseudocode is presented again in Figure 4.4.

The only differences of the code implementation and the pseudocode in Figure 4.4, are lines 4 and 6. As for line 4, the following equation is calculated in the code implementation:

$$\begin{aligned} \mathbf{U}_i &= \mathbf{A} \cdot \mathbf{U}_{i-1} \\ \Rightarrow \mathbf{U}_i^T &= (\mathbf{A} \cdot \mathbf{U}_{i-1})^T \\ \Rightarrow \mathbf{U}_i^T &= \mathbf{U}_{i-1}^T \cdot \mathbf{A}^T \end{aligned} \tag{4.1}$$

The reason why equation (4.2) is calculated is the fact that one of the two matrices in the matrices product calculation is implemented as a local matrix and the other one as a distributed matrix. The calculation of a matrices product between a local matrix and a distributed matrix is much more efficient, time-wise, than the calculation of a matrices product between two distributed matrices. Because of the nature of a matrices product calculation,

[4]Distributed RandNE implementation repository



Algorithm 1 RandNE: Iterative Random Projection Network Embedding

```
Require: Adjacency Matrix $\mathbf{A}$, Dimensionality $d$, Order $q$,
    Weights $\alpha_0, \alpha_1, ..., \alpha_q$
Ensure: Embedding Results $\mathbf{U}$
1: Generate $\mathbf{R} \in \mathbb{R}^{N \times d} \sim \mathcal{N}(0, \frac{1}{d})$
2: Perform a Gram Schmidt process on $\mathbf{R}$ to obtain the
   orthogonal projection matrix $\mathbf{U}_0$
3: for i in 1:q do
4:     Calculate $\mathbf{U}_i = \mathbf{A} \cdot \mathbf{U}_{i-1}$
5: end for
6: Calculate $\mathbf{U} = \alpha_0 \mathbf{U}_0 + \alpha_1 \mathbf{U}_1 + ... + \alpha_q \mathbf{U}_q$
```

FIGURE 4.4: The pseudocode implementation of the RandNE algorithm [1]

the right matrix of the product operation must be the local matrix and be shared to all the machines of cluster. $\mathbf{A}^T$ is selected to be shared to all the machines, despite the fact that $\mathbf{A}^T$ is of size $N \times N$ and $\mathbf{U}_{i-1}^T$ is of size $d \times N$, where $d \ll N$. Because in practice, $\mathbf{A}^T$ needs less memory than $\mathbf{U}_{i-1}^T$ to be preserved in a machine, in case where sparse matrix techniques are utilized, due to the fact that $\mathbf{A}^T$ is a very sparse matrix, while $\mathbf{U}_{i-1}^T$ is a dense matrix.

Since in line 4 equation (4.2) is calculated, in line 6 the following equation is calculated, where afterwards $\mathbf{U}^T$ is transposed to calculate the final embeddings $\mathbf{U}$.

$$\begin{aligned} \mathbf{U} &= \alpha_0 \mathbf{U}_0 + \alpha_1 \mathbf{U}_1 + \ldots + \alpha_q \mathbf{U}_q \\ \Rightarrow \mathbf{U}^T &= (\alpha_0 \mathbf{U}_0 + \alpha_1 \mathbf{U}_1 + \ldots + \alpha_q \mathbf{U}_q)^T \\ \Rightarrow \mathbf{U}^T &= \alpha_0 \mathbf{U}_0^T + \alpha_1 \mathbf{U}_1^T + \ldots + \alpha_q \mathbf{U}_q^T \end{aligned} \tag{4.2}$$

When an instance of the *RandNE* case class is instantiated, the adjacency matrix $\mathbf{A}$ is created. Since RandNE concerns undirected networks, $\mathbf{A} = \mathbf{A}^T$. The current implementation can create $\mathbf{A}^T$, loading only files that represent the networks in an edge list format. It supports both types of edge lists, where in the first one the edges are directed, and the second one where the edges are undirected. *RandNE* has an input parameter that indicates which type of edge list the loaded file is to act accordingly to create the adjacency matrix correctly. The adjacency matrix creation is performed by the method *createAdjacencyMatrix* of *RandNE*, that takes no input parameters and utilizes all the transformations function analyzed in 4.2.2. *createAdjacencyMatrix* returns a broadcast variable that is an *Array*, which represents the adjacency matrix and contains key-value pairs, where each key-value pair represents a row of the adjacency matrix; the key is a *String* that indicates the index the of the row, and the value is a *SparseVector* that represents the row. The return type signature of *createAdjacencyMatrix* is presented below:

Broadcast[Array[(String,SparseVector)]]

In order to calculate the embeddings, the *RandNE* instance must perform its *execute* method, which takes no parameters. *execute* is a method of the *RandNE* case class, that essentially implements the pseudocode in Figure 4.4. At first, *execute* creates a *list* named *U_ list* that will contain all the $\mathbf{U}_i^T$, $0 \leq i \leq q$. The type signature of each $\mathbf{U}_i^T$ is the following:

RDD[(String,Array[Double])]

Each $\mathbf{U}_i^T$ is a RDD which represents the $\mathbf{U}_i^T$ matrix and contains key-value pairs. Each key-value pair represents a row of the $\mathbf{U}_i^T$ matrix; the key is a *String* that indicates the index the of the row, and the value is an *Array* that represents the row.

$\mathbf{U}_0^T$ is the first element that is stored in *U_ list*. In order to generate $\mathbf{U}_0^T$, that is line 1 in Figure 4.4, MLlib library is utilized, which provides a *normalVectorRDD* method that generates a RDD which elements are samples drawn from the standard normal distribution $\sim \mathcal{N}(0,1)$. The elements of $\mathbf{U}_0^T$ are then mapped from $\sim \mathcal{N}(0,1)$ to $\sim \mathcal{N}\left(0,\frac{1}{d}\right)$ through a map transformation.

Next, lines 3 and 4 are implemented, where a method named *matricesMultiplication* is used in each *for* iteration, which takes as input the corresponding $\mathbf{U}_i^T$ matrix and the iteration index respectively. Finally, after calculating all the $\mathbf{U}_i^T$, $0 \leq i \leq q$, line 6 is implemented by a methods named *calculateEmbenddings* that takes as input parameter the *U_ list* variable. *calculateEmbenddings* calculates the $\mathbf{U}^T$ initially as an *IndexedRowMatrix*, and then just transposes $\mathbf{U}^T$ to find the final embeddings, by utilizing methods of the *IndexedRowMatrix* object.

Chapter 5

Performance Evaluation

After presenting in chapter 3 the implementation of RandNE [1] provided by the thesis, this chapter presents the results of the experiments that were contacted in order to examine the implementation behavior. The behavior of the implementation is examined in regard to its time efficiency.

Hardware environment. As mentioned in 4.1, Apache Spark is a framework for developing distributed application that run in a cluster of machines. But Spark provides also the capacity to run an application in parallel in a machine using its cores, simulating a distributed environment that way. Therefor the experiments were conducted in machine with the following characteristics:

- CentOS release 6.10
- Processor Intel(R) Xeon(R) CPU E7- 4860 @ 2.27GHz
- 80 cores
- 1010GB memory

Outline. As for the outline of the chapter, section 5.1 presents the datasets and their characteristics that were used in the experimentation phase concerning the distributed implementation of the RandNE method provided by the thesis. Then section 5.2 presents the results of the experiments and provides their detail analyzes in order the explain the implementation's time-wise behavior.

5.1 Datasets

In order to evaluate the time efficiency of the implementation of the RandNE algorithm [1] developed by the thesis, experiments on four different datasets are conducted. All the datasets, which are Real World datasets, are provided from the Stanford Network Analysis Project repository (SNAP[1]). Table 5.1 shows in summary the structural characteristics of the datasets. Below, information of each dataset is provided.

Dataset	**Nodes**	**Edges**	**Undirected**
Brightkite	58.228	214.078	✓
Twitch gamers	168.114	6.797.557	✓
Gowalla	196.591	950.327	✓
Amazon product co-purchasing	334.863	925.872	✓

TABLE 5.1: Summary of datasets' structural characteristics

Brightkite.The friendship network of Brightkite, which used to be location-based social networking service provider, has 58.228 nodes and 214.078 edges. Originally, the network is directed, but this dataset[2] consist only the bidirectional friendships resulting to a undirected network. It was collected using Brightkite's public API.

Twitch gamers. The Twitch gamers dataset[3] is a social network of Twitch users, where the nodes are Twitch users and edges are mutual follower relationships between them. It has 168.114 nodes and 6.797.557 edges. It was collected from Twitch's public API.

Gowalla network. The friendship dataset[4] of Gowalla is an undirected network that was collected using their public API. It consists of 196.591 nodes and 950.327 edges.

Amazon product co-purchasing network. The Amazon product co-purchasing dataset[5] was collected by crawling the Amazon website. The logic behind the network construction is that if a product i is frequently co-purchased

[1]Snap website
[2]Brightkite dataset
[3]Twitch gamers dataset
[4]Gowalla dataset
[5]Amazon product co-purchasing dataset

with product j, then the network contains an undirected edge from i to j. The network has 334.863 nodes and 925.872 edges.

5.2 Results

Intending to understand the time-wise behavior of the RandNE implementation provided by this thesis, this sections presents the execution results based on our experimentation and analyzes them in order to interpret the implementation's behavior.

A number of different executions based on different parameters were carried out in order to examine the implementation's behavior. The parameters of the experimentation setup is **1)** different q (proximity order); $q = 1$, $q = 2$, $q = 3$. **2)** different number of cores that Spark utilizes; 1, 2, 4, 8, 16. **3)** the produced embeddings have size equal to 130 in all the executions **4)** the predefined weights $a_0, a_1, \ldots, a_q$ that the RandNE implementation takes as input parameter are dummy values since its behavior is only examined in regard to its time efficiency, so the actual values does not matter. The aforementioned setup is executed for the Twitch gamers, Gowalla and Amazon product co- purchasing datasets. As for the Brightkite dataset, experiments are conducted on it to examine the implementation's behavior for different embeddings sizes, so a different setup is followed that is explained later.

Beginning with the the first setup, figures 5.1, 5.2, 5.3 and tables 5.2, 5.3, 5.4 present the execution time results for the Twitch gamers, Gowalla and Amazon product co-purchasing datasets, correspondingly. An analyses of the results is provided too. At first, the results and their analyzes is provided for the Twitch gamers dataset, next for the Gowalla dataset and then for the Amazon product co-purchasing dataset. Afterwards a comparison between the results of these three datasets is provided that discuss the impact of the network size in regard to the execution time of the RandNE implementation provided by the thesis.

Twitch gamers results. In Figure 5.1 and Table 5.2, that present the results for the Twitch gamers dataset. In Figure 5.1, the vertical axis corresponds to execution time (in seconds) and the horizontal axis is the number of

CPU cores utilized. Each line in the diagram corresponds to a different proximity value for which the behavior of the implementation is examined. As for the table, the columns present the execution time for the distinct proximity orders, where the rows present the execution time for the different number of cores utilized. The execution time results are presented in seconds.

From the results, we observe that the implementation scales from 1 to 8 cores, and that calculating embeddings for a higher proximity needs more time than calculating embeddings for a lower proximity. In the worst case scenario when only 1 core is utilized, where $q = 1$ then 256s are needed, when $q = 2$ then 295s are need and when $q = 3$ then 338s are needed. In the best case scenario where 8 cores are utilized, when $q = 1$ then 117s are needed, when $q = 2$ then 141s are need and when $q = 3$ then 180s are needed.

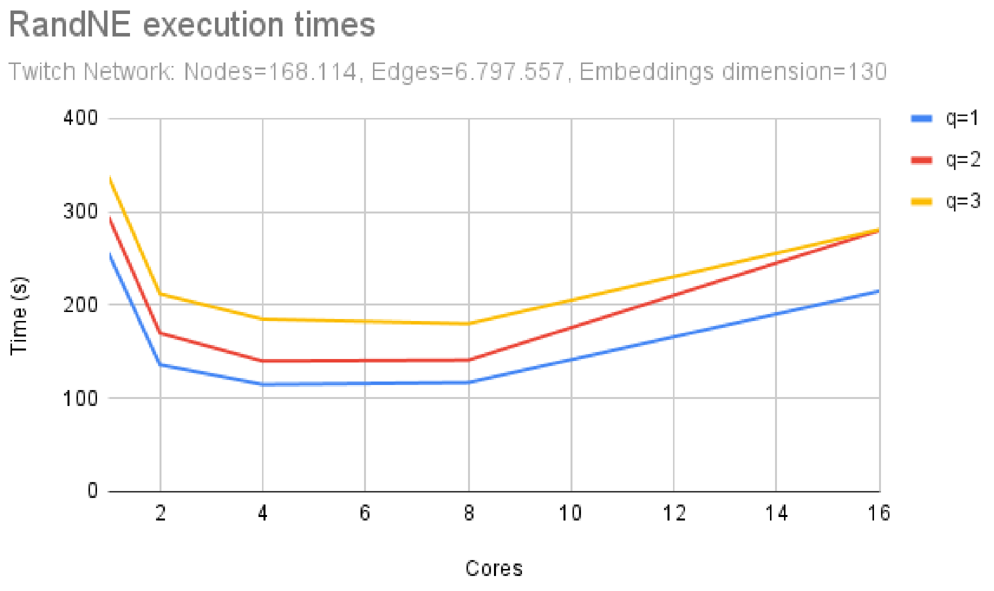


FIGURE 5.1: Diagram that presents the execution time results of the RandNE implementation provided by the thesis for the Twitch gamers dataset

Gowalla results. Figure 5.2 and table 5.3 show the results for the Gowalla dataset, and are interpreted the same as figure 5.1 and table 5.2 respectively.

In this case, the same types of results are produced as the Twitch gamers case. The implementation scales from 1 to 8 cores, and calculating embeddings

Cores/q	**1**	**2**	**3**
1	256	295	338
2	136	170	212
4	115	140	185
8	117	141	180
16	215	280	281

TABLE 5.2: Table that presents in detail the execution time results of the RandNE implementation provided by the thesis for the Twitch gamers dataset

for a higher proximity needs more time than calculating embeddings for a lower proximity. As for the worst case scenario where only 1 core is utilized, when $q = 1$ then 138s are needed, when $q = 2$ then 176s are need and when $q = 3$ then 221s are needed. In the best case scenario where 8 cores are utilized, when $q = 1$ then 64s are needed, when $q = 2$ then 76s are need and when $q = 3$ then 102s are needed.

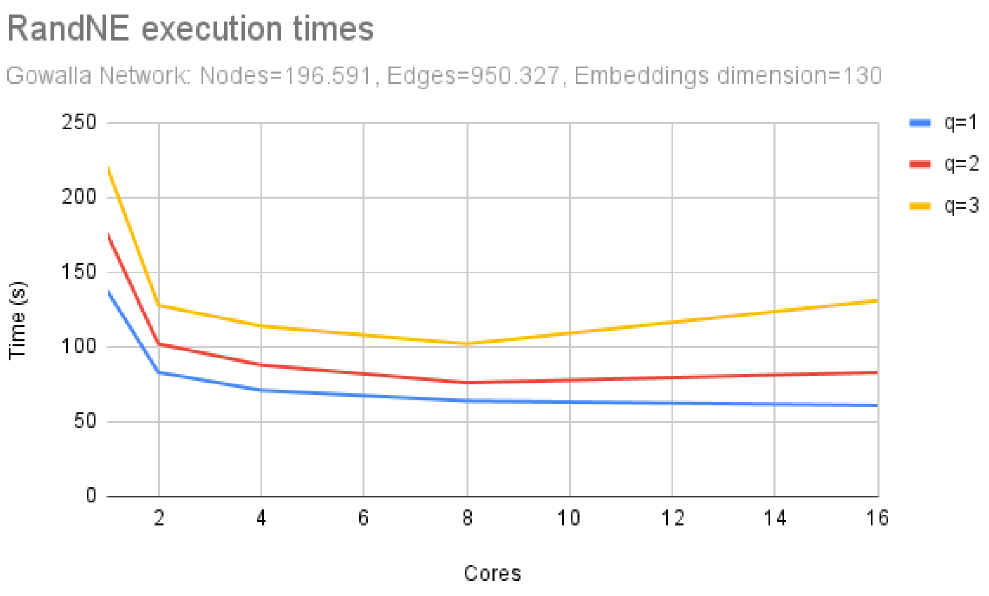


FIGURE 5.2: Diagram that presents the execution time results of the RandNE implementation provided by the thesis for the Gowalla dataset

Cores/q	**1**	**2**	**3**
1	138	176	221
2	83	102	128
4	71	88	114
8	64	76	102
16	61	83	131

TABLE 5.3: Table that presents in detail the execution time results of the RandNE implementation provided by the thesis for the Gowalla dataset

Amazon product co- purchasing results. As for the Amazon product co-purchasing dataset case, figure 5.3 and table 5.4 show the respectively results, which are interpreted as the previous figures and tables.

As for the proximity order cost, the same applies as the two previous cases, that is that calculating embeddings for a higher proximity needs more time than calculating embeddings for a lower proximity. On the other hand, in this the implementation scales from 1 to 4 cores. As for the worst case scenario where only 1 core is utilized, when $q = 1$ then 218s are needed, when $q = 2$ then 285s are need and when $q = 364$ then 221s are needed. In the best case scenario, which in this case is when 4 cores are utilized, when $q = 1$ then 123s are needed, when $q = 2$ then 157s are need and when $q = 3$ then 218s are needed.

Cores/q	**1**	**2**	**3**
1	218	285	364
2	136	165	222
4	123	157	218
8	117	218	323
16	137	209	321

TABLE 5.4: Table that presents in detail the execution time results of the RandNE implementation provided by the thesis for the Amazon product co-purchasing dataset

Network size impact. An important aspect to examine is the implementation's behavior in regard to the network size, that is the number of nodes and edges in a network. Comparing the results of the Gowalla and Amazon

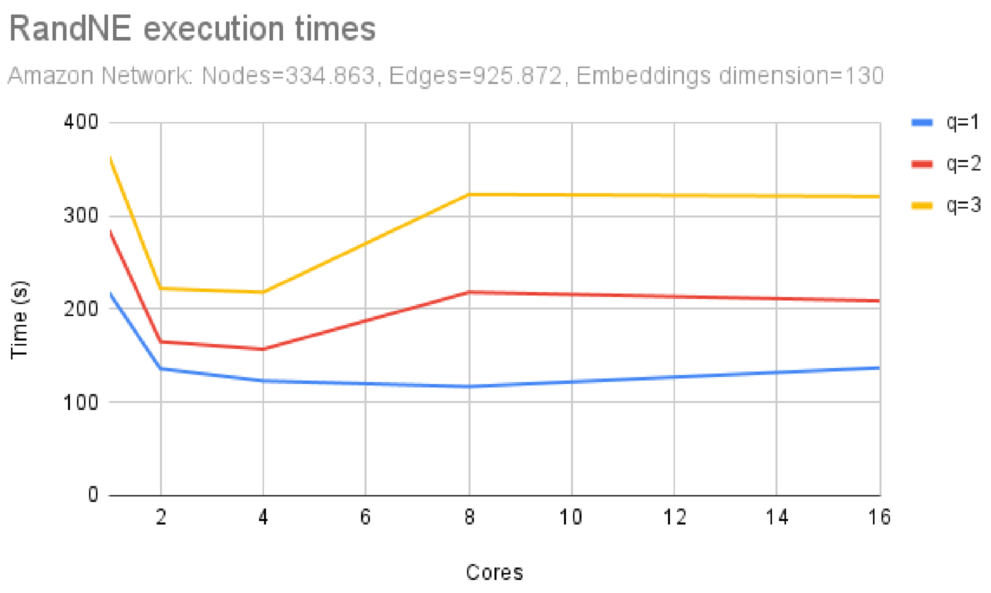


FIGURE 5.3: Diagram that presents the execution time results of the RandNE implementation provided by the thesis for the Amazon product co-purchasing dataset

product co-purchasing datasets presented in Figures 5.2, 5.3 and tables 5.3, 5.4 respectively, we can see the impact of the network's number of nodes on the execution time results of the implementation. Despite the fact that the Amazon product co-purchasing network has slightly less edges than the Gowalla network, the fact that it has more nodes leads to higher execution time results. On the other hand, comparing the results of Gowalla and Twitch gamers datasets presented in Figures 5.2, 5.1 and tables 5.3, 5.2 respectively, we can see the impact of the network's number of edges on the execution time results of the implementation. Despite the fact that the Twitch gamers network has slightly less nodes than the Gowalla network, the fact that it has approximately seven time more edges than the Gowalla network leads to higher execution time results. The conclusion of the aforementioned analyzes is that both, the number of nodes and the number of edges in a network, have impact to the execution time of the RandNE implementation provided by the thesis. More nodes and more edges in a network results to higher execution time results.

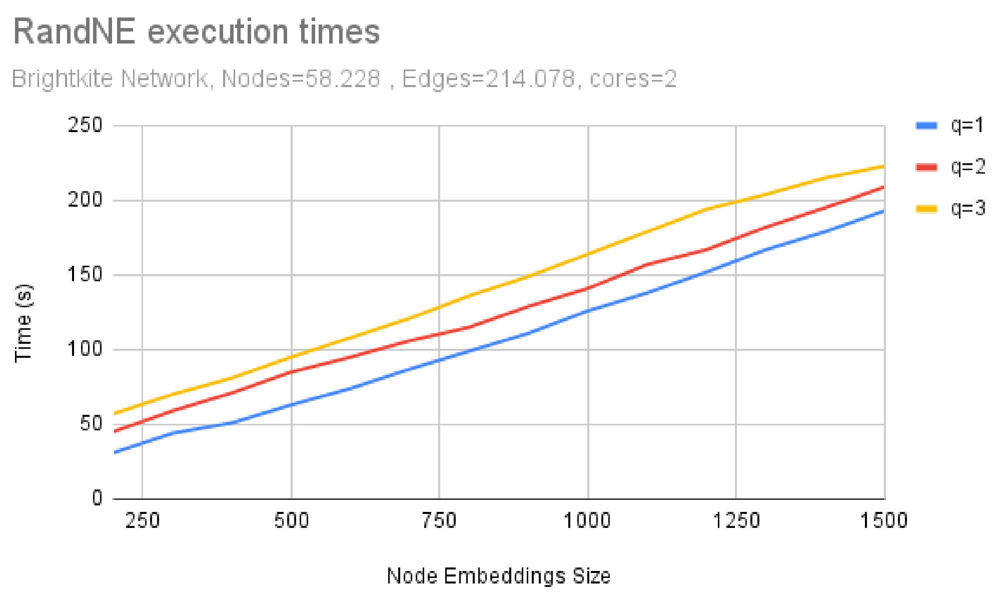


FIGURE 5.4: Diagram that presents the execution time results of the RandNE implementation provided by the thesis for different embedding sizes on the Brightkite dataset. All the execution were carried out utilizing 2 cores

Brightkite results. As mentioned, experiments are conducted on the Brightkite dataset to examine the implementation's behavior for different embeddings sizes. For this case, a number of different executions based on different parameters were carried out too. **1)** embedding sizes; from 200 to 1500 per 100. **2)** different q (proximity order); $q = 1$, $q = 2$, $q = 3$. **3)** different number of cores that Spark utilizes; 2, 4. Figures 5.4,5.5 and tables 5.5,5.6 show the results for the two distinct core cases respectively. In the diagrams of the 2 figures, the vertical axis concerns the execution time (second) results and the horizontal axis the embeddings sizes. Each line in the diagram regard a different proximity order for which the behavior of the implementation is examined. As for the the 2 tables, the columns present the execution time results for the distinct proximity orders, where the rows present the execution time results for the different size of node embeddings. The execution time results are presented in seconds.

Cores=2			
Node Embeddings size/Cores	**1**	**3**	**3**
200	31	45	57
300	44	59	70
400	51	71	81
500	63	85	95
600	74	95	108
700	87	106	121
800	99	115	136
900	111	129	149
1000	126	141	164
1100	138	157	179
1200	152	167	194
1300	167	182	204
1400	179	195	215
1500	193	209	223

TABLE 5.5: Table that presents in detail the execution time results of the RandNE implementation provided by the thesis for the Brightkite dataset. In the specific table, all the execution time results regard executions where 2 cores of the machine were utilized

Analyzing the results, we see that in both cases the execution time of the implementation is linear in regard to the embeddings sizes.

Summary. Now that all the results of the experiments regarding the RandNE implementation provided by the thesis are presented and analyzed, a summary of their conclusions is provided below.

- Both, the number of nodes and the number of edges in a network, have impact to the execution time results of the implementation. More nodes and more edges in a network leads to higher execution time results.
- The execution time of the implementation is linear in regard to the node embeddings size parameter.
- From all the results, it is clear that the execution time of the implementation increases for higher proximity orders.

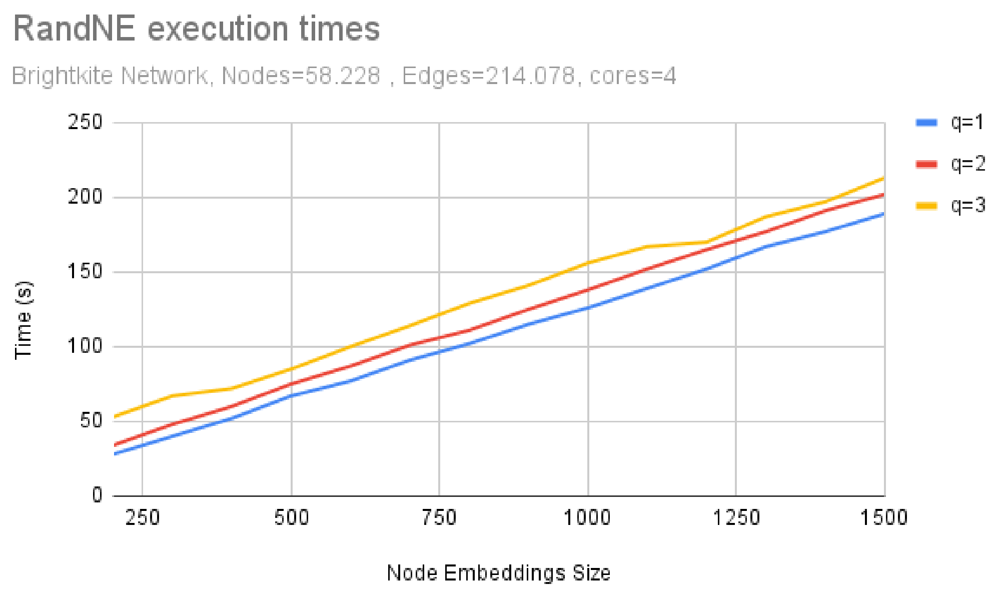


FIGURE 5.5: Diagram that presents the execution time results of the RandNE implementation provided by the thesis for different embedding sizes on the Brightkite dataset. All the execution were carried out utilizing 4 cores

- The optimal number of cores (machines in a cluster) is 4 to 8 for networks, of networks' sizes examined by the thesis. For more than 8 cores, the communication cost among the cores (machines in the cluster) is actually more than the benefits of the parallelization (distribution).

- The execution time of the RandNE implementation provided by the thesis is sufficient regarding if it is feasible to be executed in real case scenarios. As presented in the Amazon product co- purchasing dataset that contains a network with 334.863 nodes and 925.872 edges, figure 5.3 shows that the RandNE implementation needs a few minutes to be executed for optimal number of cores (machines in a cluster).

Cores=4			
Node Embeddings size/Cores	**1**	**3**	**3**
200	28	34	53
300	40	48	67
400	52	60	72
500	67	75	85
600	77	87	100
700	91	101	114
800	102	111	129
900	115	125	141
1000	126	138	156
1100	139	152	167
1200	152	165	170
1300	167	177	187
1400	177	191	197
1500	189	202	213

TABLE 5.6: Table that presents in detail the execution time results of the RandNE implementation provided by the thesis for the Brightkite dataset. In the specific table, all the execution time results regard executions where 4 cores of the machine were utilized

Chapter 6

Conclusions

This thesis makes an effort to research the network representation learning field (NRL) in regard to its scalability area. More specifically, utilizing the Apache Spark framework, a distributed implementation of the RandNE method [1] is provided by the thesis. RandNE is a method that Zhang and his colleagues presented at 2018 in a centralized version, and that showed a great promise to be time-wise feasible to be applied in large networks.

Despite fact that the thesis examines the time-wise behavior of the distributed implementation of the RandNE method, its contribution goes beyond that. This thesis is a great entry point for anyone that is not familiar with NRL field. Chapter 1 introduces the NRL field and analyzes its applications, challenges and the notations and definitions used in this area by all the resources available in the literature. Next, chapter 2 presents and analyzes the most important methods that have been developed in the literature in regard to the NRL field. Since the thesis examines the RandNE method, a detail analyzes of it could not be missed, which is provided by chapter 3. After analyzing the RandNE method (chapter 3), chapter 4 presents its distributed implementation provided by the thesis. As mentioned, the Apache Spark framework is utilized for the implementation's development, so chapter 4 provides an introduction the Apache Spark framework discussing its fundamental concepts and the programming manner in which an Apache Spark application is developed. Chapter 4 discusses also the programming logic behind the implementation provided by the thesis, as well an analyzes of the code implementation which is available at the following repository[1].

[1]Distributed RandNE implementation repository

Finally chapter 5 presents the most important contribution of the thesis by presenting the results of the experimentation phase, which shows the time-wise behavior of the distributed RandNE implementation. Based on the results, the most important conclusions are **1)** RandNE is method that is time-wise feasibly to be executed on large networks **2)** the network size, that is the number of nodes and edges, impacts the execution time of the RandNE implementation and **3)** the execution time increases for higher proximity orders (higher q) in the same network.

6.1 Future work

The distributed RandNE implementation provided by the thesis, except from examining the time-wise applicability of the RandNE method in large networks, could also be used to examine more concepts; **1)** the efficiency of the produced node embeddings by the RandNE implementation for various machine learning tasks (1.2) for large networks and **2)** extend the RandNE method for attributed networks and examine the execution time efficiency and the produced node embeddings efficiency for various machine learning tasks (1.2), both for large networks.

Embeddings efficiency. In order to examine the efficiency of the RandNE method in various machine learning tasks for large networks, the provided distributed RandNE implementation could be utilized as it is. The only needed is to execute the implementation to produce the node embeddings, which are then utilized to train machine learning models, where afterwards the results are analyzed to examine the node embeddings efficiency, consequently the efficiency of the RandNE method.

RandNE for large attributed networks. In 2017, Weiyi Liu and his colleagues proposed a multilayer network method [34], which regard the NRL field. Each network is constructed as a multilayer network, where each layer represents the topological structure of a group of nodes corresponding to a particular relationship. [34] suggests three techniques to handle a multilayer network; **1)** *network aggregation*, **2)** *results aggregation* and **3)** *layer co-analysis*.

1) and **2)** can be utilized to extend the RandNE method [1] for large attributed networks. In **1)**, all the edges from the different layers are assumed to be equal [35], [36]. Consequently, a single weighted network is constructed by aggregating all the individual networks of the multilayer network. Afterwards, any existing NRL method can be applied to the produced single network. On the other hand, in **2)** the assumption is that the edges of distinct layers are totally different from each other [37], so they cannot be merges into a single network. A NRL methods is applied to each layer separately, and then the produced node embeddings are merged to construct the final node embeddings.

In order to extend RandNE for large attributed networks utilizing the techniques proposed in [34], a two-layer network must be constructed, where both layers are programmatically preserved as an adjacency matrix. The first one preserves the structural neighbors for each node in the network. The second one preserves the neighbors of the nodes again, but in this case the node's neighbors are not all the nodes that are connected to them with an edge, as in the first case. Instead, the neighbors of a node are defined based on their attribute-similarity, which can be calculated by any corresponding technique. After calculating the attribute-similarity among the nodes, who are the final neighbors of a node can be decided by one of the following logic; **i)** top-k neighbors, that is the first k nodes with the highest attribute-similarity and **ii)** the nodes that have attribute-similarity higher than a threshold value. After the construction of the two-layer network, the distributed implementation of the RandNE method provided by the thesis can be utilized as it is to execute **1)** and **2)** proposed by [34] to produce node embeddings for large attributed networks. This coupling of the RandNE method with [34] has never been done to produce embeddings for large attributed networks, until the day that the thesis had been written, consequently its examination is worth to be carried to study the results.

Bibliography

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

[2] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey", *IEEE transactions on Big Data*, vol. 6, no. 1, pp. 3–28, 2018.

[3] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey", *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[4] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications", *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[5] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding", *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 5, pp. 833–852, 2018.

[6] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications", *arXiv preprint arXiv:1709.05584*, 2017.

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

[8] J. Tang, M. Qu, M. Wang, *et al.*, “Large-scale information network embedding”, in *Proceedings of the 24th international conference on World Wide Web. International World Wide Web Conferences Steering Committee*, pp. 1067–77.

[9] A. Grover and J. Leskovec, “Node2vec: Scalable feature learning for networks”, in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[10] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, “Struc2vec: Learning node representations from structural identity”, in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 385–394.

[11] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec, “Spectral graph wavelets for structural role similarity in networks”, 2018.

[12] R. Feng, Y. Yang, W. Hu, F. Wu, and Y. Zhang, “Representation learning for scale-free networks”, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

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

[14] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang, “Network representation learning with rich text information”, in *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[15] D. Zhang, J. Yin, X. Zhu, and C. Zhang, “Homophily, structure, and content augmented network representation learning”, in *2016 IEEE 16th international conference on data mining (ICDM)*, IEEE, 2016, pp. 609–618.

[16] C. Li, S. Wang, D. Yang, *et al.*, “Ppne: Property preserving network embedding”, in *International Conference on Database Systems for Advanced Applications*, Springer, 2017, pp. 163–179.

[17] J. Li, J. Zhu, and B. Zhang, "Discriminative deep random walk for network classification", in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 1004–1013.

[18] C. Tu, W. Zhang, Z. Liu, M. Sun, *et al.*, "Max-margin deepwalk: Discriminative learning of network representation.", in *IJCAI*, vol. 2016, 2016, pp. 3889–3895.

[19] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Collective classification via discriminative matrix factorization on sparsely labeled networks", in *Proceedings of the 25th ACM international on conference on information and knowledge management*, 2016, pp. 1563–1572.

[20] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation", *Network*, vol. 11, no. 9, p. 12, 2016.

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

[22] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering.", in *Nips*, vol. 14, 2001, pp. 585–591.

[23] N. Natarajan and I. S. Dhillon, "Inductive matrix completion for predicting gene–disease associations", *Bioinformatics*, vol. 30, no. 12, pp. i60–i68, 2014.

[24] J. Zhu, A. Ahmed, and E. P. Xing, "Medlda: Maximum margin supervised topic models", *the Journal of machine Learning research*, vol. 13, no. 1, pp. 2237–2278, 2012.

[25] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, "Supervised dictionary learning", *arXiv preprint arXiv:0809.3083*, 2008.

[26] Q. Le and T. Mikolov, "Distributed representations of sentences and documents", in *International conference on machine learning*, PMLR, 2014, pp. 1188–1196.

[27] S. S. Vempala, *The random projection method.* American Mathematical Soc., 2005, vol. 65.

[28] E. Liberty, "Simple and deterministic matrix sketching", in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 581–588.

[29] T. Fawcett, "An introduction to roc analysis", *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[30] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding", in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1225–1234.

[31] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification", *the Journal of machine Learning research*, vol. 9, pp. 1871–1874, 2008.

[32] P. Erdős, A. Rényi, *et al.*, "On the evolution of random graphs", *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[33] *Apache spark ecosystem - complete spark components guide*, Accessed = 2022-01-24. [Online]. Available: https://data-flair.training/blogs/apache-spark-ecosystem-components/.

[34] W. Liu, P.-Y. Chen, S. Yeung, T. Suzumura, and L. Chen, "Principled multilayer network embedding", in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, IEEE, 2017, pp. 134–141.

[35] C. W. Loe and H. J. Jensen, "Comparison of communities detection algorithms for multiplex", *Physica A: Statistical Mechanics and its Applications*, vol. 431, pp. 29–45, 2015.

[36] S. Boccaletti, G. Bianconi, R. Criado, *et al.*, "The structure and dynamics of multilayer networks", *Physics reports*, vol. 544, no. 1, pp. 1–122, 2014.

[37] M. Berlingerio, F. Pinelli, and F. Calabrese, "Abacus: Frequent pattern mining-based community discovery in multidimensional networks", *Data Mining and Knowledge Discovery*, vol. 27, no. 3, pp. 294–320, 2013.