Spring 5-25-2021

Higher-order Link Prediction using Node and Subgraph Embeddings

Kalpnil Anjan

Higher-order Link Prediction using Node and Subgraph Embeddings

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Kalpnil Anjan

May 2021

The Designated Project Committee Approves the Project Titled

Higher-order Link Prediction using Node and Subgraph Embeddings

by

Kalpnil Anjan

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2021

Dr. Katerina Potika	Department of Computer Science
Dr. Chris Pollett	Department of Computer Science
Dr. William Andreopoulos	Department of Computer Science

ABSTRACT

Higher-order Link Prediction using Node and Subgraph Embeddings

by Kalpnil Anjan

Social media, academia collaborations, e-commerce websites, biological structures, and other real-world networks are modeled as graphs to represent their entities and relationships in an abstract way. Such graphs are becoming more complex and informative, and by analyzing them we can solve various problems and find hidden insights. Some applications include predicting relationships and potential links between nodes, classifying nodes, and finding the most influential nodes in the graph, etc.

A large amount of research is being done in the field of predicting links between two nodes. However, predicting a future relationship among three or more nodes in a graph is a more recent active research topic. Relationships that involve more than two nodes is called a higher-order link. One of the approaches, that we follow in this work, is that of mapping the graph entities, such as nodes, edges, and triangles, into a low dimensional space by generating embeddings vectors. In that way, we work with vectors and reduce the higher-order link prediction to a classification problem.

The primary objective of this project is to utilize the GloVeNoR node embedding technique, as well as Simplex2Vec triangle embedding technique, to perform higher-order link prediction, i.e., to predict the possibility of interaction. Additionally, we evaluate the predictions generated by our methods and compare them with existing higher-order link prediction approaches using benchmark datasets. Based on our experiments, we show that the triangle embeddings generated using the techniques discussed in the report increase the average performance over the five datasets evaluated using the AUC-PR relative to random baseline as a metric for higher-order link prediction by 48%.

ACKNOWLEDGMENTS

I am deeply grateful to my project advisor, Dr. Katerina Potika for her valuable inputs and guidance throughout every stage of my master's project. Her immense expertise in the domain of graphs and social network gave the impetus for better understanding of the related topics and shaping my research approach. I appreciate her constant encouragement and efforts to steer me in the most efficient direction.

I would also like to extend my sincerest gratitude towards Dr. Chris Pollett and Dr. William Andreopoulos, for being a part of my defense committee. I would like to thank them for their time and valuable feedback.

Lastly, I am humbled to acknowledge the unwavering faith and support of my family and friends. Their encouragement when the going got tough was duly noted and much appreciated.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

In computer science, the graph data structure is used to illustrate the relationships between different data items or entities. This property of providing information about relationships between different objects has enabled graphs to be widely used in different fields to visualize data and find insights from the data. Graphs are now being used in fields like social networks, academia, chemistry, e-commerce websites, etc. With huge chunks of data being generated in these fields, the graphs have become more complex as well as more informative. Predicting relationships and potential links between nodes, classifying the similar nodes, finding the most influential nodes in the graph, etc. are some of the applications that have emerged with the rise in studying graphs.

With an increase in the volume of data being generated due to businesses shifting to the digital environment, there is an ever-increasing demand to use the data to uncover vital insights. This increase in data has made the graphs more complex and informative, prompting businesses to use the graph data to find insights that can be used to ameliorate business as well as find related entities. Since most of the graphs represent data where there are multi-node interactions, higher-order link prediction could be used to uncover these interactions that could be used to enhance network analysis. This report discusses the implementation of higher-order link prediction on data from a number of different domains using node embedding and triangle embedding approaches.

1.1 Node Representation

Graphs, in their pictorial form, provide useful insights into the network structure. However, applications like friend recommendation in social media, link prediction, etc., require machine learning algorithms to be applied to the graph data. To use machine learning algorithms on graphs, the graph data needs to be encoded or represented in a

way that can be used the machine learning algorithms [7]. Vectors or numerical data are the most popular data type used in implementing numerous machine learning techniques [8]. Thus, to use machine learning algorithms on graph data, mapping graph data into vectors is being done extensively researched topic [8]. This technique to compute a corresponding low-dimensional vector using the graph data and represent the graph nodes or higher-order structures using vectors is known as representation learning [8].

To represent the nodes of a graph in terms of vectors, features of the nodes and their interactions with the neighboring nodes need to be extracted. Node embedding is a technique that can be used to present the nodes of a graph in lower-dimensional vectors of different sizes called feature vectors [9]. Using node embedding, the features of the nodes in a graph are extracted and the node is mapped to a lower-dimensional space such that the nodes retain their similarity from the original network in the embedding space. Node embedding typically involves three major stages: defining an encoder function, defining a decoder function, and optimizing the encoder-decoder function [10]. The encoder function is used to map the nodes in a graph into a lower-dimensional vector that represents the node in the embedding space. The decoder function is used to reconstruct or decode the graph statistics of the nodes from the node embedding generated by the encoder function [10]. Optimizing the encoder-decoder function involves reducing the loss function and ensuring that the encoder and decoder function can accurately represent the node.

The accuracy of the tasks like node classification, link prediction, and other applications on graphs, involving the use of machine learning algorithms depends on the quality of features extracted using the node embedding technique. Thus, to ensure these tasks achieve high accuracy, selecting the most suitable node embedding technique based on the desired features is important. The node embedding techniques

can be classified into different types based on the features that are extracted as well as the graph structure (node or edge or subgraph) they represent as vectors. Deep Walk [11], node2vec [4], global vectors for node representations [9], [12] are few of the famous node embedding techniques. Techniques like node2vec [4] and global vectors for node representations [9], [12] are derived from word embedding techniques used for natural language processing.

1.2 Link Prediction

Link prediction is a well-known and significant area of research that has applications in numerous fields where data can be represented using graphs. Link prediction can be defined as the technique of predicting future relationships in a graph, i.e., predicting the links that will be formed in a graph in the future. Friend recommendation in social networks, product recommendation in e-commerce applications, interdisciplinary collaboration in academia, etc., are some of the applications of link prediction techniques [13]. With the advancement of technology in various fields where data can be represented as graphs, link prediction is now being extensively used to enhance the existing systems and utilize the data being generated.

A link prediction problem can be defined as, given a graph network G consisting of nodes representing entities and links representing the relationships between the entities, which nodes are going to be most likely connected in the near future i.e., the nodes that are not connected right now [14]. Link prediction can be classified based on the number of nodes involved in the link prediction. Link prediction involving two nodes in a graph is known as pairwise link prediction, while those involving more than two nodes are known as higher-order link prediction. Pairwise link prediction has been extensively researched since it's computationally less expensive for large datasets and does not require temporal as well as structural information of the entire graph [1].

Though pair-wise link prediction solves the rudimentary problem of predicting links between two nodes, there are relationships in a network that would involve interactions between higher-order structures in a graph. Multiple people interacting in social media, different concepts being integrated into a single research topic, multiple products bought together from an e-commerce website, etc. are some of the examples where multiple entities represented as nodes in a graph interact together [1]. The higher-order interactions in a graph cannot be predicted by visualizing the graph structure. It requires temporal data i.e., the graph data at a particular time instance, and is not necessarily used while performing pair-wise link prediction or in many cases, eliminated from the graph if the graph is not constructed over time [1]. Due to the lack of temporal graph data, research on higher-order link prediction has been limited to the few available datasets [1].

1.3 Motivation and Problem Statement

As discussed in the previous section, link prediction is in increasing demand for the graphs used to represent data in domains like academic collaborations, social media, e-commerce applications, chemistry involving drugs and chemical reactions, etc. However, interactions in all of the domains are not limited to two entities, thus accurate and efficient higher-order link prediction algorithms are required to predict potential interactions involving three or more entities. Higher-order link prediction has been a budding research topic and yet to be explored vastly.

One of the major reasons for research in higher-order link prediction being sparse is due to the fact that finding such interactions is computationally expensive. Thus, for the scope of this project, the multi-node interactions are limited to three nodes, i.e. interactions forming a closed triangle. Expanding the pairwise link prediction for predicting relationships among three nodes would ensure that the technique is

computationally feasible, as well as useful to experiment with higher-order prediction. This project focuses on implementing and modifying the existing node embedding technique i.e., Global Vectors for Node Representation (GloVeNoR) [9] and a subgraph embedding technique, Simplex2Vec [5] for predicting the probability of higher-order interactions in the future in a temporal graph.

The project report is organized into seven chapters. Chapter 2 describes the important terminologies related to graphs and higher-order link prediction. Chapter 3 discusses the existing node embedding, subgraph embedding, and link prediction techniques that are used as a reference for the technique proposed in this project. Chapter 4 provides details about the proposed approach and the algorithms used for project implementation. Chapter 5 enlists the datasets used for the project and the experiments performed for the project, along with the evaluation metrics. Chapter 6 presents the results for the experiments carried out in the project and a comparative study with the results from past research projects. Chapter 7 concludes the project and mentions the future potential improvements to the proposed approach.

CHAPTER 2

Terminology

This chapter provides an overview of the terminologies that are used throughout the report and for the project.

- **Graph:** A graph can be defined as a non-linear data structure representing the entities as a set of vertices or nodes and the relationships between the entities as links between the nodes called edges [15]. Graphs are used to represent many real-world networks like social media networks, academic networks of collaborators, protein interactions in chemistry, etc.

  Figure 1 below shows the graph structure and its corresponding adjacency matrix and adjacency list representation. The number of rows and columns of an adjacency matrix is the same as the number of nodes present in the graph. A link between the i^{th} node and the j^{th} node is represented by a value (in this case is 1) in the i^{th} row and j^{th} column as well as the j^{th} row and i^{th} column of the matrix if the direction of link is not specified. In the adjacency list, each node in the graph has a separate list that specifies the nodes it is connected with. The adjacency matrix for the graph depicted in figure 1 is:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

  The adjacency list of the graph can be represented as:

  $0 : 2 \longrightarrow 4 \longrightarrow 5$

  $1 : 4 \longrightarrow 5$

$2 : 0 \longrightarrow 3$

$3 : 2$

$4 : 0 \longrightarrow 1 \longrightarrow 5$

$5 : 0 \longrightarrow 1 \longrightarrow 4$

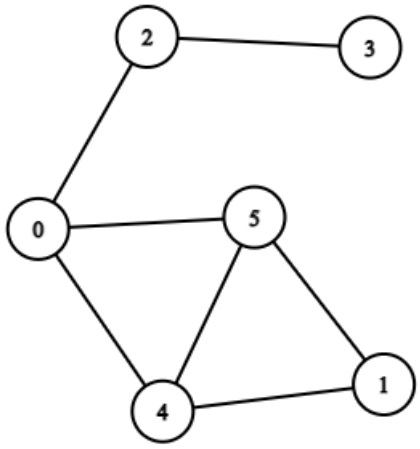

Figure 1: Sample Undirected Graph

- **Temporal Graphs:** Temporal graphs can be defined as the graphs that are not complete at any given time instance and the relationships and entities in the graphs evolve i.e. new relationships or nodes being added as time progresses [16]. In temporal graphs, new nodes and edges can be added and the graph is always evolving. Temporal graphs can be used to depict dynamic systems where the data evolves or changes over a course of time. Figure 2 depict a temporal graph where the graph structure evolves over time instances t_1, t_2 $\ldots t_n$.
- **Higher-order Structure in Graphs:** Interaction between three or more nodes in a graph can be visualized as a sub-graph of the original graph and is known as a higher-order structure [17]. In figure 2, the time instances t_1, t_2, t_5, and, t_8 represent higher-order structures since there are three or more nodes interacting

$t_1 : \{1, 2, 3, 4\}$
$t_2 : \{1, 3, 5\}$
$t_3 : \{1, 6\}$
$t_4 : \{2, 6\}$
$t_5 : \{1, 7, 8\}$
$t_6 : \{3, 9\}$
$t_7 : \{5, 8\}$
$t_8 : \{1, 2, 6\}$

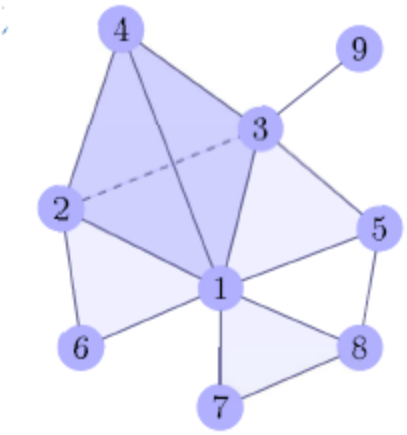

Figure 2: Temporal Graph [1] [2]

with each other at a single instance.

- **Simplex:** In terms of a graph, a simplex can be defined as a set of nodes interacting with each other. For temporal graphs, a set of nodes appearing together at a time instance can be considered a simplex. In figure 2, the nodes 1, 2, 3, 4 appearing at time instance t_1 form a 4-node simplex.
- **Open Triangle:** For temporal graphs, an open triangle can be defined as a sub-graph of three nodes where every pair of the three nodes have co-appeared at a single time instance in a simplex but none of the time instances contain all the three nodes together [1]. In figure 2, the nodes 1, 5, 8 forms an open triangle since the pairs of nodes appear in the time instances t_2, t_5 and t_7, while none of the time instances have all the three nodes.
- **Closed Triangle:** A closed triangle in temporal graphs is a higher-order structure of three nodes where all the three nodes appear together in at least one of the simplices [1]. In figure 2, the nodes 1, 3, 5 form a closed triangle since all the three nodes appear together in the time instance t_2.
- **Simplicial Closure:** Simplicial closure can be defined as a time instance where a subset of nodes in the graph, co-appear together thus forming a closed simplex.

In figure 2, a simplicial closure event occurs for nodes 1, 2, 6 at time instance t_8 since all the nodes appear together.

- **Simplicial Complex:** A simplicial complex can be defined as a type of graph where the nodes of the graph are replaced by higher-order simplices i.e. edges, triangles, tetrahedrons, etc. If simplicial complex (X) consists of multiple higher-order simplices $S_1, S_2, S_3 ... S_n$ then each of the subsets of the simplices must also be a part of the simplicial complex [18]. In simple terms, a simplicial complex is a graph with higher-order simplex representing the node. Figure 3 represents a simplicial complex as well as the different simplices that are used to form the simplicial complex.

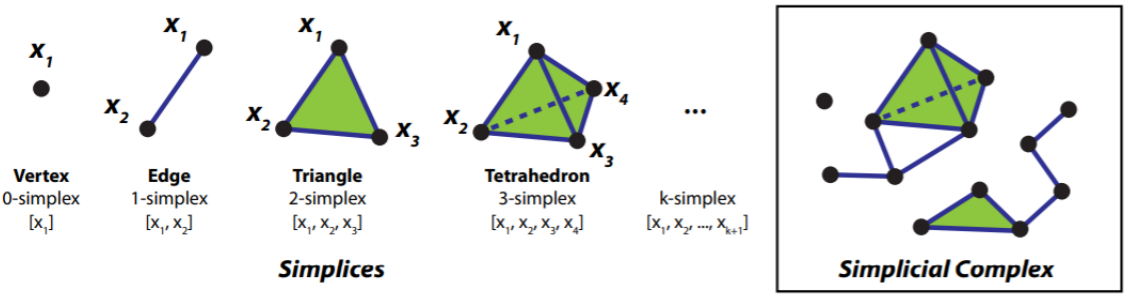

Figure 3: Simplices of different orders and simplicial complex [3]

- **Node Embedding:** Node embedding consists of a vector that is used to represent the node of a graph in a lower-dimensional space. Figure 4 represents a high-level diagram depicting node embedding in a graph.
- **Random Walk:** A random walk can be defined as a finite sequence of nodes that is formed by selected a random node in the graph and traversing the neighbors of the node up to a finite length [19]. Random walks are used to extract information from the network about a node and its neighbors. Figure 5 shows a sample random walk on the graph starting from node 1.
- **Hasse Diagram:** A Hasse diagram can be defined as a directed acyclic graph

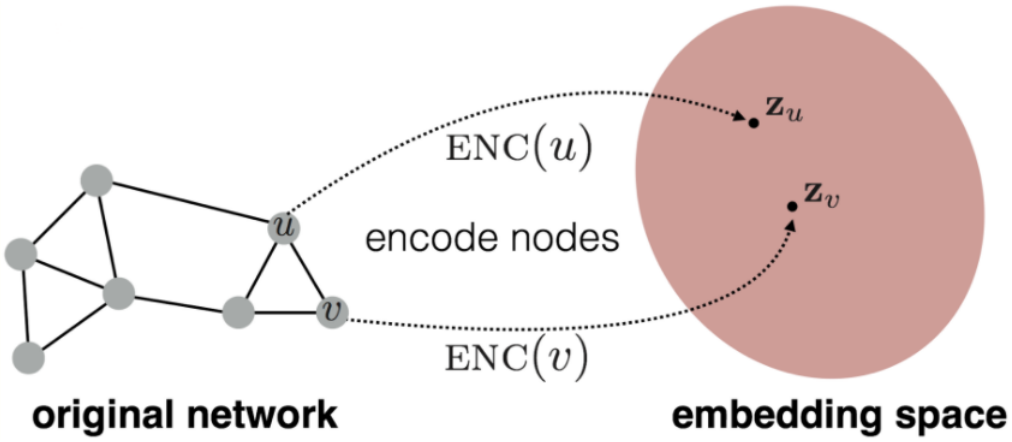

Figure 4: Embedding nodes of a graph [4]

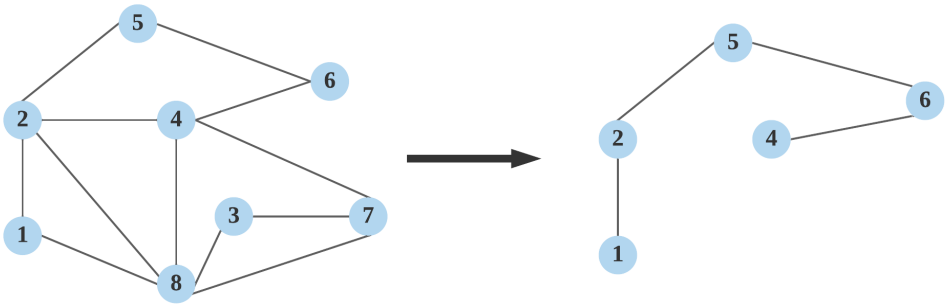

Figure 5: Sample Random Walk on a graph

representing the ordering relationships between the elements of a partial-ordered set. The Hasse Diagram will contain an edge between two elements (X and Y) directed from X to Y if x is a subset of Y and there is no other path from X to Y [20], [21]. For the two elements i.e., X and Y, the edge would be directed upward from X to Y.

For simplicial complexes, the Hasse Diagram is used to represent the ordering relationships between the simplices of different orders. The simplices in a simplicial complex are represented as nodes in the Hasse Diagram [21]. The Hasse diagram will have an edge from the (k-1) simplex to k-simplex if the (k-1)

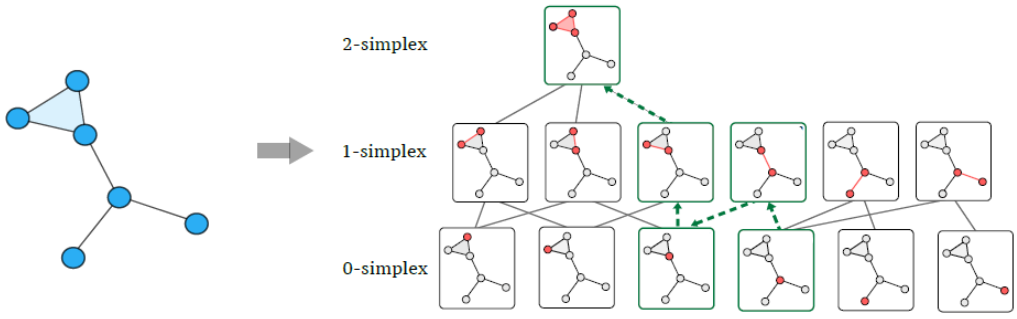

Figure 6: A Simplicial Complex and it's corresponding Hasse diagram [5]

simplex is a subset of the k-simplex. The figure 6 shows a Hasse diagram created for a graph (simplicial complex) created using simplices.

CHAPTER 3

Related Work

This chapter describes the previous research that is done for generating node embeddings and triangle embeddings for a graph as well as the technique used for higher-order link prediction. Global vectors for node representation [9] presents an approach for calculating node embeddings while preserving the global context of a node. Simplex2Vec [5] proposes a technique for generating embeddings for simplices or subgraphs using the word2vec algorithm. The section further discusses the technique used for higher-order link prediction in [2].

3.1 Global Vectors for Node Representations (GloVeNoR)

Global vectors for node representation (GloVeNoR) proposes a node embedding technique that can be used for generating embeddings for the nodes in a graph using the local as well as the global context of the node. [9]. The intuition behind GloVeNoR [2] for generating node embeddings, is based on the technique called Global Vectors for word representations (GloVe) [22] that is used for generating vector embeddings for the words in a corpus. The GloVe word embedding technique was developed for natural language processing (NLP) to represent words in the form of vectors and find similar words [9], [22]. The GloVe algorithm uses an unsupervised regression model that incorporates both the local context well as the global context similarity of the words for computing their corresponding embeddings.

GloVe uses word-to-word co-occurrence statistics that could reveal the similarities between words appearing throughout the corpus since it considers the global context using the context window [22]. Based on this intuition, the authors of GloVeNoR tried to integrate the technique in graphs to generating embeddings for nodes that would have similar characteristics in the graph. For GloVeNoR, the corpus for calculating the co-occurrence statistics as well as for incorporating global context is generating

using second-order random walks on the graph.

GloVe [22] is an unsupervised regression model that uses the local context as well as the global context window similarity for computing word embeddings, [9]. The intuition behind this technique is that it uses the global co-occurrence matrix to extract information about word similarities. GloVeNoR uses the same embedding technique for community detection problems since GloVe performs well for tasks that involve finding word similarities since communities in the graph are viewed as a group of nodes that are semantically similar.

The corpus for GloVeNoR is generated using second-order random walks on the graph. The author state that second-order random walks provide flexibility of neighborhood exploration and the random walks can be sampled between breadth-first search (BFS) and depth-first search (DFS) by setting the value of their parameters i.e. p and q as 1 [9]. The cost function used by the model is a function of weighted least squares and for dealing with outliers and co-occurrences that are sparse, the model uses a weight function [9]. The objective function for the model can be stated mathematically as:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \cdot \tilde{w_j} + b_i + \tilde{b_j} - logX_{ij})^2 \tag{1}$$

where:

i = main word

j = context word

w_i = main word vector

w_j = context word vector

X = co-occurrence matrix

b_i, b_j = bias vectors

V = vocabulary size i.e. corpus

$f(X_{ij})$ = Weight function

Algorithm 4 describes the approach used for calculating node embeddings using global vectors. Since GloVe uses a corpus for calculating the local and global context statistics, a similar corpus for the graph is created using second-order random walks or biased random walks. The second-order random walks use the in-out hyper-parameter (p) and return hyper-parameter (q). The values of these hyper-parameters are used to sample the random walks and perform BFS or DFS based on their values. If the value of the return parameter i.e. q is low then the random walk performs a BFS-like walk and if the value of the in-out parameter i.e. p is low, then it performs a DFS like walk.

The corpus created using the random walks is then used to compute the co-occurrence matrix of size |N| x |N| where N is the number of nodes in the graph [9]. The co-occurrence matrix values for two nodes i and j i.e. the $(i, j)^{th}$ entry is calculated by iterating over the corpus for a window size of w and the value is the inverse of the distance between the two nodes i and j [9]. Thus, the value in the co-occurrence matrix for two nodes x and y would be higher if the two nodes appear together with a large number of times in the corpus in the given window size.

Using the co-occurrence matrix, the GloVe [22] model is trained to generate embeddings for the nodes in the graph. Initially, the node vector is initialized with random vectors. The object function in equation 1 is optimized using the gradient descent while generating the node embeddings of size 'd' in the GloVe algorithm. The model returns the node list as output with the calculated embeddings for all the nodes in the corpus. The embeddings are then used for detecting communities in the graph using the K-means clustering algorithm. Based on the results mentioned in [9], GloVeNoR algorithm outperforms node2vec [4] and DeepWalk [11] node embedding

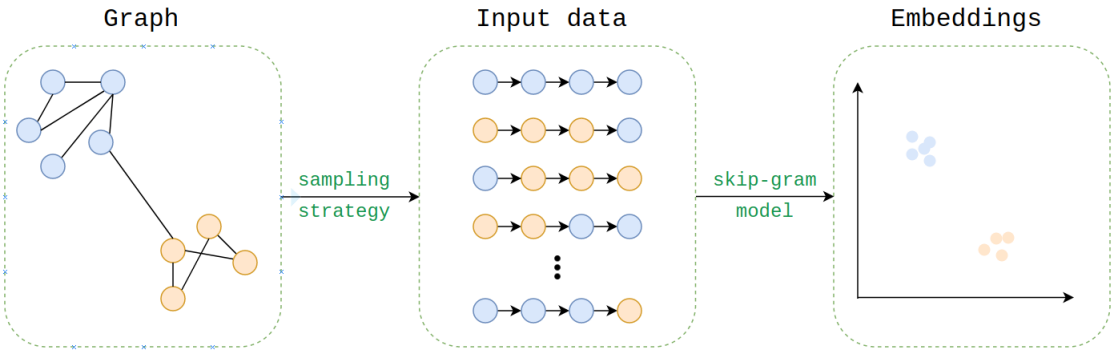

Figure 7: Node Embedding using node2vec [6]

algorithms in calculating the modularity scores.

3.2 Simplex2Vec embeddings for community detection

Simplex2Vec proposes an advanced technique for generating embeddings for the nodes as well as the higher-order structures in a graph while capturing the higher-order interactions from the graph [5]. The resultant embeddings are used as input for the clustering algorithms to perform community detection in the graph. Simplex2Vec incorporates Hasse Diagrams [21] for representing the simplicial complex structure of a graph. The Hasse diagram integrates the simplices of different order into a directed acyclic graph (DAG) structure to preserve the topological structure of the graph.

Using the Hasse diagram of a simplicial complex as the based, the algorithm performs random walks on the Hasse Diagram i.e., the directed acyclic graph and ignores the directions for more flexibility [5]. The results of the random walks act as the corpus for the word2vec model that produces the resultant embeddings. The node embeddings are used as the input data for hierarchical clustering to find the communities in the graph. One major differentiating factor between the existing node embeddings algorithms like node2vec [4] and DeepWalk [11] and Simplex2Vec is, the former algorithms perform random walks on the graph structure while Simplex2Vec performs random walks on the Hasse Diagram thus integrating the information about

the higher-order interactions of the nodes into the walks.

Similar to the previous embedding algorithms like node2vec, DeepWalk, and GloVeNoR, Simplex2Vec uses the word2vec algorithm for computing the embeddings. Since the word2vec algorithm requires a corpus representing the elements for whom the embedding is calculated, Simplex2Vec also uses random walks for creating the corpus. For generating embeddings of higher-order structures in the simplicial complex, the Simplex2Vec algorithm performs random walks on the Hasse diagram of the simplicial complex. Since the random walks are performed on the Hasse diagram, the symbolic sequences incorporate the higher-order strictures as well as preserve the higher-order interactions of the simplicial complex [5]. Simplex2vec limits the order of the simplicial complex to 3 nodes i.e. 2-simplex. Figure 12 depicts a sample random walk on the Hasse diagram.

Simplex2Vec performs experiments using both biased as well as unbiased random walks [5]. For a biased random walk, the random walk selects the next node based on the hyper-parameters and the hyper-parameters ensure that for each time step the selected simplex is of a higher-order than the current node. As illustrated in figure 12, the biased random walk uses the edge weight as the hyper-parameter for selected the next node and the value would be higher for the simplices above the current simplex order. The biased random walk ensures that all the higher-order interactions of the simplicial complex are captured in each of the symbolic sequences [5]. Since the symbolic sequences preserve the higher-order structures, the word2vec model also computes the embeddings for the simplices of higher-order. The resultant embeddings from the word2vec model are used for finding communities in the simplicial complex using the clustering algorithms [5].

Figure 11 summarizes the workflow of the Simplex2Vec algorithm. From the input graph or list of simplices, the algorithm first creates a simplicial complex structure.

The simplicial complex structure is used to compute the Hasse diagram [21] where the nodes comprise of the simplices and the edges denote the interaction between a lower order simplex to a higher-order simplex. The green dotted lines in figure 11(b) represent a sample biased random walk performed on the Hasse diagram. The simplices in the random walk are encoded into symbols to generate a symbolic sequence where each symbol is mapped to a simplex in the Hasse diagram. The symbolic sequences are used as a corpus for the word2vec model [5] that computes the embeddings for the simplices.

Simplex2Vec performs community detection experiments on the datasets consisting of face-to-face interactions between teachers and students in high school and primary school [5]. Based on the graphs and the results obtained on these experiments, the authors infer that inclusion of the higher-order structures in the symbolic sequences for generating the embeddings has a strong influence on differentiating the communities in the two datasets [5]. Thus, incorporating higher-order interactions in the corpus for the word2vec model would produce more accurate embeddings for the simplices.

3.3 Higher-order Link Prediction using Triangle Embeddings

Higher-order Link Prediction using triangle embeddings [2], presents two new approaches for computing triangle embeddings i.e. interactions between three nodes in a graph and uses the embeddings to predict the probability of a set of three nodes would appear together in a simplex in the future. The authors propose to use node2vec [4] embedding algorithm with different mathematical operators to generate triangle embeddings for the nodes comprising a triangle. The second approach comprises computing the embeddings for three nodes in a triangle using the 1-hop subgraphs and embedding the subgraphs using the graph2vec algorithm and graph neural networks

[2]. The input data used for generating the embeddings is in the form of timestamped simplices i.e. a set consisting of the node interacting together at a particular time instance. The simplices are represented as $S_i = \{n_1, n_2, n_3, ..., n_j\}$ where S_i represents the set of nodes interacting in the i^{th} simplex.

In [2], the authors define *closed triangle* as a set of three nodes that appear together in a simplex and *open triangle* for a set of three nodes that appear in pairs in different simplices but never appear in a simplex together. The implementation of both the embedding approaches involves the following three steps: the first step is to list the open and close triangles in the training and test data, the second step involves computing the node and triangle embeddings and the third step predicts the closure of open triangles in the test dataset [2]. The paper limits the higher-order structures up to three nodes [2].

For the approach using the node2vec [4], the link prediction algorithm first computes the node embeddings for each node in the graph. Using the node embeddings, the triangle embeddings for a set of three nodes in the training and test data is computed using mathematical operators like average, Hadamard, weighted l1 and weighted l2. The resultant triangle embeddings act as input for the binary classifier i.e. logistic regression model to train the model using embeddings for training data and predict the probability of triangle closure using the embeddings of the test data [2]. The node2vec algorithm using the graph structure as input for generating random walks that acts as the corpus for the skip-gram model. Figure 7 illustrates the workflow of the node2vec algorithm [4].

For the approach using graph2vec and graph neural networks for computing the triangle embeddings for a set of three nodes directly, the first step is to extract the one-hop subgraphs for the nodes present in the triangle. For extracting the one-hop subgraph, the immediate neighbors of the three nodes in the triangle are first visited

and a fixed number of these neighboring nodes are then randomly extracted, forming a subgraph with the nodes of the triangle [2]. If the number of neighboring nodes exceeds the set limit, then neighboring nodes are randomly sampled and only nodes up to the limit are selected for the subgraph. The limit for a fixed number of nodes is kept to ensure that the graph does not exceed beyond a limit since the nodes could contain a large number of neighboring nodes.

The one-hop subgraphs are extracted for each triangle in the training and test data and act as input for the graph2vec algorithm. The graph2vec algorithm [23] uses the back-propagation algorithm with stochastic gradient descent to compute the triangle embeddings using their corresponding one-hop subgraphs [23]. The output of the graph2vec algorithm comprises a list of vector embeddings for each triangle in the training and test data. Using the triangle embeddings, the binary classifier predicts the probability of triangle closure in the future.

For the approach using graph neural networks, the paper makes use of the Deep Graph Convolutional Neural Network (DGCNN) since it generates the graph objects for the triangle and these graph objects are then used to predict the probability of triangle closures [2]. In DGCNN, the extracted one-hop subgraphs are first labeled to separate the nodes in the triangle from their neighbors. Using the labeled subgraphs, the DGCNN model is trained to obtain the graph objects for training data and test data. Using the graph objects, a graph neural network can be trained using the training graph objects for predicting the probability of triangle closure in the future [2].

The differentiating factor between the node embedding technique used in [2] and the node embedding technique i.e. GloVeNoR [9] that is used in this project is: the node2vec algorithm [4] does not consider the global context of the node in the graph i.e. the algorithm generates embeddings for the nodes based on their local context and

interactions with the neighboring nodes only. Thus, the project tries to understand the impact the global context of a node has on higher-order link prediction.

For subgraph embedding using the graph2vec [23] and the approach used in this project i.e. Simplex2Vec, the differentiating factor [5] is that Simplex2Vec generates the embeddings by taking into account the higher-order interactions of the nodes in the simplicial complex and integrating the higher-order structures in the embedding algorithm corpus. One-hop subgraphs on the other hand only carry information about the local structure and ignore the interaction with higher-order structures in the graph.

CHAPTER 4

Methodology

This chapter describes in detail the problem under consideration and presents a formal definition of the problem. The further sections provide details about the implementation workflow and the later sections discuss in detail each step of the implementation plan along with the respective pseudo-code.

4.1 Higher-order link prediction in temporal graphs

Given a dataset with timestamped simplices representing a graph G = $\{S_i, t_i\}$ where S_i represents the simplex observed at time t_i. S_i is a set consisting of all the nodes $(n_1, n_2, n_3, ...n_j)$ interacting with each other in the i^{th} simplex i.e. $S_i = \{n_1, n_2, ...n_j\}$ [2]. This represents a temporal graph that grows over time and captures all of the higher-order interactions. As described in Chapter 2, higher-order link prediction involves predicting the occurrence of three or more nodes simultaneously in a simplex. For the scope of this project, the problem is narrowed down to predicting the occurrence of three nodes together in a simplex. This subset of three nodes can be referred to as an *open triangle* if the three nodes do not appear together in a simplex but occur in pairs in different simplices. When the group of three nodes appears together in a simplex, the subset can be referred to as *closed triangle* since the three nodes interact with each other at the same time.

The problem can be stated as given a graph G $\{S_i, t_i\}$ with timestamped simplices, predict the occurrence of triangle closure i.e., the probability of given three nodes appearing together in a simplex in the future, and label the triangle as a closed triangle [2]. The implementation is split into four phases - process the input dataset and extract graph information, enumerate the open triangles and close triangles for all the timestamped simplices and split the data for training and testing, generate embeddings for the nodes or the triangle i.e three nodes together, train a binary classification

model to predict the probability of triangle closure.

4.2 Workflow for higher-order link prediction

The implementation workflow for the project comprises multiple stages from data cleansing and preprocessing to generating embeddings and predicting triadic closure. Figure 8 depicts a high-level implementation workflow followed while developing the algorithm. The various modules of the workflow are:

1. The first module involves data preprocessing, which accepts a graph dataset consisting of timestamped simplices as input and constructs the graph using the simplices and extract the node dictionary that maps the nodes with their corresponding labels.

2. The second module aims at generating the labeled data that would be used by the subsequent modules for generating embeddings. This module splits the data for training and testing and enumerates the open and closed triangles i.e. set of three nodes appearing together in a simplex or not.

3. The node embedding approach (Approach 1) uses GloVeNoR [9] module to utilize the graph generated in the first module to generate random walks for each node and use the extracted random walks data as a corpus for generating node embeddings using the GloVe algorithm [22]. Further, the individual node embeddings are combined for creating triangle embeddings, using the embedding operators like Hadamard, Average, Weighted L1 distance, Weighted L2 distance.

4. The subgraph embedding approach (Approach 2) uses the Simplex2Vec algorithm [5] that uses the constructed graph to create a Hasse diagram [20] from the simplicial structure of the graph. The algorithm uses the Hasse Diagram to extract random walks for simplices from $0to2^{nd}$ order i.e. 2-simplex and

using the random walks as corpus, generates embeddings for all of the simplices present in the Hasse diagram.

5. The final step in the implementation is to train the binary classifies and use it to predict the probability of triangle closure for open triangles in the test data. A logistic regression model is trained using the training data for the current implementation.

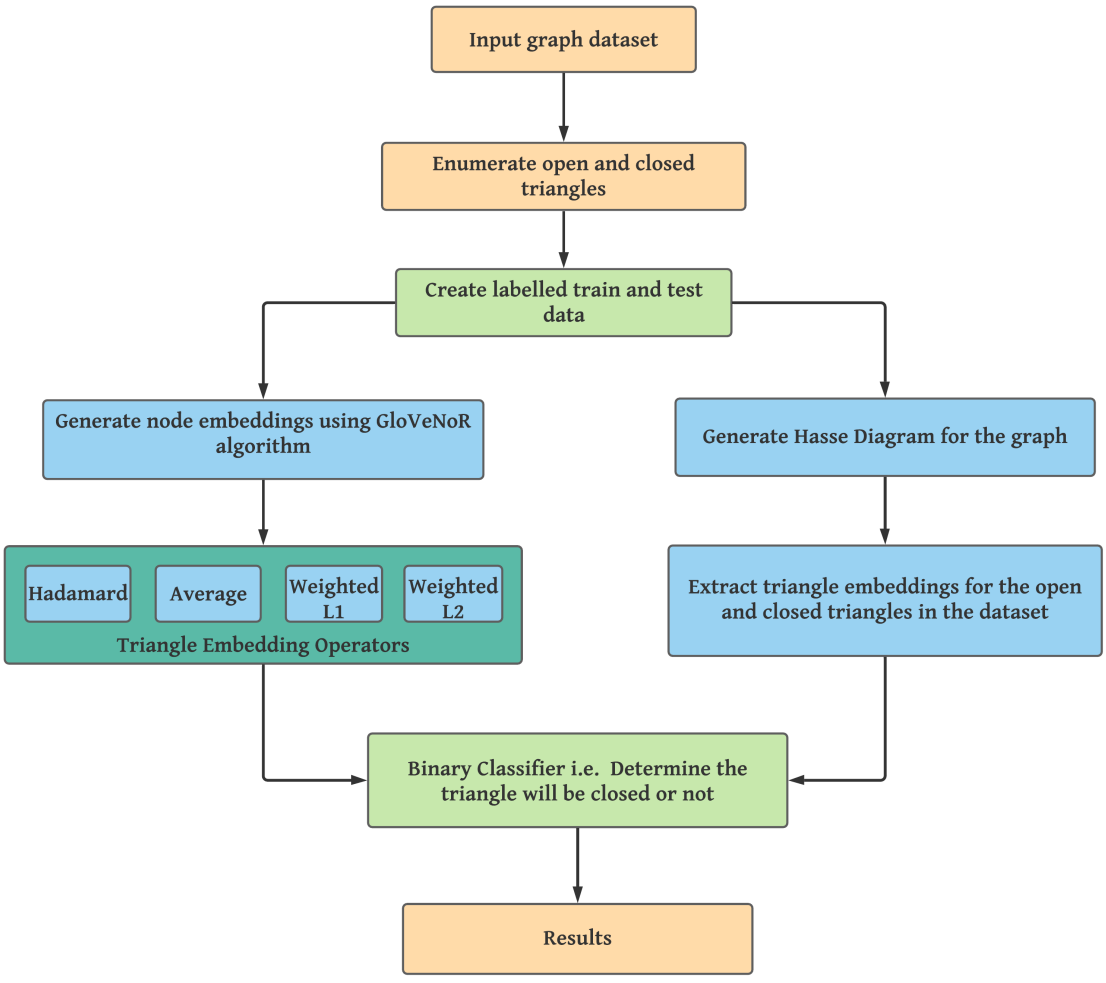

Figure 8: Implementation Workflow

4.3 Data Preprocessing

The graph information for each dataset used in the project consists of three files i.e. timestamp of the simplices, the number of vertices/nodes in each simplex, and the labels of the nodes present in a simplex. To represent, the given information into a graph, the data preprocessing stage iterates over the list of simplices, and for all the pairs of nodes in a simplex, add the nodes to the graph and assigns an edge to the pair of nodes.

The data preprocessing stage generates the graph for the dataset using the information from the simplices. For the graph created using the simplices, it also creates a dictionary that maps the node to its corresponding label. Algorithm 1 describes the implementation used to build the graph using the input dataset. The node dictionary can be extracted from the graph object. The graph object is then exported into a file.

4.4 Enumerating Triangles

Higher-order link prediction for three nodes involves predicting whether a set of three nodes would appear together in a 2-simplex i.e. triangle [1]. Since the datasets contain information of the simplices in a graph occurring at a given time, the data can be used to label the open triangles and the closed triangles i.e. the triangles that appear together in a simplex. Figure 9 depicts the triangle of closure of three nodes in a graph. Initially, none of the nodes in the figure are connected. With the occurrence of timestamped simplex t_1, the nodes 1, 2, 3, 6 of the graph would have an edge between them since they interact in the simplex.

After the timestamped simplex t_3, the nodes 1, 2, 4 of the graph form a triangle where all the edges are connected, however, since the three have never appeared together in a simplex, this triangle is labeled as an open triangle. After the simplex t_4,

Algorithm 1: Generate graph and node dictionary

Function : build_graph (simplices, number_of_vertices)
Input : *simplices*: List of nodes present in each simplex
number_of_vertices: List containing the number of vertices present in each simplex
Output : *graph*: A graph object consisting of the nodes and edges represented by the simplices

```
// Create an empty graph object
graph_object: New Graph Object;
index = 0;
// Iterate over each simplex
for current_number_of_vertices in number_of_vertices do
    // Fetch nodes of the current simplex
    nodes ← simplices[index : current_number_of_vertices];
    index = index + current_number_of_vertices;
    // Add the nodes from current simplex to the graph
    graph ← add_nodes(nodes);
    // Iterate over all pairs of nodes in the current set and
       add the edge in the graph
    foreach combination in current_list_of_vertices do
        graph ← add_edge(node_1, node_2);
    end
end
return graph;
```

the three nodes 1, 2, 4 appeared together in the current simplex and since all the edges between them are connected, the triangle can now be labeled as a closed triangle. As seen in the figure, the datasets used for experiments consist of such triangles that are open initially and are eventually closed. This module focuses on finding such triangles in the dataset and splitting the dataset into training data and test data wherein the open and closed triangles are labeled as 0 (open) and 1 (closed).

4.5 Embedding Algorithms

The embedding algorithms used for generating vectors represent the nodes or the higher-order structures in the graph in a lower-dimensional space. These embedding vectors can be used for applying machine learning algorithms to study graph data.

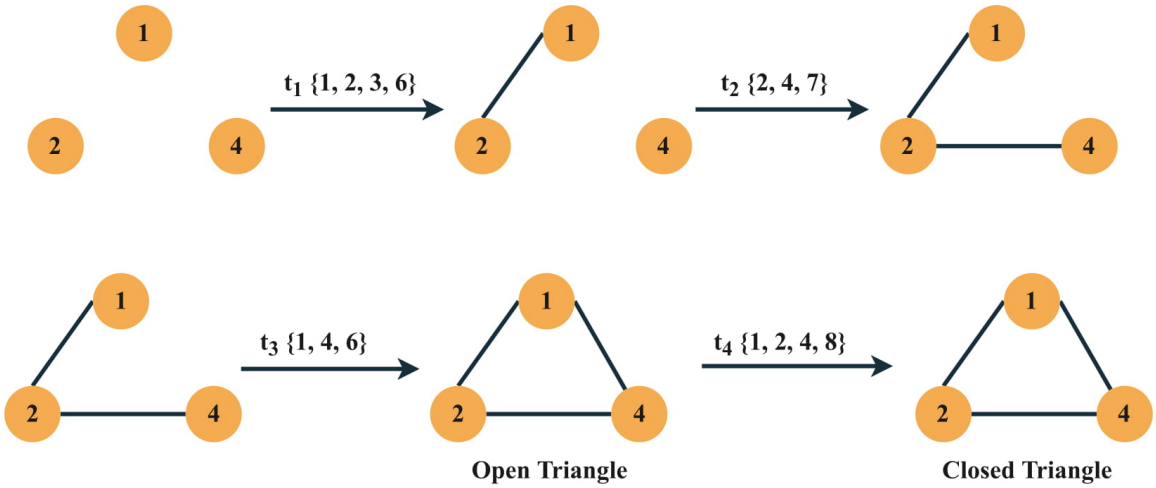

Figure 9: Triadic closure over time [2]

The GloVeNoR algorithm (Approach 1) [9] generates the embeddings for each node in the graph and the Simplex2Vec algorithm (Approach 2) [5] generates embeddings for nodes, pair of nodes, and the triangles in the graph.

4.5.1 Approach 1: Node Embedding using GloVeNoR

The global vectors for node representation (GloVeNoR) [9] algorithm incorporates the local as well as the global context of a node for generating node embeddings. The graph generated in algorithm 1 is used as an input for the algorithm 4 to generate the node embeddings. Using the node embeddings of the individual nodes, the triangles embeddings for each triangle in the dataset are calculated using the different operators like average, Hadamard, weighted L1, weighted L2. The algorithm 5 is used for calculating the triangle embeddings for all the triangles using their respective node embeddings. Figure 10 depicts the process followed while calculating the triangle embeddings using the node embeddings of all the nodes in the triangle.

4.5.2 Approach 2: Subgraph Embedding using Simplex2Vec

The Simplex2Vec algorithm [5] can be used for generating node embeddings as well as embeddings for higher-order simplices by setting the maximum order of the

Algorithm 2: Enumerating Open Triangles {Similar to [2]}

Function : list_open_triangles (graph, simplices, number_of_vertices, edges)

Input : *graph*: The graph object
simplices: List of nodes present in each simplex
number_of_vertices: List containing the number of vertices present in each simplex
edges: List of edges in the graph

Output : *open_triangles_list*: List consisting of set of triangles that are still open

```
// Get triangles that are already closed
closed_triangles ← get_closed_triangles();
open_triangles ← set();
// Iterate over the edges in the graph
for {node_1, node_2} in edges do
    // Iterate over all the other nodes to find triangles
    for node in graph do
        if has_edge(node_1, node) and has_edge(node_2, node) then
            current_triangle ← list(node_1, node_2, node);
            // Check if the triangle is not already closed
            if current_triangle not in closed_triangles then
                open_triangles.add(current_triangle);
            end
        end
    end
end
return open_triangles;
```

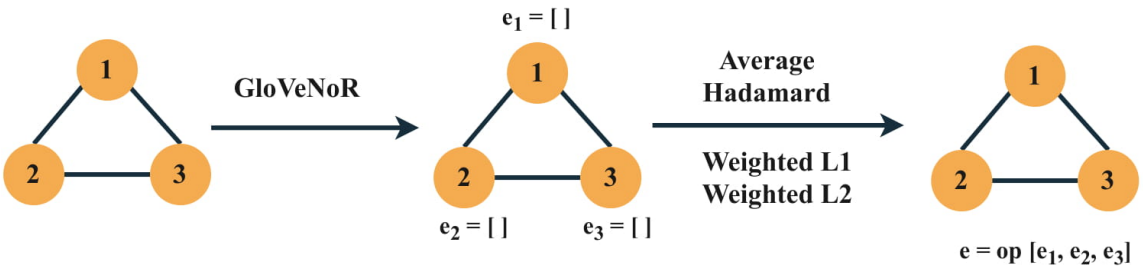

Figure 10: Approach 1: Generating Triangle Embeddings using GloVeNoR [2]

Hasse diagram. For the current implementation since 2-simplex is under consideration, the maximum order of the Hasse diagram is therefore set to 2. Thus, the Simplex2Vec

Algorithm 3: Enumerate New Closures {Similar to [2]}

Function : get_new_closures (graph, old_simplices, old_number_of_vertices, new_simplices, new_number_of_vertices)

Input : *graph*: The graph object
old_simplices: List of nodes present in the old simplices based on timestamp
old_number_of_vertices: List containing the number of vertices present in the old simplices
new_simplices: List of nodes present in the new simplices based on timestamp
new_number_of_vertices: List containing the number of vertices present in the new simplices

Output : *new_triangles_list*: List consisting of set of triangles that were closed in the new simplices

```
// Get triangles that are already closed
closed_triangles ← get_closed_triangles();
unique_nodes ← {set of nodes from old simplices} ;
new_triangles ← set();
current_index ← 0;
// Iterate over the new simplices
for number_of_vertices in new_number_of_vertices do
    current_simplex ← new_simplices[current_index :
     number_of_vertices];
    current_index = current_index + number_of_vertices;
    if number_of_vertices >= 3 then
        foreach combination of 3 nodes in current_simplex do
            if all nodes are unique and triangle not already closed then
                new_triangles_list.add(current_combination);
            end
        end
    end
end
return new_triangles_list;
```

approach generates embeddings for 0-simplex (node), 1-simplex(pair of nodes), 2-simplex(triangles i.e. 3 nodes together). The algorithm 6 is used for generating the embeddings with the netwrokx graph object build in algorithm 1 as input.

For the different experiments, the size of walks, the walk length, and the size

Algorithm 4: Approach 1: GloVeNoR Embedding Algorithm {Similar to [9]}

Function : glovenor (G, w, d, l, k, p, q, i)
Input : G: Input graph object
w: Length of context window
d: Size of embeddings
l: Length of random walks
k: Number of random walks for a node
p: In-out hyper-parameters (walk away parameter)
q: Return hyper-parameter
i: Number of training iterations
Output : node_vectors: List of vectors representing the embedding of each node in the graph

```
graph_corpus ⟵ generate_biased_walks(G, k, p, q, l);
cooccurrence_matrix ⟵ build_cooccurr(graph_corpus, w);
node_vectors ⟵ initiate_random_vectors();
train_word2vec_model(cooccurrence_matrix, node_vectors, i);
return node_vectors;
```

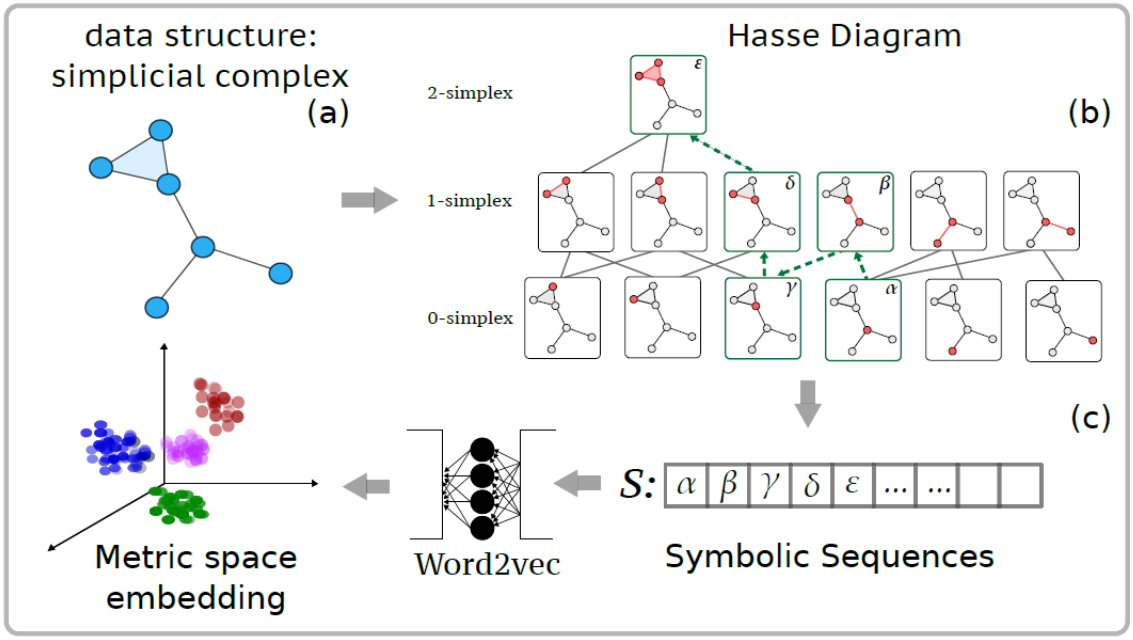

Figure 11: Approach 2: Simplex2Vec algorithm work flow [5]

of embeddings are changed with the values from table 3. The Hasse diagram for each dataset is extracted and save to avoid recomputing the Hasse diagram for the

Algorithm 5: Approach 1: Compute triangle embeddings using GloVeNoR

Function : glovenor_triangle_embeddings(graph, triangles, node_dictionary, dimension, number_of_iterations, operator)

Input : *graph*: The graph object
triangles: List of triangles i.e. sub graph of three nodes
node_dictionary: Dictionary that maps the nodes to their labels
dimension: Size of the embeddings/vectors
number_of_iterations: Number of iterations while training the GloVe model for generating embeddings
operator: Operator to be used for triangle embedding

Output : *triangle_embedding_list*: List consisting of embeddings for each triangle in the triangles list

```
random_walks ← generate_random_walk(graph);
cooccurrence_matrix ← generate_cooccurrence_matrix(graph, random_walks);
node_embeddings ← glove_model(graph, cooccurrence_matrix);
triangle_embedding_list ← [];
for set(x, y, z) in triangles do
    x_vec ← node_embeddings(x);
    y_vec ← node_embeddings(y);
    z_vec ← node_embeddings(z);
    if operator == "Average" then
        value = ((x_vec + y_vec + z_vec)/3);
        triangle_embedding_list.append(value);
    end
    else if operator == "Hadamard" then
        value = (x_vec * y_vec * z_vec);
        triangle_embedding_list.append(value);
    end
    else if operator == "L1" then
        value = (abs(x_vec − y_vec) + abs(y_vec − z_vec) + abs(z_vec − x_vec))/3;
        triangle_embedding_list.append(value);
    end
    else if operator == "L2" then
        value = (abs(x_vec − y_vec)^2 + abs(y_vec − z_vec)^2 + abs(z_vec − x_vec)^2)/3;
        triangle_embedding_list.append(value);
    end
end
return triangle_embedding_list;
```

Algorithm 6: Approach 2: Generate Embeddings using Simplex2Vec

Function : Simplex2Vec (G, max_order, num_of_walks, walk_length, size_of_embeddings)

Input : G: Networkx object of the input graph
max_order: Maximum simplex order up to which the hasse diagram must be constructed
num_of_walks: Number of random walks to be performed for each node in the Hasse Diagram
walk_length: Length of the random walk performed on the Hasse Diagram
size_of_embeddings: Dimension of the embeddings to be computed by the word2vec model

Output : embedding_list: Map consisting of the key representing the simplex and a vector containing the embedding of the corresponding simplex

```
// Generate Hasse Diagram from graph object
hasse_diagram ⟵ generate_hasse_diagram(G);
walks ⟵ compute_random_walks(G, hasse_diagram, num_of_walks, walk_length);
model ⟵ word2vec(walks, size_of_embeddings);
// Training the word2vec model
model.fit();
embedding_list ⟵ model.vectors();
return embedding_list;
```

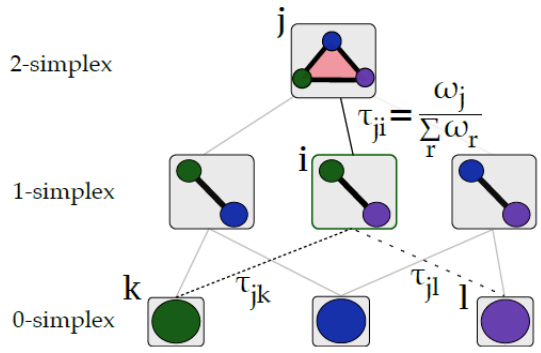

Figure 12: Approach 2: Random Walks on Simplicial Complexes [5]

experiments. For all the following experiments, the Hasse Diagram is read as input instead of constructing it again thus reducing the time required by the algorithm. The

embeddings for each triangle in the training and test data are fetched using the labels of the triangles and used for higher-order link prediction.

4.6 Triangle Closure Prediction

For predicting the triangles that would eventually close in the graph, a logistic regression binary classification model is used. The model is used to predict the probability of whether a given set of three nodes would form a closed triangle denoted with the label 1 or would remain open denoted by 0. The binary classifier predicts the label for the triangles that are open in the training data. The logistic regression model is trained using the triangle embeddings for the triangles listed while generating the labeled dataset. Algorithm 7 describes the implementation for training and classifying the triangles using the logistic regression model. The accuracy of the model can be examined by comparing the actual labels with the labels predicted by the model for test data.

For training the logistic regression model and for testing the trained model, the timestamped simplices in the dataset are split into training and test data. The data is split using the timestamp of the simplices. The logic for splitting the data into train and test as well as labeling the triangles are open or closed is based on the approach followed in [2]. For training data, the open triangles from 0 to 60 percent of the dataset based on the timestamp are considered. The triangles are labeled based on whether they get closed or remain open in the data ranging from 60 to 80 percent. If the triangle is closed in the 60 to 80 percent time frame, then it is labeled as one i.e. closed else it is labeled zero i.e. open. A similar approach is followed for test data where the triangles are labeled if they go through closure in the 80 to 100 percentile timestamps.

Algorithm 7: Predicting Triangle Closure

Function : classify_data (dataset_name, operator_type, solver)

Input : *dataset_name*: Name of dataset to read labelled data
operator_type: Type of operator used for triangle embedding *(only for GloVeNoR)*
solver: Type of solver to be used in logistic regression model

Output : *predicted_labels*: Prediction labels for the testing data.

```
train_triangles, train_labels ← read_labelled_data(dataset_name, range);
test_triangles, test_labels ← read_labelled_data(dataset_name, range);
// Get triangle embeddings for training and test data
x_train ← feature_matrix(train_triangles, operator_type);
x_test ← feature_matrix(test_triangles, operator_type);
// Initialize logistic regression model
model ← logistic_regression();
// Train model using training data
model.fit(train_triangles, train_labels);
// Predict labels for test data
predicted_labels ← model.predict(x_test);
// Compute accuracy of the model by comparing predicted labels
   with actual labels
average_precision_score(test_labels, predicted_labels);
return predicted_labels
```

CHAPTER 5

Datasets and Experiments

This chapter gives information about the dataset used for the approaches mentioned in the previous chapters as well as discusses the different experiments that were carried out while examining the accuracy of the proposed approaches. The chapter discusses, in brief, the evaluation metric that is used for comparing the performance of the proposed approach using the results of the previous experiments.

5.1 Datasets

The datasets used for the experiments in this project consist of graph data representing the interactions between different entities that are defined using timestamped simplices [1]. The timestamped simplices consist of a set of nodes from the graph that interact together at a given time instance. For example, the dataset contact-primary-school contains data of the students in contact with each other in the primary school. The nodes in the dataset represent the students and the timestamped simplices give information about the students interacting in a given time duration [1].

Table 1 provides information about the number of nodes, edges, and the number of timestamped simplices available for each of the datasets under consideration for this project. Each dataset consists of the following three files:

- *dataset-name-nverts.txt*: Contains a list of integers where each integer represents the number of nodes within a simplex.
- *dataset-name-simplices.txt*: The file contains a contiguous list of integers that represent the nodes present in the simplices [1].
- *dataset-name-times.txt*: The list of integers in the file represents the timestamp of the simplices present in the *dataset-name-nverts.txt* file.

The datasets used for the experiments from various domains are:

- **Email networks:** The datasets Email-Eu [1] [24] [25] and Email-Enron [1] are

networks that represent the email communication between people. The nodes in the graph are the email addresses of the people and the simplices of the graph represent the sender as well as the multiple recipients for an email sent at a particular time. Thus the simplices represent the communication between a sender and multiple recipients via email [1].

- **School Interaction Networks:** The datasets contact-primary-school [1] [26] and contact-high-school [1] [27] represent the interactions between the people at school that was recorded using sensors. The simplices in the dataset represent the interactions between people in a time frame of 20 seconds, recorded using the sensors. The nodes in the graph represent the people at the school.
- **National Drug Code (NDC) Directory Networks:** The datasets NDC-classes [1] and NDC-substances [1] represent the class labels and substances used to make the drugs as the nodes. The simplices in NDC-classes dataset represents the drug and the class labels represent the nodes that are applied for a particular drug [1]. For NDC-substances, the simplices represent the drug and the nodes in a simplex are the substances that are used to prepare the drug [1].
- **Question Tags Network:** The datasets tags-ask-ubuntu [1] and tags-math-sx [1] represent the relationship between the tags and the questions asked on the forum. The simplices in both the datasets represent questions and the nodes in a simplex represent the tags associated with the question [1]. The simplices thus provide information about the number of nodes i.e. tags that co-appear for different questions i.e simplices.

5.2 Experiments

The results in Chapter 6 are based on the different experiments that were performed using the embedding algorithms and the binary classifier. The experiments

Dataset Name	Number of Nodes	Number of Edges	Number of simplices
email-Enron	143	1,800	10,883
email-Eu	998	29,299	234,760
contact-primary-school	242	8,317	106,879
contact-high-school	327	5,818	172,035
NDC-classes	1,161	6,222	49,724
NDC-substances	5,311	88,268	112,405
tags-ask-ubuntu	3,029	132,703	271,233
tags-math-sx	1,629	91,685	822,059

Table 1: Information about datasets [1] [2]

differ in the size of embeddings that are generated using the embeddings algorithms, tweaking the parameters of the embedding algorithms used for reading the graph data, the solvers used in label prediction, and the operators used for generating triangle embeddings.

Table 2 and 3 present the different parameters that are used for the experiments performed using Approach 1 i.e. GloVeNoR and Approach 2 i.e. Simplex2Vec embedding algorithms. The experiments were performed for all the possible combinations from the table 2 and 3 for majority of the datasets. The objective behind performing multiple experiments using different parameters was to understand the impact of each of the parameters on the result. Performing the experiments on large datasets was computationally expensive thus, the experiments on large datasets were performed using the best performing parameters on the small dataset.

Parameter name	Values for the parameter
Number of Iterations	10, 20, 50, 100, 200
Size of Embeddings	10, 20, 32, 50, 64, 128, 256, 512, 1024, 2048
Triangle Embedding Operator	Average, Hadamard, Weighted L1, Weighted L2
Solver (Logistic Regression)	liblinear, newton-cg, lbfgs, sag, saga

Table 2: Parameters for experiments using Approach 1 (refer 4.5.1)

Parameter name	Values for the parameter
Number of Random Walks	5, 10, 20
Length of Random Walk	10, 20, 32, 48
Size of Embeddings	10, 32, 50, 64, 128, 200, 256, 512, 1024, 2048
Solver (Logistic Regression)	liblinear, newton-cg, lbfgs, sag, saga

Table 3: Parameters for experiments using Approach 2 (refer 4.5.2)

5.3 Evaluation Metric

To compare and contrast the results of the project with the results in [2] and [1], the evaluation metric used is similar to the metric specified in [2]. Area under the precision-recall curve (AUC-PR) relative to the random baseline [1] [2] is used for evaluating the accuracy of the predicted data. The evaluation metric gives information about the number of triangles that would undergo closure, which was correctly predicted by the classifier. The random baseline for the dataset is calculated using the number of triangles that undergo closure in test data divided by the total number of triangles in the test data. The formula for calculating random baseline is:

$$\text{random\_baseline} = \frac{\text{number of triangles that underwent closure in test data}}{\text{total number of triangles in test data}}$$

The formula for calculating the performance of the model on a given dataset is [2]:

$$\text{performance} = \frac{\text{AUC-PR}}{\text{random\_baseline}}$$

5.4 Experimental Setup

The algorithms mentioned in chapter 4 are implemented in Python. The automation script used to perform all the steps from data preprocessing to triangle closure prediction along with all the different experimental parameters is implemented in Python. The python libraries used for the implementation are:

- Graph Processing: networkx, python-igraph
- Generating embeddings: numpy, pandas, ray, pickle, joblib, networkx

- Label Prediction: numpy, pickle, sklearn

The experiments on large datasets were performed using AWS Deep Learning AMI using 36 vCPUs and 75 Gb memory. Initial testing while preparing the automation script and testing the implementation of the project was done on a local machine with 16 Gb memory.

CHAPTER 6

Inference and Results

This chapter presents the results for the different experiments conducted using the two approaches [5]. The experiments were performed on several real-life datasets from different domains discussed in section 5.1. The results from approach 1 (GloVeNoR node embedding) are compared with the results of the node2vec embedding algorithm from [2]. And the results for approach 2 (Simplex2Vec subgraph embedding) are compared with the results obtained for the graph2vec and graph neural network approaches from [2].

This chapter is divided into three sections: the first section discusses the results obtained for higher-order link prediction using the first approach i.e., node embedding using the GloVeNoR algorithm. The second section provides results of using the second approach i.e., subgraph embedding using the Simplex2Vec algorithm. The third section compares and contrasts the results from the previously conducted experiments with the results of the current two approaches. The metric used for analyzing throughout this section is the area under the precision-recall curve (AUC-PR) relative to the random baseline (refer section 5.3) [2]. The evaluation metric in the graphs and tables is referred to as **Performance**.

6.1 Results for Approach 1 (Node Embedding)

For the first approach using the node embedding algorithm GloVeNoR, many experiments were performed by varying the size of the embeddings, using different operators for calculating the triangle embeddings, and using the different number of iterations for training the word2vec model used for computing the resultant embeddings. Table 4 presents the best results obtained for all the datasets used for higher-order link prediction using approach 1 (section 4.5.1). The binary classifier used for predicting the probability of triangle closure is the logistic regression model with a maximum of

1000 iterations used for converging the solver [2].

Dataset name	Result (AUC-PR relative to random baseline)
email-Enron	4
email-Eu	1.7
contact-primary-school	1.5
contact-high-school	1.9
NDC-classes	1.6
NDC-substances	1.2
tags-math-sx	1.45
tags-ask-ubuntu	4.2

Table 4: Best results for prediction using Approach 1 (GloVeNoR Node Embedding)

Table 5 presents the best results obtained for the different datasets while varying the size of the node embeddings computed using the GloVeNoR algorithm. The binary classifier performs better with a lower number of embeddings when the number of simplices in the dataset is less. This can be attributed to the fact that with a lower number of simplices in the dataset, the number of triangles would be less thus, increasing the size of embeddings would not provide a more global context since the number of interacting triangles would be less.

Dataset Name	Size of vector embeddings									
	10	**20**	**32**	**50**	**64**	**128**	**256**	**512**	**1024**	**2048**
email-Enron	1.5	3	2.91	**4**	3.05	3.1	3.4	1.8	1.73	1.6
email-Eu	1.57	1.6	1.41	**1.7**	1.4	1.46	1.54	1.47	1.38	1.34
contact-primary-school	1.18	1.44	1.35	**1.5**	1.34	1.38	1.27	1.35	1.25	1.28
contact-high-school	1.45	**1.9**	1.48	1.44	1.66	1.65	1.55	1.41	1.25	1.35
NDC-classes	**1.6**	1.3	1.34	1.32	1.24	1.39	1.42	1.23	1.19	1.21
NDC-substances	1.07	1.09	1.18	**1.2**	1.15	1.08	1.21	1.16	1.13	1.12
tags-math-sx	1.31	1.36	1.38	1.3	1.4	**1.45**	1.37	1.31	1.39	1.32
tags-ask-ubuntu	1.26	1.2	1.37	1.4	1.35	1.43	1.34	1.28	2.6	**4.2**

Table 5: Results (AUC-PR relative to random baseline) for different embedding vector sizes using Approach 1 (GloVeNoR Node Embedding)

Table 6 lists the results obtained while training the word2vec model with varying

number of iterations. Based on the table it can be seen that with an increase in the number of training iterations, there is an increase in the performance of the algorithm for all the datasets. Increasing the training iterations for the word2vec model would mean that the model converges more and compute more accurate embeddings for the nodes with lesser context data. Also, increasing the iterations would mean that the embeddings generating using the model represent the node embeddings better than the embeddings generated with a less trained model. Thus the number of training iterations is a parameter that has a direct impact on the performance of the approach.

Dataset Name	**Number of training iterations**				
	10	**20**	**50**	**100**	**200**
email-Enron	2.08	1.81	3	**4**	2.52
email-Eu	1.28	1.42	1.48	1.54	**1.7**
contact-primary-school	1.44	1.31	1.35	**1.5**	1.35
contact-high-school	1.64	1.62	1.47	1.47	**1.9**
NDC-classes	1.37	1.42	1.28	1.31	**1.6**
NDC-substances	1.08	1.09	1.16	**1.2**	1.18

Table 6: Results (AUC-PR relative to random baseline) for different number of training iterations using Approach 1 (GloVeNoR Node Embedding)

Table 7 outlines the performance of the triangle embedding operators used in Approach 1 for representing the triangles in the graph using vectors from the individual node embeddings. Based on the results, it can be inferred that the average and Hadamard operators outperform the other operators for the task of higher-order link prediction. However, a single operator cannot be narrowed down to have the highest impact since the performance of all the operators for the large datasets seems to be near equal suggesting that the operators do not have a major impact on the performance of the algorithm.

Dataset Name	Triangle Embedding Operators			
	Average	**Hadamard**	**Weighted L1**	**Weighted L2**
email-Enron	**4**	3	1.67	1.7
email-Eu	**1.7**	1.22	1.46	1.47
contact-primary-school	**1.5**	1.33	1.35	1.3
contact-high-school	1.64	1.65	**1.9**	1.57
NDC-classes	1.37	**1.6**	1.3	1.25
NDC-substances	**1.2**	1.07	1.05	1.05

Table 7: Results (AUC-PR relative to random baseline) for different triangle embedding operators using Approach 1 (GloVeNoR Node Embedding)

6.2 Results for Approach 2 (Subgraph Embedding)

For approach 2 i.e., the subgraph embedding algorithm, table 8 summarizes the best results obtained for higher-order link prediction using the logistic regression model as the binary classifier. The results depict that the algorithm performs consistently on all the datasets thus implying that the performance of the algorithm is not affected by the size of the dataset and can be used for both small as well as large datasets. This highlights that incorporating higher-order interaction of the graph into the embeddings helps in computing embeddings that could perform better for predicting the other higher-order interactions.

Dataset name	Result (AUC-PR relative to random baseline)
email-Enron	2.5
email-Eu	2.2
contact-primary-school	2.7
contact-high-school	2.5
NDC-classes	2

Table 8: Best results for prediction using Approach 2 (Simplex2Vec subgraph Embedding)

Table 9 presents the performance of the model for varying size of the embeddings vectors. Based on the table, the results of the algorithm seem to be better for medium-

sized embedding vectors. The results for higher embeddings vectors are competitive however, for very small embedding size, the results are very poor for all the datasets. One reason for lower size embeddings performing poorly could be accounted for the fact that with small embeddings, the higher-order structure information that would be integrated into the embeddings is far less as compared to the medium-sized embeddings.

Dataset Name	**Size of vector embedding**									
	10	**32**	**50**	**64**	**128**	**200**	**256**	**512**	**1024**	**2048**
email-Enron	1.79	2	**2.5**	1.97	1.88	2.14	1.78	1.34	1.51	1.54
contact-primary-school	1.57	**2.7**	2.12	2.22	1.73	2.14	2.5	2.09	2.1	2.23
contact-high-school	1.18	1.76	1.71	1.73	2.03	**2.5**	1.7	2.21	1.81	1.78
NDC-classes	1.45	1.63	1.76	1.76	1.88	1.91	**2**	1.75	1.65	1.6

Table 9: Results (AUC-PR relative to random baseline) for different embedding vector sizes using Approach 2 (Simplex2Vec subgraph Embedding)

The random walks in approach 2 are performed on the Hasse Diagram created for the input dataset using the simplicial complex structure of the graph. The table 10 presents the results for the performance of the subgraph embeddings approach while varying the number of random walks i.e. the number of symbolic sequences used for computing the embeddings. As seen from the table, it can be concluded that increasing the number of random walks helps to improve the performance of the model. By increasing the number of random walks performed on the Hasse diagram, the corpus used as input for the word2vec model becomes richer with the information about the higher-order interactions. Thus, more random walks provide better context for each node and the triangles in the graph, resulting in the computation of optimal embeddings.

The length of the random walk specifies how big the symbolic sequence would be for a simplex in the graph. The table 11 represents the results obtained for the

Dataset Name	Number of random walks		
	5	**10**	**20**
email-Enron	1.46	2.23	**2.5**
email-Eu	2.07	2.1	**2.2**
contact-primary-school	1.65	1.9	**2.7**
contact-high-school	1.31	1.72	**2.5**
NDC-classes	1.61	1.9	**2**

Table 10: Results (AUC-PR relative to random baseline) for different number of random walks using Approach 2 (Simplex2Vec subgraph Embedding)

different datasets while varying the length of the random walk. The assumption here is that with a larger random walk length, the symbolic sequence would incorporate a higher number of interactions in a single context window. Based on the results, it can be seen that while a bigger length of random walk performs better, the length of the random walk would perform better for an optimal value. For a small dataset, the number of nodes in the Hasse diagram would be low thus resulting in duplication of the symbolic sequence.

Dataset Name	Length of random walk			
	10	**20**	**32**	**48**
email-Enron	1.82	1.87	2.14	**2.5**
email-Eu	2.13	2.12	**2.2**	2.1
contact-primary-school	1.6	2.12	**2.7**	2.5
contact-high-school	1.42	2.21	**2.5**	2.1
NDC-classes	1.8	1.91	**2**	1.85

Table 11: Results (AUC-PR relative to random baseline) for different lengths of random walks using Approach 2 (Simplex2Vec subgraph Embedding)

6.3 Results using different binary classifiers

Another experiment performed using the computed node and triangle embeddings was to check the performance of different binary classifiers for the task of higher-order link prediction. The binary classifiers used for performing the comparative study

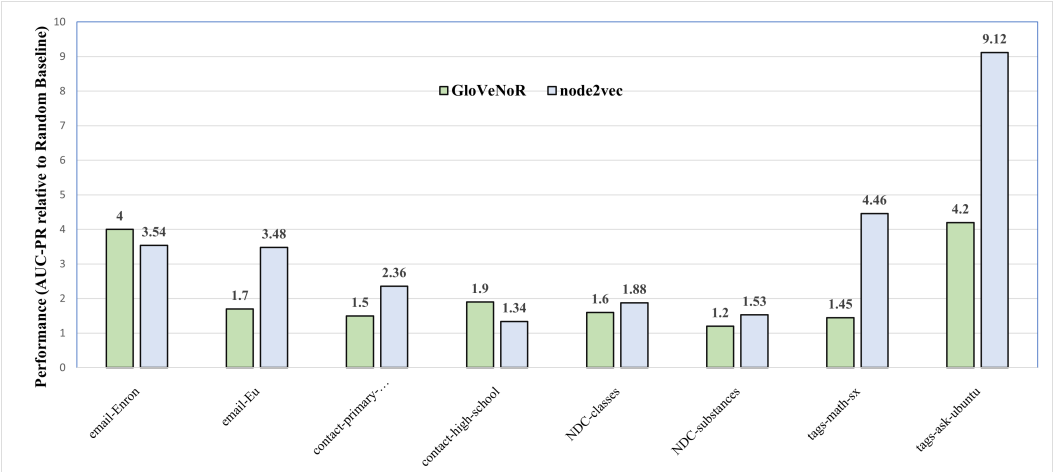

Figure 13: Comparison of the best results obtained using Approach 1 (Node Embedding) with the results of node2vec embedding algorithm in [2]

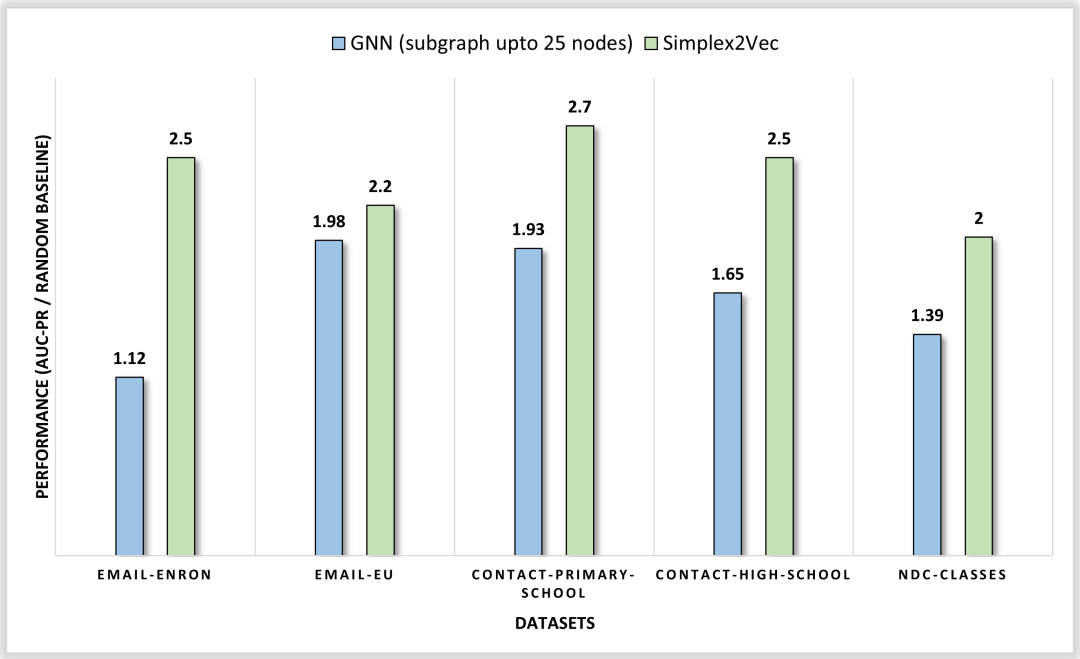

Figure 14: Comparison of the best results obtained using Approach 2 (Subgraph Embedding) with the results of graph neural networks in [2]

are logistic regression, support vector machine (SVM), decision tree classifier, and convolutional neural networks (CNNs). Due to the process being computationally expensive, the experiments were performed on four datasets i.e., email-Enron, contact-

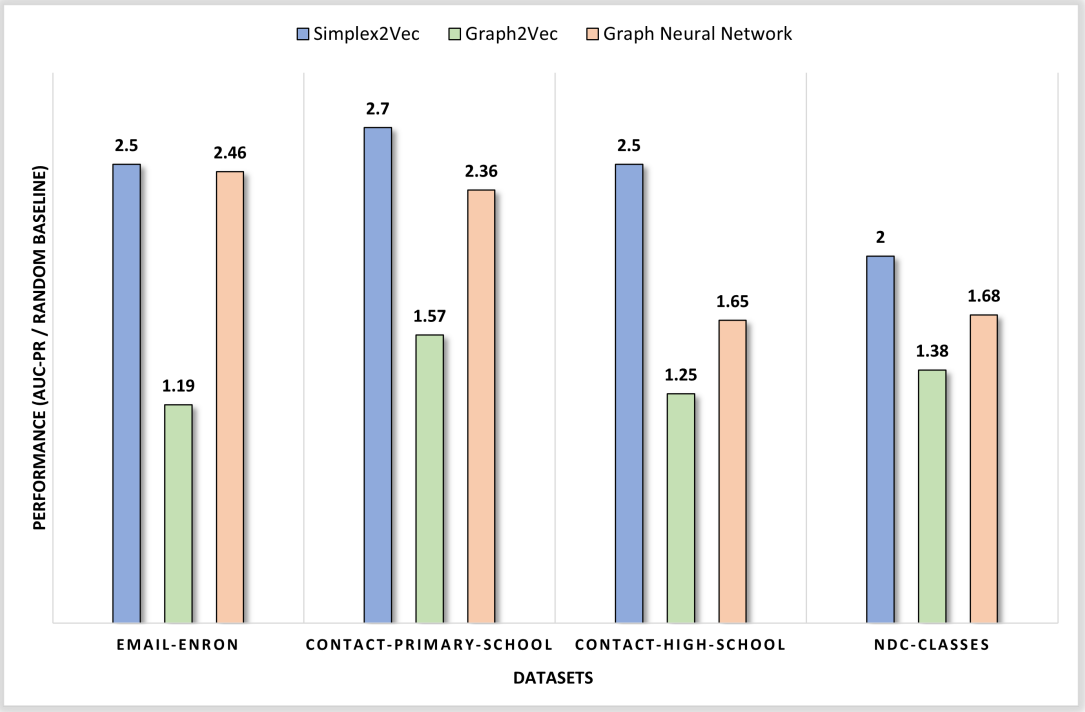

Figure 15: Comparison of the best results obtained using Approach 2 (Subgraph Embedding) with the results of graph2vec and graph neural networks in [2]

high-school, contact-primary-school, and NDC-classes.

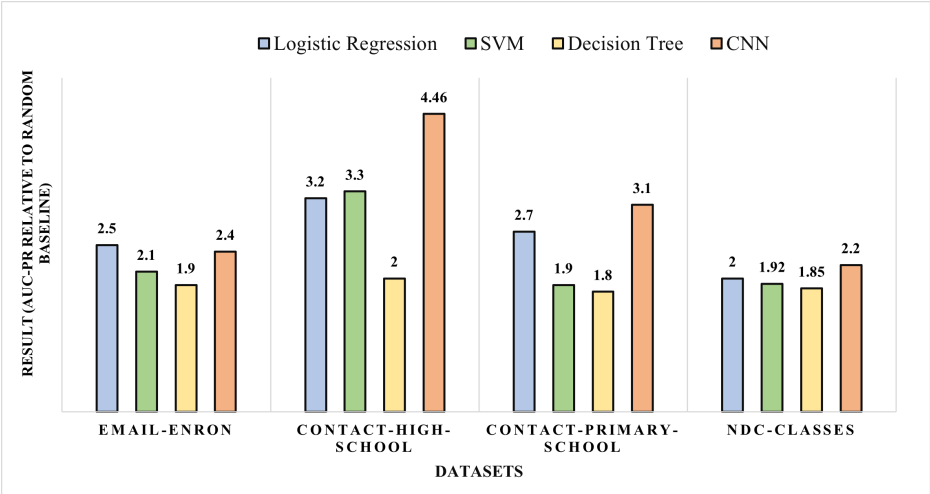

Figure 16: Comparison of the performance of different binary classifiers

Figure 16 depicts the results obtained for the datasets under consideration using

the different binary classification techniques. Based on the graph, it can be seen that the CNNs perform much better as compared to the other classifiers. This can be attributed to the fact that CNNs employ deep learning to extract hidden features from the embeddings and thus have high performance. On the other hand, CNNs take a longer time for training and are computationally expensive. Logistic regression is the next best performer among the other classifiers and is relatively inexpensive to train computationally. Thus there is a trade-off between computation time and the performance of the binary classifiers.

6.4 Comparison with existing results

This section encompasses a comparative study of the results obtained using the experiments performs for the two approaches with the results from the previous experiments from [2]. Approach 1 in the current project involves generating node embeddings using the GloVeNoR model and using the resultant embeddings, calculating the triangle embeddings using the various operators. The results of approach 1 are compared with a similar node embedding technique used in [2]. And the results from approach 2 i.e., subgraph embedding using the Simplex2Vec algorithm are compared with the results of the graph2vec algorithm and graph neural networks from [2].

Figure 13 depicts the comparison of the results obtained using GloVeNoR and the results of the node2vec algorithm. It can be seen that the node2vec algorithm outperforms the GloVeNoR algorithm in some datasets. This attributes that local context contributed more while computing the embeddings for a node as compared to the global context of the node. The GloVeNoR performs better for small datasets since the global context in those datasets would not contribute much to the overall embedding results.

Figure 14 depicts the comparison of the results obtained using Simplex2Vec

subgraph embedding algorithm with the results of the graph neural networks approach from [2]. Based on the chart, it can be seen that approach 2 used in this project outperforms the graph neural networks approach. It can be inferred that the Simplex2Vec incorporates higher-order interactions and higher-order simplices into the random walks thus providing a much better context for the triangles in the train and test dataset.

Further, as seen in figure 15, it can be seen that Simplex2Vec provides higher performance compared to the graph2vec algorithm. One major reason for the Simplex2Vec outperforming the graph2vec algorithm is the graph2vec only considers the immediate neighbors of the nodes in the triangle while computing the embeddings. On the other hand, Simplex2Vec considers the higher-order simplices into the random walks for generating the embeddings thus providing a much better context of the interaction of the triangle inside the graph.

CHAPTER 7

Conclusion and Future Work

7.1 Conclusion

The objective of the object was to understand the impact the global context of a node and higher-order interactions in a graph have on computing the embeddings as well as on the performance of predicting triangle closure. The project proposes two approaches i.e., approach 1 for predicting triangle closure using node embedding algorithm (GloVeNoR) and approach 2 uses subgraph embedding algorithm (Simplex2Vec) for predicting triangle closure. Approach 1 incorporates the global context of a node along with its local context by computing a co-occurrence matrix for computing the node embeddings. On the other hand, the subgraph embedding approach makes use of the Hasse diagram to integrate the higher-order interactions in the graph for computing the embeddings.

The results for the node embedding approach underperform as compared to the previous techniques but provide useful insights into the impact global context has on the performance. The size of embeddings and the higher number of training iterations for the word2vec model could result in a more improved performance from the model. Further, experimenting with different binary classifiers would provide more insights into the embeddings and their impact on higher-order link prediction. The experiments with different triangle embedding operators didn't uncover any hidden insights are the performance was quite consistent throughout the different operators.

The subgraph embedding approach outperformed the results from the previous experiments that attempt to integrate the higher-order interactions. Using Hasse diagram for generating symbolic sequences as the corpus for the word2vec provides the model with the necessary insights about the higher-order interactions and extracts the features of the higher-order structures into the random walks. Achieving a

much-improved performance by increasing the number of random walks and having an optimal walk length have a huge impact on the performance of the computed embeddings. Subgraph embedding using Hasse diagram thus could prove a breakthrough for predicting the higher-order interactions in large networks and could be used in social networks.

7.2 Future Work

Based on the results obtained in the project, it is evident that using Hasse diagrams for performing random walks helps to extract the higher-order interactions from the graph. Though generating Hasse diagram from graphs or simplicial complexes is computationally expensive and time-consuming. The approaches discussed in the project have more room for improvements and would increase the performance for higher-order link prediction. Performing binary classification using neural networks, integrating the current approaches with BERT, experimenting with other classification approaches like random forests, etc could be the next steps to research further. From the embedding perspective, integrating the global context of the node along with the higher-order interactions would help to integrate the more impact features from the network into the embeddings.

LIST OF REFERENCES

[1] A. R. Benson, R. Abebe, M. T. Schaub, A. Jadbabaie, and J. Kleinberg, ''Simplicial closure and higher-order link prediction,'' *Proceedings of the National Academy of Sciences*, vol. 115, no. 48, p. E11221–E11230, Nov 2018. [Online]. Available: http://dx.doi.org/10.1073/pnas.1800683115

[2] N. Chavan and K. Potika, ''Higher-order link prediction using triangle embeddings,'' in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 4535--4544.

[3] M. Zhang, W. Kalies, S. Kelso, and E. Tognoli, ''Topological portraits of multiscale coordination dynamics,'' *Journal of Neuroscience Methods*, vol. 339, p. 108672, 06 2020.

[4] A. Grover and J. Leskovec, ''node2vec: Scalable feature learning for networks,'' *CoRR*, vol. abs/1607.00653, 2016. [Online]. Available: http://arxiv.org/abs/1607.00653

[5] J. C. W. Billings, M. Hu, G. Lerda, A. N. Medvedev, F. Mottes, A. Onicas, A. Santoro, and G. Petri, ''Simplex2vec embeddings for community detection in simplicial complexes,'' Jun 2019. [Online]. Available: https://arxiv.org/abs/1906.09068

[6] E. Cohen, ''node2vec: Embeddings for graph data,'' Apr 2018. [Online]. Available: https://towardsdatascience.com/node2vec-embeddings-for-graph-data-32a866340fef

[7] W. L. Hamilton, R. Ying, and J. Leskovec, ''Representation learning on graphs: Methods and applications,'' *CoRR*, vol. abs/1709.05584, 2017.

[8] Z. Kurtz, ''The vectors of code: On machine learning for software,'' Jun 2019. [Online]. Available: https://insights.sei.cmu.edu/sei_blog/2019/06/vectors-of-code-on-the-foundations-of-machine-learning-for-software.html

[9] S. Kulkarni, J. K. Katariya, and K. Potika, ''Glovenor: Glove for node representations with second order random walks,'' in *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2020, pp. 536--543.

[10] W. L. Hamilton, *Graph Representation Learning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2020. [Online]. Available: https://doi.org/10.2200/S01045ED1V01Y202009AIM046

[11] B. Perozzi, R. Al-Rfou, and S. Skiena, ''Deepwalk,'' *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, 2014. [Online]. Available: http://dx.doi.org/10.1145/2623330.2623732

[12] R. Brochier, A. Guille, and J. Velcin, ''Global Vectors for Node Representations,'' *The World Wide Web Conference on - WWW '19*, 2019. [Online]. Available: http://dx.doi.org/10.1145/3308558.3313595

[13] H. Cho and Y. Yu, ''Link prediction for interdisciplinary collaboration via co-authorship network,'' *Social Network Analysis and Mining*, vol. 8, no. 1, Mar. 2018. [Online]. Available: https://doi.org/10.1007/s13278-018-0501-6

[14] Y. Yang and N. V. Chawla, *Link Prediction: A Primer*. New York, NY: Springer New York, 2018, pp. 1202--1210. [Online]. Available: https://doi.org/10.1007/978-1-4939-7131-2_365

[15] G. Lin, ''Top ranked coding school in nyc, la & online: Codesmith,'' Sep 2019. [Online]. Available: https://codesmith.io/blog/introduction-to-graphs

[16] O. Michail, ''An introduction to temporal graphs: An algorithmic perspective,'' in *Algorithms, Probability, Networks, and Games*. Springer International Publishing, 2015, pp. 308--343. [Online]. Available: https://doi.org/10.1007/978-3-319-24024-4_18

[17] M. Kaul and M. Imaizumi, ''Understanding higher-order structures in evolving graphs: A simplicial complex based kernel estimation approach,'' *arXiv preprint arXiv:2102.03609*, 2021.

[18] M. T. Schaub, A. R. Benson, P. Horn, G. Lippner, and A. Jadbabaie, ''Random walks on simplicial complexes and the normalized hodge 1-laplacian,'' *SIAM Review*, vol. 62, no. 2, pp. 353--391, 2020.

[19] L. Lovász, ''Random walks on graphs: A survey, combinatorics, paul erdos is eighty,'' *Bolyai Soc. Math. Stud.*, vol. 2, 01 1993.

[20] E. W. Weisstein, ''Hasse diagram. From MathWorld---A Wolfram Web Resource,'' last visited on 05/04/2021. [Online]. Available: https://mathworld.wolfram.com/HasseDiagram.html

[21] M. Fattore, ''Hasse diagrams, poset theory and fuzzy poverty measures,'' *Rivista Internazionale di Scienze Sociali*, vol. 116, no. 1, pp. 63--75, 2008. [Online]. Available: http://www.jstor.org/stable/41625201

[22] J. Pennington, R. Socher, and C. D. Manning, ''Glove: Global vectors for word representation,'' in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532--1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[23] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, ‘‘graph2vec: Learning distributed representations of graphs,’’ *CoRR*, vol. abs/1707.05005, 2017. [Online]. Available: http://arxiv.org/abs/1707.05005

[24] J. Leskovec, J. Kleinberg, and C. Faloutsos, ‘‘Graph evolution: Densification and shrinking diameters,’’ *ACM Transactions on Knowledge Discovery from Data*, vol. 1, no. 1, 2007. [Online]. Available: https://doi.org/10.1145/1217299.1217301

[25] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich, ‘‘Local higher-order graph clustering,’’ in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, 2017. [Online]. Available: https://doi.org/10.1145/3097983.3098069

[26] R. Mastrandrea, J. Fournet, and A. Barrat, ‘‘Contact patterns in a high school: A comparison between data collected using wearable sensors, contact diaries and friendship surveys,’’ *PLOS ONE*, vol. 10, no. 9, p. e0136497, 2015. [Online]. Available: https://doi.org/10.1371/journal.pone.0136497

[27] J. Stehlé, N. Voirin, A. Barrat, C. Cattuto, L. Isella, J.-F. Pinton, M. Quaggiotto, W. V. den Broeck, C. Régis, B. Lina, and P. Vanhems, ‘‘High-resolution measurements of face-to-face contact patterns in a primary school,’’ *PLoS ONE*, vol. 6, no. 8, p. e23176, 2011. [Online]. Available: https://doi.org/10.1371/journal.pone.0023176