NLP, Machine Learning & Deep Learning Methods & Applications on Graphs

by

Elika Haji Agha Bozorgi

(Under the Direction of Krzysztof J. Kochut)

Abstract

Graph datasets are important because they allow for efficient analysis of interconnected data. Also, they enable the exploration of relationships between entities, facilitating pattern recognition, decision making, classification, clustering and statistical analysis, especially for large datasets. Graph databases are also well-suited for handling complex data changes with minimal manual input and offer techniques for data integration and sharing. Graphs are widely used across different fields; such as: social network analysis, recommendation systems, fraud detection, supply chain management, bioinformatics and many others. To be able to benefit from the information of the graph datasets, one has to use different analysis methods including using query languages such as Cypher and SQL, using GNNs or employ machine leaning algorithms like community detection, link prediction, or node classification to gain insights. As already mentioned, graphs usually get large in size and as a result,

it is more efficient in many cases to reduce the size of the graph before applying machine learning models on them directly. To reduce the size of the graph, we can use an embedding method. Embedding is the act of translating high dimensional data into low dimensional ones while keeping the important information. After applying the embedding method, we can use the machine learning model to obtain the data we need from the graph. In this research, we focus on graph analysis and how to use information stored in graphs. First, we introduce a novel method for embedding graphs. While the resulted embedding can be used for various purposes, we use it for the link prediction task. Then, we discuss several recent works for embedding graphs, their advantages and disadvantages and finally compare the model we developed to them. Then, we introduce another method for graph analysis in which we use LLMs to generate multi-hop Cypher query and run that query on a Neo4j database.

INDEX WORDS: AI & LLM, Deep Learning, Machine Learning, Natural Language Processing, Graph

NLP, Machine Learning & Deep Learning Methods & Applications on Graphs

by

Elika Haji Agha Bozorgi

B.S., Azad University of Tehran, Iran, 2016

A Dissertation Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2025

NLP, Machine Learning & Deep Learning Methods & Applications on Graphs

by

Elika Haji Agha Bozorgi

Major Professor: Krzysztof J. Kochut

Budak Arpinar
Khaled Rasheed

Electronic Version Approved:

Ron Walcott
Vice Provost for Graduate Education and Dean of the Graduate School
The University of Georgia
August 2025

Acknowledgments

I would like to express my deepest gratitude to the many people who have supported me throughout this doctoral journey.

First and foremost, I am profoundly grateful to my parents, whose unwavering support and encouragement have been the foundation of my academic pursuits. Their belief in me never wavered, even during the most challenging moments of this journey. To my siblings, I extend my heartfelt appreciation for not only their constant support but also for the invaluable mentorship they provided. Their guidance and wisdom have shaped both my personal growth and academic perspective in ways that extend far beyond this dissertation.

I owe immense gratitude to my advisor, Dr. Kochut, whose expertise, patience, and dedication have been instrumental in bringing this work to fruition. Under his guidance, I have grown not only as a researcher but also as a critical thinker. His insights and constructive feedback have been invaluable throughout every stage of this research.

I would also like to thank my committee members, Dr. Rasheed and Dr. Arpinar, for their thoughtful contributions and scholarly input. Their diverse perspectives

and constructive criticism have significantly enriched this work and challenged me to think more deeply about my research.

Special recognition goes to Dr. Arabnia, whose support throughout my entire PhD program has been remarkable. His encouragement and guidance have been a constant source of motivation, and I am deeply appreciative of the time and effort he invested in my academic development.

The completion of this dissertation would not have been possible without the collective support of these remarkable individuals. Their contributions have not only shaped this research but have also profoundly influenced my journey as a scholar and as a person.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A graph is a data structure that consists of vertices (nodes) and edges. A vertex, also called a node, is a point or an object in the graph, and an edge is used to connect two vertices with each other. Graphs serve as powerful frameworks for analyzing and solving challenges involving interconnected elements. By representing interconnected data, graphs can reveal trends, patterns, and relationships that might not be immediately obvious from raw numbers or text. This allows for a better understanding, decision-making, and communication of complex data. They are used in various fields such as: social media, transportation systems and many others. In social media platforms, individuals appear as nodes and their connections—friendships, follows, professional associations—function as connecting lines. This structure enables recommendation algorithms to identify potential new connections based on existing relationship patterns Saadati et al. [2024].
For transportation systems, geographical points (cities, intersections, transit stops)

become nodes, with pathways (roads, routes, transit lines) forming the connections between them. When structured as a graph, navigation systems can efficiently calculate optimal routes between any two locations.

The vast architecture of the World Wide Web itself follows a graph model, where websites exist as nodes, and the hyperlinks connecting them form the network's structure. This representation allows search engines to navigate and understand the relationships between different online resources.

To use the information stored in graphs, we need to analyze them. Graph data analysis makes it easier to understand and interpret information in graphs. On the other hand, graphs usually grow large in size, which makes it inconvenient and impossible to gain information directly from them without using any methods. As a result, it is necessary to utilize efficient ways to take advantage of the information stored in graphs.

In this work, our aim is to introduce new methods for graph analysis. There are different ways for graph analysis; such as using query languages like Cypher or utilizing machine learning models including but not limited to clustering, classification, link prediction and etc. The first two methods, utilizing query languages and using GNNs, do not require any modification to the graph as these methods can be applied directly to the graph. However, in most cases, for using machine learning methods on graphs, we need to reduce the size of the graph. There are various techniques to reduce the size of data in machine learning including embedding, sampling, compression, and data aggregation. Each of these techniques come with advantages and limitations. In this work, we focus on embedding, since

it was a better technique to reduce the size of the graph dataset in our case.

Embedding is a technique to reduce the size of the data by convert the data to a vector space which transfers data from high dimensional space to a lower dimensional space. By embedding, we keep the most important information of the data and get rid of the unnecessary part. There have been previous novel methods on embedding graphs and we introduce a new method which stands out by giving the user more authority.

In addition, we leverage Large Language Models (LLMs) to analyze data in graphs, too. LLMs are AI models which are basically new technologies that copy how humans think to tackle problems, make choices, and come up with new ideas. These systems get smarter by analyzing massive amounts of data and spotting patterns in all that information. Once they have learned these patterns, AI can respond to what you ask or tell it in ways that feel pretty natural and human-like. Also, LLMs have revolutionized many domains in the industry; as they are widely used in chatbots, customer service, virtual assistants, recommendation systems and others. While LLMs are already advanced, they are still struggling in some specific fields. For example, it is challenging for LLMs to answer multi-hop questions. A multi-hop question is a type of question that requires combining information from multiple sources or steps to arrive at the correct answer. Unlike simple fact-based questions that can be answered from a single sentence or passage, multi-hop questions demand reasoning across several pieces of information, often spread across documents, sentences, or knowledge bases. That is because LLMs are not able to always find a path between two nodes of the graphs which

are not connected to each other by an edge. We introduce a new method to solve this challenge.

1.1 Dissertation Organization

The current chapter (chapter 1) discusses the motivation for our work as well as challenges and how we address these challenges. Chapter 2 introduces our novel method for graph embedding. In chapter 3, we summarize recent embedding method of graphs, their advantages and limitations and comparison of our method to these ones. In chapter 4, we introduce a new method for generating multi-hop Cypher queries on graphs using LLMs. We solve two issues in this method: 1. Generating Cypher query using LLMs; which helps users with no to little knowledge of Cypher language to be able to generate them and use them to gain more information about the graph dataset. 2. LLMs have challenges finding a path in multi-hop question answering; in our method we introduce a new way which helps LLM find a path between two nodes in the graph that are not connected directly. Finally, in chapter 5, we discuss the conclusion and the future work.

1.2 Motivations

Graph-structured data is ubiquitous across a wide range of real-world domains, including healthcare, finance, education, and social networks. In these settings,

entities and their complex relationships can be naturally modeled as graphs — enabling more nuanced reasoning, better predictions, and deeper insights. For example, patient-treatment interactions, transaction networks, and knowledge hierarchies all benefit from graph-based representations.

While graph neural networks and embedding techniques have shown promising results in capturing the structure and semantics of such data, there remain open challenges in designing models that are scalable, interpretable, and effective across diverse tasks. At the same time, the rapid advancement of large language models (LLMs) introduces new opportunities for integrating language understanding with structured graph reasoning, particularly in tasks like multi-hop question answering and knowledge-based inference.

This thesis is motivated by the need to bridge foundational methods in graph learning with practical NLP and deep learning techniques that can be applied across domains. Specifically, it presents: (1) a novel method for graph embedding that improves structural representation learning, (2) a comprehensive survey of random walk-based embedding methods, clarifying their theoretical underpinnings and practical trade-offs, and (3) a framework for multi-hop question answering using large language models, leveraging graph structure for complex reasoning.

By addressing both theoretical and applied challenges, this work contributes to the development of general-purpose, graph-aware machine learning techniques that can support applications across multiple sectors — from personalized education and fraud detection to medical knowledge discovery and intelligent search systems.

1.3 Contributions

This thesis makes the following key contributions to the fields of machine learning, natural language processing, and graph representation learning:

A novel graph embedding method that captures both structural proximity and deeperr semantic relationships, improving performance on downstream tasks such as node classification and link prediction across multiple datasets.

A comprehensive survey and taxonomy of random walk-based graph embedding techniques, covering foundational algorithms, theoretical insights, comparative analysis, and practical guidelines for selecting appropriate methods across domains.

An approach for multi-hop question answering over knowledge graphs using large language models (LLMs), combining symbolic graph structure with neural language representations to enable reasoning across multiple relational paths.

Demonstration of the generalizability of the proposed methods across multiple real-world domains, including potential applications in healthcare, finance, and education, showing the adaptability of graph-based models in diverse structured data settings.

Open-source implementations and reproducible experiments to support further research and practical adoption of the methods presented in this thesis.

Chapter 2

Subgraph2vec: A random walk-based algorithm for embedding knowledge graphs

Graph is an important data representation which occurs naturally in the real world applications Goyal and Ferrara [2018]. Therefore, analyzing graphs provides users with better insights in different areas such as anomaly detection Ma et al. [2021], decision making Fan et al. [2023], clustering Tsitsulin et al. [2023], classification Wang et al. [2021] and etc. However, most of these methods require high levels of computational time and space. We can use other ways like embedding to reduce these costs. Knowledge graph (KG) embedding is a technique that aims to achieve the vector representation of a KG. It represents entities and relations of a KG in a low-dimensional space while maintaining the semantic meanings of them.

There are different methods for embedding graphs including random walk-based methods such as node2vec, metapath2vec and regpattern2vec. However, most of these methods bias the walks based on a rigid pattern usually hard-coded in the algorithm. In this work, we introduce *subgraph2vec* for embedding KGs where walks are run inside a user-defined subgraph. We use this embedding for link prediction and prove our method has better performance in most cases in comparison with the previous ones.

Keywords: Representation Learning, Information Engineering, Link Prediction, Deep Learning, Graph Embedding

2.1 Introduction

Knowledge graphs play a crucial role in organizing, understanding, and leveraging information in various domains. Hence, they become increasingly popular in different areas due to their valuable features. For example, they are widely used in recommendation systems in E-commerce to model the relationships between users and items. Next, they enable personalized recommendations by leveraging knowledge about user interactions and item characteristics Huang et al. [2004]. As another example, they are widely used in healthcare to support practitioners for disease diagnosis by knowledge discovery from patients' personal health repositories Tao et al. [2020]. In addition, they provide a structured way of connecting entities with relationships, which creates a network to organize and represent in-

formation. Thus, it is easier to discover relevant information either manually, by navigating through nodes to discover visions, or by using machine learning and AI to make predictions and generate insights into data. In addition, Knowledge graphs facilitate querying and discovery of complex patterns within data Verborgh et al. [2016] and also capture semantic connections which leads to more accurate queries.

Although knowledge graphs offer many benefits, they have several issues. For instance, Knowledge graphs are often incomplete Chen et al. [2022], as they are frequently populated using various external resources which are often incomplete as well. Also, these outside resources have different structure and format which makes integrating data more difficult. In addition, knowledge graphs usually grow in size and complexity which makes them inefficient and therefore special infrastructure and algorithms are needed to make them more scalable Polleres et al. [2023]. This complexity also makes interpreting patterns and drawing insights more challenging and time-consuming. Moreover, building and maintaining knowledge graphs often need domain expertise to ensure the accuracy of the represented data.

A reasonable number of such issues can be addressed using Artificial Intelligence (AI). In fact, different methods of AI can be used for resolving the previous mentioned problems with knowledge graphs as well as: knowledge graph completion Chen et al. [2020], node representation learning Zhang et al. [2021], semantic search Reinanda et al. [2020], question answering Saxena et al. [2020], anomaly detection Ma et al. [2022], quality assessment Huaman [2022], etc. However, ap-

plying AI algorithms on different types of Knowledge graphs requires a significant number of adjustments due to the high number of the dimensions of the input data. In fact, in many cases, these methods cannot be applied on Knowledge graphs directly. Therefore, it is desired to reduce the number of the dimensions of the input data or knowledge graph by embedding it with different AI methods. Knowledge graph embedding is the act of translating the high-dimensional data to a low-dimensional space, while trying to maintain the semantic meanings of the KG elements. The embedding of each element in the dataset is a unique vector-representation of that element. The resulted embedding of the Knowledge graph can be used for different purposes, such as link prediction, entity classification, semantic search, and others. There exist different types of embedding methods for Knowledge graphs based on supervision, which include *supervised* methods such as Graph Convolutional Networks (GCNs) Yao et al. [2019], *Unsupervised* methods like TransE Bordes et al. [2013] and DeepWalk and *Hybrid* ones such as GraphSAGE Hamilton et al. [2017]. In this paper, we introduce an unsupervised algorithm based on random walks for embedding Knowledge graphs. There are previous random walk-based methods for embedding knowledge graphs, such as node2vec, metapath2vec and regpattern2vec. However, these methods come with challenges. For example, in node2vec the random walks are biased to highly visible types of nodes, the ones with a dominant number of paths. On the other hand, in metapath2vec and regpattern2vec the walks are biased by a series of relationships (or node types) or a fixed regular pattern of relationships (or node types), respectively. In our method, the user enters an arbitrary pattern which

defines a schema subgraph in the actual knowledge graph and the walk is done within this subgraph. In the next section, we will compare our method with the previous related ones.

2.2 Preliminaries

In this section, we will explain some primitive concepts which are fundamental to the understanding of our method and the previous ones, starting with the explanation of the Knowledge graphs (or heterogeneous networks).

Knowledge graph: A knowledge graph is a data set that represents real-world facts and semantic relationships in the form of triplets, where the triplets are represented as a graph with edges as relations and nodes as entities Bordes et al. [2011]. Mathematically, consider $G = (V, E)$ where G represents the knowledge graph and V are the nodes or entities and E represent the relations.

Walk: A walk is a finite sequence of edges which join a sequence of vertices. In $G = (V, E)$ where G is a knowledge graph and V represents the nodes and E represents the edges, a finite walk is a sequence of edges $(e_1, e_2, ..., e_{n-1})$ for which there is a sequence of vertices $(v_1, v_2, ..., v_n)$ such that $\phi(e_i) = (v_i, v_{i+1})$ for $i = 1, 2, ..., n-1$. $(v_1, v_2, ..., v_n)$ is the vertex sequence of the walk. The walk is closed if $v_1 = v_n$, and it is open otherwise Wikipedia contributors [2023].

Path: A path is a walk on a graph where the vertices are repeated.

Subgraph: Graph $S = (V_S, E_S)$ is considered as a subgraph of $G = (V_G, E_G)$ if

and only if its vertex set (V_S) and edge set (E_S) are subsets of those of G. In other words: $V_S \in V_G$ and $E_S \in E_G$.

Schema: Given a knowledge graph G, an edge with a relation type R connects source nodes of type S and target nodes of type T defines a meta edge $S \xrightarrow{R} T$. A schema graph (aka known as meta-template) for G is a set of all such meta edges. In fact, a schema graph is a directed graph defined over node types T, with edges from R, denoted as $G_S = (T, R)$ Sipser [2012].

Heterogeneous Network Yang et al. [2018]: or HIN, is a graph denoted as $G = (V, E, T)$, where each $v \in V$ and $e \in E$ has a mapping function $\Phi(V) = V \rightarrow T_v$ and $\phi(E) = E \rightarrow T_e$ and T_v and T_e denote sets of node and relation types where $|T_v| + |T_e| > 2$. In simple words, in these networks, nodes and edges can belong to different types, and the connections between nodes can have various semantic meanings.

2.3 Related works

DeepWalk Perozzi et al. [2014], is a graph embedding method which aims to learn continuous representations of nodes in a graph. For each walk, it begins by generating the walks from a random starting node and moves to the next random node on the graph. Each random walk sequence is treated as a sentence, in which the nodes are considered as the words of the sentence. Next, the word2vec Church [2017] model from NLP is applied to learn embeddings for the nodes. The resulting sentences are embedded by giving them as an input to a Skip-gram model, a

variant of the one used in the word2Vec model. After the skip-gram is trained, it predicts the context nodes based on its input and the output is the embedding of the nodes.

Node2vec Grover and Leskovec [2016], is a technique for learning node embeddings in a graph. In fact, it is an extension of the word2vec model which is a method for word embedding in textual data. It learns embeddings of the nodes in a graph by using a neighborhood sampling strategy which captures both local and global structures. Node2vec generates random walks which are balanced between breadth-First Search (BFS) and depth-First search (DFS) and applies it on the input data. After generating random walks, node2vec learns embeddings using the Skip-gram model. The Skip-gram model predicts the context (neighboring nodes) of a target node based on its embedding. Node2vec optimizes the obtained embeddings by maximizing the likelihood of observing neighboring nodes within the context window of each target node.

Metapath2vec Dong et al. [2017], is a representation learning method designed specifically for HINs to learn embeddings of the nodes and captures both semantic and structure of the network. In this method, meta-path guided random walks are used to make a sentence of the nodes. The meta-paths are created by domain experts based on nodes according to the dataset. For example, on a DBLP computer science bibliographic dataset Ley and Reuther [2006], the created meta-paths are : $APA, APVPA, OAPVPAO$, where A represents the author, P the paper, O the organization and V the venue. Consider APA as an example. The first node to choose must be of type A, the second node must be of type P and

the third node must be of type A. This means that at each step the next node to visit is chosen according to a pre-defined meta-path (APA in this example), ensuring that the walk follows a meaningful path in the network. After generating a large number of meta-path based random walks, these sequences of nodes are used as training data for the Skip-gram model. Finally, these embeddings are aggregated to obtain comprehensive representations for each node in the network beside capture the diverse relationships and semantic meanings associated with each node.

Regpattern2vec Keshavarzi et al. [2021], is a method for embedding KGs which samples a large knowledge graph to learn node embedding while capturing the semantic relationships among the nodes of the graph. In this method, the walk is biased by a fixed pattern $H[\hat{}T] + HT$ which is based on edges. The walk starts at a random node from a set of given nodes known as source nodes (S) on the knowledge graph. After choosing the source node, the walk chooses an edge of type H and moves to a randomly chosen neighbor creating a path at each time. Next, the walk chooses a random edge of type $\hat{}T$ and moves to the next random node. The walk follows the pattern $H[\hat{}T] + HT$ and continues the walk according to the a parameter called *walk length*. We can control the number of the walks in each path by *walk length* and when the number of the walks reaches to *walk length*, the walk is complete. Once the first walk is complete, we can start another walk by choosing a random node from S. The number of the times we start a new walk is based on a parameter called *number of walk*. Choosing a node from the set of source nodes each time as the starting node and creating random walks

will result in several paths of random walks. To obtain the embedding of these nodes, the resulted paths are fed into a modified version of a skip-gram model. This method of embedding is an unsupervised method and it can be applied on any given raw text corpus or document. In regpattern2vec, the algorithm runs the walks on a rigid regular expression $H[\hat{}T] + \text{HT}$ which is defined by domain experts and is hard-coded in the algorithm and cannot be changed.

Contribution: In the previously mentioned methods, the algorithms are either based on a pre-defined sequence of node/edge types or a pattern designed by domain experts or is biased toward specific nodes by experts. It means that they are not generic and the user does not have any role in guiding the walks. In our algorithm, however, we define a method in which the algorithm runs on any arbitrary random walk path inside a user-defined schema subgraph based on edges. The user enters a schema subgraph by entering integers where each integer represents an edge in the knowledge graph. This schema subgraph defines the subgraph. After the subgraph is defined, we choose the first random node as the source node inside this subgraph and continue the walk based on the *walk length* parameter and move to any random nodes via any random edges. The walks are valid only if they are within the subgraph and invalid otherwise. The advantage of using a subgraph is that it is more permissive; since we can run the walks totally randomly inside the user-defined subgraph rather than having biased walks based on a rigid pattern like the previous mentioned methods. The walks are valid as long as they are within the subgraph and invalid otherwise.

2.4 Methodology

Our algorithm runs with any user-given schema subgraph (s) based on the edges. The schema subgraph is entered in the form of integers where each integer denotes an edge of the KG. This schema subgraph which defines the subgraph (S) is actually a part of the original graph (G). Let's assume the user enters a schema subgraph such as $s = 'x_1, x_2, x_3'$ where $x_{i=1,2,3}$ is an integer representing an edge based on the dataset.

This schema subgraph defines subgraph $S = (V', E') \in G = (V, E)$ where G is the primary knowledge graph. The algorithm chooses the first node inside S randomly and from there, uses the below equation to calculate the probability of each neighbor edge based on its type (t). To calculate the probability of moving to the next edge based on subgraph S, we use this equation:

$$\sum_{i=1+1}^{l} P(r^{i+1}|r^i, S) =$$

$$\begin{cases} \frac{1}{r_{t_i}} \times \frac{1}{\sum_{i=1}^{n} t_i} & (r \in S) \\ 0 & Otherwise \end{cases}$$

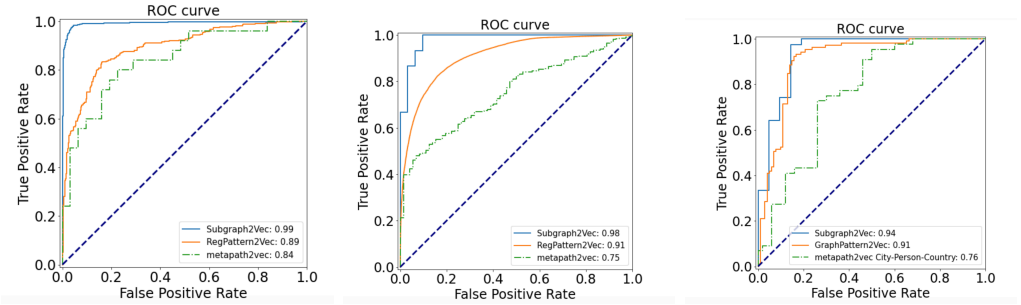
where t_i denotes each type of edges connected to the current node and r_{t_i}

denotes the number of the edges of each type. We choose the type of the next edge based on its probability. An important thing to consider is that we have a hierarchy of edges in our knowledge graph; which means that a type of edge might have different sub-types. With that being said, all the sub-types of a specific type should be considered as the same type.

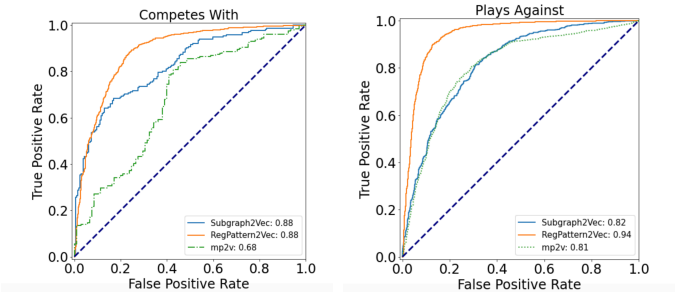
In addition, we have a parameter called *number of walks* in our code, which defines the number of the walks that should be walked in each path. The default number for it is set as 40, which the user can adjust on their own interest in the code.

2.4.1 Walk-based Embedding

After the user has entered the schema subgraph, the subgraph (S) is defined and we are able to conduct the walk. We start the walk by choosing the first node randomly within the sub-graph. Then we will choose an edge connected to this node randomly and if it is inside the schema-graph, the chosen edge is valid which means we choose the next random edge. Otherwise, we will delete that edge type from the neighbor of the node and choose another random edge and check its validation. We have decided to conduct the walk based on 40 steps (walk_length = 40). We should repeat this scenario for 40 walks (number_of_walks = 40) and these walks are written to a file. Both walk-length and number_of_walks can be modified by the user. After traversing all the walks and writing them on the file, we obtain the walk file which contains all the walks and their steps and then we should embed these walks. Embedding, in simple words is translating



(a) ROC on YAGO Dataset, predicted link:is citizen of (b) ROC on YAGO Dataset, predicted link:Located In (c) ROC on YAGO Dataset, predicted link:Leader of



(d) ROC on NELL Dataset, predicted link:competes with (e) ROC on NELL Dataset, predicted link:plays against

Figure 2.1: Comparing ROC of different links of Subgraph2vec, Regpattern2vec and Metapath2vec on NELL and YAGO.

high-dimensional vectors to a low-dimensional space. To embed the walks of the walk file, we can consider each walk as a sentence of the words in which each node is considered as one word. Therefore, we can get help from the word2vec model in which a neural network learns node embedding from a corpus.

In this paper, we are using a modified version of the skip gram model which captures the similarity of the walks based on their types. Skip gram is from the word2vec family and all of the word2vec models are consisting of two-layer neural networks used for word embedding. In a general sense, in the skip gram architecture, the model uses the current word (input) to predict the surrounding window -usually of size 5 to 10- of the context words (output). In fact, a skip gram is trying to find a semantic similarity between the words in a context by learning a meaningful representation of each word (embedding) in the document. After feeding the walk file as an input to the skip gram model and getting the embedding file, we are able to use it. The embedding file can be used for various tasks such as link prediction, node classification, community detection and etc. In this work, we use the embedding file for link prediction.

2.4.2 Application

In this paper, we decided to move forward with the link prediction task. For conducting link prediction, we need to train our model first. We do it by using the vector representation of the current edges in the graph which is considered as the positive example. Therefore, for negative examples, we can consider combining pairs of edges in the graph that are not connected. Both of the positive and

negative examples are used to train the classification model. We use Logistic Regression as the classifier, which can be used for link prediction as well.

2.5 Experiments

In this section, we will evaluate the conductance of the subgraph2vec method by running it on different datasets.

2.5.1 Dataset

We use two different datasets to evaluate our model.

(i) The first dataset is YAGO39K Lv et al. [2018], which includes data from Wikipedia, WordNet and GeoNames and is a subset of the YAGO knowledge base Chirita et al. [2007]. It contains 123,182 unique entities and 1,084,040 unique edges with 37 different relation types. (ii) The second dataset used was NELL, which is built from the Web via an intelligent agent and contains 49,869 unique nodes 296,013 edges and 827 relation types.

2.5.2 Experimental Setup

To apply random walks in Subgraph2vec, we set the number_of_walks = 40 and maximum walk_length = 40. Also, the logistic regression parameters are constant for all the datasets.

For both of our datasets, we split the dataset in order to train and test for any

Table 2.1: Statistics of split data (MST method) based on the relation to be predicted Keshavarzi et al. [2021]

Dataset	**Relations**	**Train est**	**Test set**
NELL	CompetesWith	9,154	1,070
	PlaysAgainst	2,945	2,225
YAGO	isLocatedIn	44,542	44,541
	isCitizenOf	3,128	342
	isLeaderOf	855	106

relation we want to predict in either of our datasets. Note that in our model, it is necessary to have the pair of the nodes we want to do prediction on in both the training and the test dataset. However, the relation between that pair is different in each of the train and test dataset. To achieve this goal, we apply minimum spanning tree method to take the minimum possible nodes from the graph to prepare the test dataset.

We have split the dataset for each of the relations we want to predict individually. Table 1 illustrates the number of the rows for each dataset (split with MST) based on the relation to be predicted.

2.5.3 Link Prediction

To explain the link prediction, we will give a brief explanation for each dataset. For the YAGO dataset, we decided to predict these relations: 'isLocatedIn', 'isCitizenOf', 'isLeaderOf'. Here is S defined for these relations:

For 'isLeaderOf', S consists of these edges: 'PlayisIn', 'isLeaderOf' and 'isLo-

catedIn'. For 'isCitizenOf', S consists of these edges: 'isCitizenOf', 'isLocatedIn' and 'isLocatedIn'. For 'isLeaderOf', S consists of these edges: 'isLeaderOf', 'isLocatedIn' and 'wasBornIn'.

For the NELL dataset, we chose two relations of interest: 'competesWith' and 'playsAgainst'. For 'competeswith', S consists of :'Competeswith', 'hasofficeincity' and 'cityhascompanyoffice'. For 'playsAgainst', S consists of :'teamplaysagainstteam' or in short 'playsagainst' which is the link to be predicted, 'teamplaysinleague' and 'sportsgameteam'.

Figure 1 shows the prediction results on the test datasets. Here, we are comparing the results of our method to the results we obtained from running regpattern2vec and metapath2vec algorithms using Nell and YAGO data sets. Our results imply that our method outperforms the regpattern2vec and metapath2vec methods in most cases. That is due to being capable of choosing the nodes/edges randomly within the subgraph rather than choosing them based on a regular expression. Figure 1 illustrates the ROC curve from each of the algorithms.

2.6 Conclusion and future work

In this paper, we present Subgraph2vec, a random walk-based method in which a subgraph is used for limiting random walks on a knowledge graph in a generic fashion. There are different random walk-based methods for embedding knowledge graphs such as node2vec, metapath2vec and regpattern2vec which were discussed

earlier. In the previous methods, random walks are biased based on different algorithms such as BFS and DFS in node2vec or fixed patterns in metapath2vec and regpattern2vec. However, our goal is to implement an algorithm which runs the random walks inside on a user-given subgraph; which makes the embedding algorithm more broad. The generated walk file is embedded using skipgram and the resulted embedding files can be used for various purposes such as node classification, link prediction, community detection and etc. In this work, we decided to use it for link prediction. Our results on NELL and YAGO datasets prove our method outperforms other methods such as regpattern2vec and metapath2vec in most cases. In the future work, we will use the obtained embedding file for the mentioned tasks and also in another work we will add weights to the edges to make our model more customizable.

Chapter 3

A Survey on the Recent Random Walk-based Methods for Embedding Graphs

Machine learning, deep learning and NLP methods on graphs are vastly present in different fields and have important roles in various domains from self-driving cars to friend recommendations on social media platforms. However, to apply these methods on graphs, the data usually need to be in an acceptable size and format. In fact, graphs normally have high dimensions, and therefore we need to transform them to a low-dimensional vector space. Embedding is a low-dimensional space into which one can translate high dimensional vectors in a way that intrinsic features of the input data are preserved. In this review, we first explain the importance of graphs and the embedding methods applied to them. Next,

we will review some of the random walk-based embedding methods as well as their strengths and weaknesses that have been developed recently. Later, we will address research directions for future research.

3.1 Introduction

Graphs provide a powerful data structure for representing complex relationships between entities. As a result, they naturally exist in many real-world scenarios; for example, in social networks, users are represented as nodes and connections (friendships) as edges Gavagsaz and Souri [2025]. They are also used in web crawling by modeling websites as nodes and hyperlinks as edges to determine the relevance of web pages Mohsin et al. [2024]. They are also widely used in navigation and routing such as Google Maps; representing cities as nodes and roads as edges to calculate the shortest path between two locations Rajvanshi et al. [2024]. Another area in which they are widely used is knowledge graphs; to represent information in a structured way for data integration and semantic understanding. For instance, *Google Knowledge Graph* uses semantic information from various sources to enhance its search engine results Steiner et al. [2012].

Furthermore, graphs serve as a foundation for AI and machine learning applications by providing structured data for different purposes such as training models and improving the interpretability of the AI systems. As an example, in the finance area, graphs are used by integrating different data sources and capturing

relationships between entities and using machine learning and AI techniques to discover patterns and anomalies indicative of fraud Mao et al. [2022]. In another case, graphs play a crucial role in drug discovery; they integrate large amounts of biomedical data and capture complex relationships between biological entities, facilitate data-driven decision making, and ultimately accelerate the drug development process Soleymani et al. [2023]. There are many other areas where graphs are used, such as networking and telecommunications Krinkin et al. [2020], manufacturing Buchgeher et al. [2021], autonomous vehicles Tezerjani et al. [2024], smart cities Ahmed et al. [2022], urban planning Liu et al. [2023], and etc.

To take advantage of graphs in various fields, we usually use machine learning, deep learning, and AI models on graph datasets. However, large graphs have a high dimension; which makes it challenging for many machine learning models to work with them. To overcome this problem, we use embedding. Embedding is a representation learning method to map out data to a lower-dimensional vector space, while preserving the main features of the input data Luo et al. [2003]. Good embedding offers a powerful way to represent data in an efficient manner which leads to semantic understanding, feature extraction and transfer learning capabilities in machine learning and AI. It also helps improve model performance for various tasks; such as prediction by capturing meaningful relationships, reducing noise and dimensionality, and generalizing across tasks.

According to Cai et al. [2018], there are five major categories to embed graphs.

These include matrix factorization, generative models, deep learning, graph kernels, and edge reconstruction-based optimization models. Here is a brief description for each category:

Matrix factorization models Koren et al. [2009]: In this model, the graph is represented in the form of a matrix, typically an adjacency matrix or a higher-order proximity matrix (e.g., Laplacian matrix or personalized PageRank matrix). Each entry in the matrix encodes relationships (e.g., edge weights or similarities) between nodes in the graph. Then, the matrix factorization method, decomposes the input matrix M into two smaller matrices: U and V, such that $M \approx U \cdot V^T$, where U is the embedding matrix of the node where the rows represent the nodes and the columns are features, V is another matrix that may encode complementary information, often equivalent to U in symmetric factorizations.

Generative models Ruthotto and Haber [2021]: Here, the graph is represented using an adjacency matrix, edge list or a feature matrix (if nodes or edges have attributes). The model maps out the graph components (e.g nodes) to a latent space, typically using techniques such as neural networks, matrix factorization or probabilistic models. Then, each node/edge is assigned a low-dimensional vector representation. The learned embeddings are used to reconstruct the graph or generate new graph instances which preserve the properties of the original graph. In addition, the generative model learns the probability distribution of the graph structure; which is used for tasks such as graph generation Manchanda et al.

[2024], link prediction Xian et al. [2022], or node feature reconstruction Hou et al. [2022].

Deep learning models Rumelhart et al. [1986]: Deep learning models for embedding revolutionized how we analyze and utilize graph-structured data, offering powerful tools for diverse applications. The input data in these models is the graph structure; such as adjacency matrix or edge list. The input can also include node/edge features too, but that is optional. Next, the model transforms high-dimensional or sparse input data into dense, continuous embeddings. Next, the model uses a loss function to optimize the embedding.

Graph kernels Shervashidze et al. [2011]: In fact, graph kernel is a method for measuring the similarity between graphs, nodes, or substructures and a key approach in graph embedding. In this model, the graph is decomposed into smaller substructures (e.g., walks, paths, trees, or subgraphs). Next, for a pair of graphs, a kernel function computes the similarity between these substructures. Then, a similarity matrix is constructed, capturing relationships between the graphs or their components. The similarity matrix is used as input for algorithms like SVM to perform classification Nikolentzos et al. [2021], regression Perez et al. [2024], or clustering Song et al. [2020].

Edge-Reconstruction-Based models Perozzi et al. [2014]: In this model, the core idea is to embed structural and relational information and ensure that the

model can predict the presence or weight of edges between nodes. The input is the adjacency matrix of the graph, the edge list or similar representation (including node/edge features is optional). Next, the model learns node embeddings that preserve graph topology and relationships by reconstructing the adjacency matrix or edge set of the graph. The model tries to minimize a loss function that measures the difference between the reconstructed (predicted) edges and the actual (ground truth) edges in the graph.

Each category includes different techniques for embedding knowledge graphs and several recent ones are summarized in Table 2 for each category. In addition, there are methods that do not fall into any of these categories entirely because they are combining a technique from these categories and another model. For example, there are methods that use random walks to traverse the graph and a model from the deep learning category to embed. Random walks have been used in many models for embedding, due to their important features such as exploring graph structures like local and global context, finding semantically similar nodes, feature extractions, etc.

To the best of our knowledge, embedding methods have been extensively studied across various domains of deep learning and random walks independently. However, there appears to be a lack of survey papers specifically focusing on random-walk-based embedding methods which combine deep learning models. In this work, we focus on the well-known random walk-based embedding techniques

which use a neural network from the deep learning category. These methods limit the amount of search space in a large graph developed during the recent years.

3.2 Preliminaries

In this section, we explain some preliminary concepts used in this review.

Knowledge Graph Sipser [1996]: A knowledge graph is a directed graph whose nodes are entities and edges are relations between entities. It is denoted as $G = (V, E)$ in which $v_i \in V$ are nodes or entities and $e_i \in E$ are edges or relations. Nodes have a type mapping function of $\phi : V \rightarrow T$ where T is the node type, and edges have a type mapping function of $\phi : E \rightarrow R$ where R denotes the edge type.

Homogeneous Network Cai et al. [2018]: Homogeneous Network is a graph denoted as $G = (V, E)$ where $|T^v| = |T^e| = 1$; meaning that all the nodes in G belong to a single type and all the edges in G has a single type, as well.

Heterogeneous Network Li et al. [2018]: is a graph denoted as $G = (V, E, T)$, where each $v \in V$ and $e \in E$ has a mapping function $\phi(V) = V \rightarrow T_v$ and $\phi(E) = E \rightarrow T_e$ and T_v and T_e denote sets of node and relation types respectively, where $|T_v| + |T_e| > 2$.

First-order Proximity Tang et al. [2015b]: represents the local pairwise

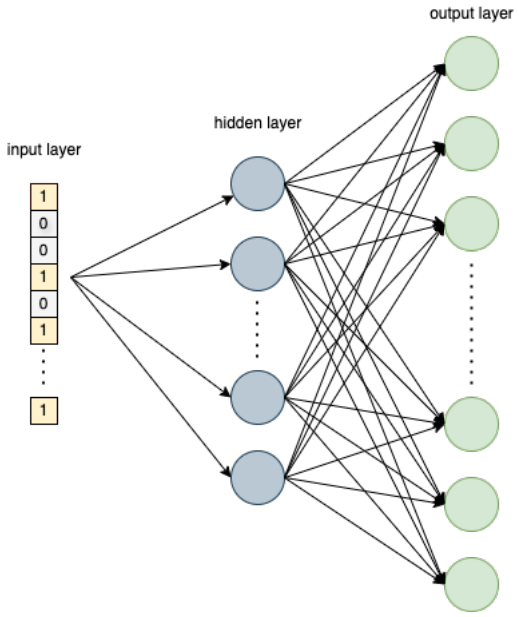


Figure 3.1: Skip-gram architecture.

relationship between directly connected vertices. For a vertex pair (v_i, v_j), if $(v_i, v_j) \in E$, the first-order proximity is w_{ij}; otherwise, it is 0. This measure captures direct neighbor relationships between vertices.

Second-order Proximity Tang et al. [2015b]: captures the two-step relationships between vertex pairs. For a pair (v_i, v_j), it is determined by their shared neighbors and can be equivalently measured by the two-step transition probability from v_i to v_j.

3.3 Methods

In this section, we summarize some of the recent algorithms for embedding knowledge graphs which were developed recently. These methods are combination of random-walk and with a deep learning model. Table 1 summarizes these algorithms.

Table 3.1: Random walk-based algorithms for embedding graphs based on their network type and used method Wang et al. [2022].

Algorithm	Year	Network Type	Random Walk Method
DeepWalk	2014	homogeneous	truncated random walks
LINE	2015		heterogeneous edges
Node2vec	2016		BFS + DFS based random walks
PTE	2017	heterogeneous	heterogeneous edges
Metapath2vec	2017		meta-path based random walks
Metapath2vec++	2019		meta-path based random walks
Regpattern2vec	2021		regular expression-based random walks
Subgraph2vec	2024		truncated random walks

3.3.1 DeepWalk Perozzi et al. [2014]

DeepWalk is a method for embedding nodes in a network, such as a social network or a biological network. Previous embedding methods often suffer from scalability issues and fail to capture the structural properties of large-scale networks effectively Tang and Liu [2009, 2011]. DeepWalk overcomes these issues in an unsupervised manner.

It is inspired by the techniques used in natural language processing (NLP), particularly the skip-gram (Fig. 1) framework from the word2vec model Church [2017]. The main idea behind DeepWalk is to treat random walks in a network as ”sentences” and learn node embeddings by predicting the context nodes given a target node in these walks.

The algorithm generates random walks of fixed length in a network, treating each node as a ”word” in a ”sentence”. It then uses the skip-gram model to learn node embedding by predicting the context nodes for each target node in these random walks. The learned embeddings capture the structural properties of the network, which can be used for various downstream tasks such as node classification, link prediction, and community detection.

Method: The algorithm aims to estimate the likelihood of a word sequence $W_1^n = (w_0 w_1 ... w_n)$ where $w_i \in V$ (V is the vocabulary) by maximizing $P_r(w_n | w_0 w_1 ... w_{n-1})$ over the training corpus. Also, it consists up of two major parts: 1) random walk generator and 2) update procedure.

Random walk generator: The algorithm has two nested loops; the outer loop, which represents the number of times (γ) the walk should start from each vertex v_i of the graph. DeepWalk starts by generating a random walk W_{vi} in the network from a random node called root (v_i). We set the parameter t to control the walk length to have walks of fixed length; however, the walks can be of any length as long as the length is smaller than t. The walks are performed starting from each node γ times and are entirely random which means they can revisit their root.

Update procedure: The inner loop iterates through the nodes of the graph and starts the walk from each of the vertices. Each random walk captures the local neighborhood information around each node. The skip-gram model learns embeddings for the nodes. The actual input for the original skip-gram model is sentences of words; therefore, we consider the walks as sentences in which the nodes represent the words. Given a sequence of nodes by random walk, the skip-gram model aims to predict the context nodes for each target node in the sequence. We use skip-gram to update the representations of the nodes according to the following objective function:

$$\underset{\Phi}{minimize} \quad -logP_r(v_{i-w}..vi + w\ v_i|\Phi(v_i))$$

where Φ is a mapping function $\Phi : v \in V \rightarrow R^{|v|*d}$. This mapping represents the latent social representation associated with each vertex v in the graph. The objective function, which uses the skip-gram model, is optimized using stochastic gradient descent to learn the parameters of the model and is as follows:

$$Pr(\{v_{i-w}...v_{i+w}\} \setminus v_i|\Phi(v_i)) = \prod_{\substack{j=i-w \\ j\neq i}}^{i+w} Pr(v_j|\Phi(v_i))$$

where $\phi(v_j)$ represents the embedding of vertex v_j.

The embeddings are learned iteratively by maximizing the log-likelihood of observing context nodes for each target node in the random walks. This involves updating the embeddings for each node using gradient descent based on the prediction error between the observed and predicted context nodes.

In general, DeepWalk is an unsupervised method for embedding homogenous graphs which is simple, effective, scalable and captures local structure by leveraging random walks. However, there are some limitations to this method such as: it generates fixed-length embeddings, it does not capture edge information such as edge weights or attributes, it only captures local network structure and it is not

efficient for embedding sparse nodes. Therefore, it is not suitable for dynamic, complex and heterogenous graphs.

3.3.2 LINE Tang et al. [2015b]

In this section, we review LINE, a method for embedding large networks. LINE is a suitable method to preserve the local pairwise proximity (local structure) between the vertices and dealing with very large networks (*millions of vertices and billions of edges*) with arbitrary types of edges (*directed, undirected, weighted*).

The pairwise proximity between vertices includes first-order proximity and second-order proximity. The first-order proximity illustrates the direct similarity between two vertices. For example, people who are friends on social networks probably share similar friends. However, first-order proximity on its own does not preserve the structure of the network. For example, consider a link is missing between two vertices sharing common neighbors. Although these two vertices are very similar, the first-order proximity in this case is 0. Therefore, another parameter that retains the network structure is needed and that is second-order proximity. The second-order proximity between a pair of vertices (u, v) implies the similarity between their neighbors. This helps identify objects that might not be directly connected but are related through shared neighbors.

Combining both of the above proximities, we form LINE; a method for embedding very large networks with arbitrary (i.e. directed, undirected, or weighted) edges.

Method: First, we combine LINE with each of the mentioned proximities individually and then we merge them. Here is a brief description of the model:

1. **LINE with first-order proximity**:

Since joint probability also implies dependencies and relationships between vertices in a graph, we model the first-order proximity for the undirected edge (v_i, v_j) between vertices v_i and v_j which is as follows:

$$p_1(v_i, v_j) = \frac{1}{1 + exp(-\overrightarrow{u_i}^T \cdot \overrightarrow{u_j})} \tag{3.1}$$

where u_i and u_j represent the vector representation of vertices v_i and v_j in a low dimensional space respectively. In particular, the joint probability of an edge represents the probability that the two specific nodes are connected by an edge simultaneously. It quantifies the probability of a specific edge which exists in the graph. On the other hand, there is another parameter to calculate the probability of edges called *empirical probability*. The empirical probability of an edge in a graph is based on observed data and represents the relative frequency with which a specific edge occurs in the observed graph and is calculated as $\hat{p_1}(i, j) = \frac{w_{ij}}{W}$ where $W = \sum_{(i,j)\in E} w_{ij}$. To preserve the first-order proximity of the network, we try to minimize the distance between these two functions:

$$O_1 = d(\hat{p}_1(\cdot, \cdot), p_1(\cdot, \cdot)) \tag{3.2}$$

where $d(\cdot, \cdot)$ is the distance between two vertices. We choose to minimize the KL-divergence of the two probability distributions. KL-divergence is a measure of how one probability distribution P diverges from a reference distribution Q. To minimize KL-divergence, we replace it with $d(\cdot, \cdot)$ in the above equation and remove some constants:

$$O_1 = -\sum_{(i,j)\in E} w_{ij} log p_1(v_i, v_j) \tag{3.3}$$

We can represent every node in the d-dimensional space by finding the $\{\overrightarrow{u}_i\}_{i=1,..|V|}$ in any undirected graph.

2. **Line with 2nd order proximity**:

The second-order proximity is applicable on both directed and undirected graphs. This proximity assumes vertices that share many other connections are similar to each other. In this proximity, each vertex has two roles: 1. "Vertex" itself and 2. "Context" of other vertices. In the second case, each vertex is considered as a specific context in which the vertices sharing similar distribution over the contexts are considered similar. Therefore, we will have two different representations for the vertex v_i: u_i and u'_i representing the embedding and the context of v_i,

respectively.

$$p_2(v_j|v_i) = \frac{1 + exp(-\overrightarrow{u_i'}^T \cdot \overrightarrow{u_j})}{\sum_{k=1}^{|V|} exp(\overrightarrow{u_k'}^T \cdot \overrightarrow{u_i})} \tag{3.4}$$

where $p(\cdot, v_i)$ is the conditional distribution over the contexts and $|V|$ is the number of vertices or contexts. In addition, this equation defines a conditional distribution $p_2(\cdot|v_i)$ over the entire set of vertices. To preserve the second-order proximity, we should minimize the distance between $p_2(\cdot|v_i)$ and the empirical distribution $\hat{p}_2(\cdot|v_i)$. Therefore:

$$O_2 = \sum_{i \in V} \lambda_i d(\hat{p}_2(\cdot|v_i), p_2(\cdot|v_i)) \tag{3.5}$$

where $d(\cdot, \cdot)$ is the distance between two distributions and λ_i denotes the importance of vertex i, which can be measured by the degree or estimated through algorithms such as PageRank Brin and Page [1998]. The empirical distribution is defined as $\hat{p}_2(\cdot, v_i) = w_{ij}/d_i$, where w_{ij} is the weight of the edge and d_i is the out-degree of vertex $i : d_i = \sum_{k \in N_i} w_{ik}$, where N_i is the set of out-neighbors of v_i. In this method, for simplicity, $\lambda = d_i$. We replace $d(\cdot, \cdot)$ with KL-divergence and omit some constants, therefore:

$$O_1 = -\sum_{(i,j) \in E} w_{ij} log p_2(v_i, v_j) \tag{3.6}$$

We can embed node v_i in the d-dimensional space by finding the $\{\overrightarrow{u}_i\}_{i=1,..|V|}$ and $\{\overrightarrow{u}'_i\}_{i=1,..|V|}$ in any directed/undirected graph.

In the LINE method, we embed the network with first-order and second-order proximity separately and then concatenate the embeddings by each of them for each vertex and get the desired embedding. By and large, here are the strengths of LINE: It is scalable since it is applicable on large-scale graphs. It also captures both local and global network structures, and is flexible and works well on both weighted and unweighted graphs. In addition, it is versatile because it can be extended on heterogeneous graphs. However, it comes with limitations, too: LINE is linear and cannot capture non-linear relationships in a graph. Also, although it can be extended to heterogenous graphs, it is not inherently designed to handle them. In addition, the embeddings are fixed length and therefore not sufficient to represent nodes in highly complex graphs. It primarily emphasizes pairwise relationships (first-order and second-order proximity) and does not naturally model higher-order structures or communities in the network. Also, LINE does not account for temporal information and therefore it is not suitable for dynamic graphs. In fact, LINE is best suited for large-scale homogeneous graphs where scalability and the preservation of local/global structures are the primary concerns.

3.3.3 Node2vec Grover and Leskovec [2016]

Node2vec is an embedding algorithm that maps the nodes of a graph to a low-dimensional space while preserving the structural properties of the network. It starts by generating random walks on the input graph. The random walks are biased to explore both local (Breadth-First Search or BFS) and global (Depth-First Search or DFS) neighborhoods of the nodes. These walks can be considered as sentences where nodes of the graph are similar to words of a sentence. The obtained walks are fed into a skip-gram model for embedding. The main reason developers use BFS and DFS is that they help us find similar nodes.

We measure the similarities between the embedded nodes by homophily Fortunato [2010], Yang and Leskovec [2014] and structural equivalence Henderson et al. [2012] hypotheses. Homophily refers to the tendency of nodes in a graph to be connected to other nodes with similar properties or attributes. Structural equivalence refers to the concept that if two or more nodes in a graph have identical relationships with the rest of the graph, then they are considered structurally equivalent. In other words, nodes are structurally equivalent if they share exactly the same neighbors. BFS emphasizes the nodes that are in the same community and follow the homophily hypothesis and DFS is used to sample nodes that share the same structural role and follow the structural equivalence law. Figure 2 illustrates BFS and DFS algorithms in a neighborhood.

Method: The algorithm is based on the random walk technique which walks

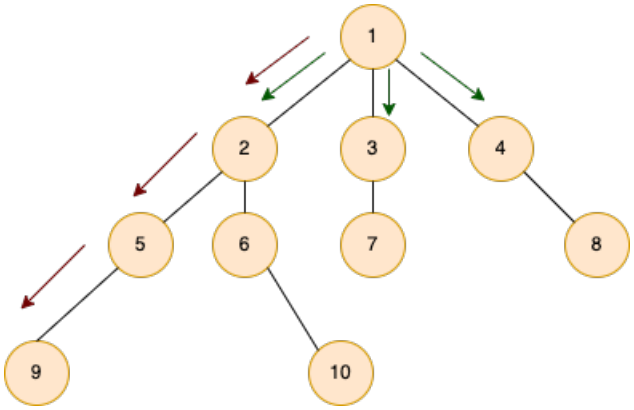


Figure 3.2: BFS and DFS algorithms for a neighborhood where node u is the source node Grover and Leskovec [2016]. Starting from node 1, BFS visits nodes: 1,2,3,4 and DFS visits nodes: 1,2,5,9.

the graph in both the DFS and BFS fashion. Let's consider u as the source node, and c_i as the ith node in the walk (therefore $c_0 = u$). The nodes in the walk are generated by this distribution:

$$P(c_i = x \,|\, c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & (r \in E) \\ \\ 0 & others \end{cases}$$

where π is the unnormalized transition probability between nodes v and x and Z is the normalizing constant.

The easiest way to bias our random walk is to sample the nodes based on the static edge weight w_{vx} i.e. $\pi_{vx} = w_{vx}$. In this case, if our graph is unweighted, we consider $w_{vx} = 1$. While this is the simplest way, it might not be a good choice since we cannot consider network structure and explore different types of network neighborhoods. Therefore, we design our algorithm which is a second-order random walk (in the first-order random walk, the walker traverses the graph from one node to a randomly chosen neighbor node at each step. In the second-order random walk, the walker considers the relationships between nodes based on their neighbors before moving to the next node). Our walk depends on two parameters *return parameter* or p and *in-out parameter* or q which control how fast the walk explores and leaves the neighborhood of the starting node u. Let's assume the walker just traversed edge (t, v) and now is at node v. The algorithm decides the next node based on this probability $\pi_{vx} = \alpha_{pq}.w_{vx}$, where:

$$\alpha_{pq}.w_{vx} = \begin{cases} \frac{1}{p} & d_{tx} = 0 \\ 1 & d_{tx} = 1 \\ \frac{1}{q} & d_{tx} = 2 \end{cases}$$

In this equation, d_{tx} is the shortest distance between the nodes x and t and can be one of $0, 1, 2$ values. Based on this formula, if we set p a high value, *i.e.*

($> max(q, 1)$), it is less likely to revisit a node that was just visited (unless the next node in the walk has no other neighbor). This strategy leads to moderate exploration and avoids two-hop redundancy in sampling. On the other hand, if we set p a low value, *i.e.* ($< min(q, 1)$), it is more likely that the walk is close to the source node since it moves the walk one step backward. In addition, if we set $q > 1$, the random walk is biased toward nodes close to t; which leads our walk to sample nodes within a small locality. On the other hand, if we set $q < 1$, the walk is more likely to explore further nodes from node t which encourages outward exploration.

The strength of node2vec includes: flexibility in capturing node similarity due to using both BFS and DFS which makes it more versatile than simpler methods like DeepWalk. It is also efficient, scalable and straightforward to use with current tools such as word2vec model. It can be applied to both homogeneous graphs and some heterogeneous ones and works well with unlabled or partially labeled data. However, it is parameter sensitive; which means the quality of the embeddings highly depends on the choice of p and q. Also, it is unsuitable for dynamic or evolving graphs. It struggles with most of the heterogeneous graphs unless they are transformed into homogeneous ones, which can lead to loss of information. For dense and comlicated graphs, they can be computationally expensive. Also, this method is not able to incorporate node or edge features and focuses only on graph topology.

3.3.4 Predictive Text Embedding (PTE) Tang et al. [2015a]

Predictive Text Embedding (PTE) is an extension of the LINE method to embed heterogeneous networks. It is a semi-supervised method used for embedding text data, which means it uses both labeled and unlabeled data to train the model. The labeled and unlabeled data are represented in a large heterogeneous network and then this heterogeneous network is embedded in a low dimensional space and can be used for text embedding. Not only does this method preserve the semantic closeness of the words and documents, but also it has good predictive power.
Compared to the unsupervised text embedding methods such as skip-gram or Paragraph Vectors (aka Doc2vec) Le and Mikolov [2014], which learn semantic representations of texts, the goal of this method is to learn a representation of the text that is optimized for a given text classification task. In other words, the authors anticipate text embedding has a strong predictive power of the performance of the given task. Since this method is applicable to different networks, we review some definitions of networks:

Word-Word Network: The word-word network captures the word co-occurrences in local contexts of unlabeled data. This data is the essential information used by some word embedding techniques such as skip-gram. Let $G_{ww} = (V, E_{ww})$ be a graph in which V is a vocabulary of words and E_{ww} is the set of edges between the words. Also, the weight w_{ij} is the number of times words v_i and v_j appear in the context window.

Word-Document Network: Word-document network, denoted as $G_{wd} = (V \cup D, E_{wd})$, is a bipartite network where V is a set of words and D denotes a set of documents. E_{wd} is the set of edges between the words and the documents. The weight w_{ij} between word v_i and document d_j is simply defined as the number of times v_i appears in document d_j. The mentioned networks are used for encoding unlabeled data. There is a network for encoding labeled information called the Word-Label network.

Word-Label Network: Let's take $G_{wd} = (V \cup L, E_{wd})$ as a bipartite network in which V is the set of words, L is the set of labels, and E_{wd} is the set of edges connecting words and labels. The weight w_{ij} of the edge between word v_i and class c_j is defined as $w_{ij} = \sum(d : ld = j)n_{di}$, where n_{di} is the term showing frequency of word v_i in document d, and ld is the class label of document d. The model embeds a network that is an integration of the above networks. This type of network is called a heterogeneous text network.

Heterogeneous Text Network: is the combination of word-word, word-document, and word-label networks constructed from both unlabeled and labeled text data.

Method: Given a large collection of text data with unlabeled and labeled information, the PTE algorithm tries to learn the embedding of the text by embedding the heterogeneous text network built from the collection.

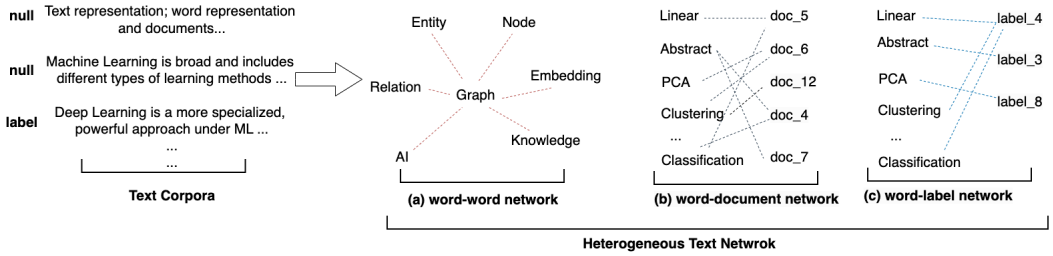


Figure 3.3: Converting partially labeled text corpora to a heterogeneous text network. The word-word co-occurrence and word-document networks encode the unsupervised information, capturing the local context-level and document-level word co-occurrences respectively. The word-label network encodes the supervised information, capturing the class-level word co-occurrences Tang et al. [2015a].

The heterogeneous graph is made up of three different bipartite networks, and therefore we have to embed each of these graphs individually (so far there is no technique to be able to embed these graphs all together simoltanously). To embed each of these bipartite graphs individually, we will use the LINE model. As mentioned earlier, PTE is an extension of the LINE method but LINE cannot be used to embed heterogeneous networks. Therefore, to start the embedding, we use the LINE method to embed a bipartite network. To use LINE, it is essential to make use of the second-order proximity between vertices; where every two nodes that have similar neighbors, can be considered similar to each other, which leads to closer vectors in the embedding space.

Given a bipartite graph, $G = (V_A \cup V_B, E)$, V_a, and V_b are two different types of nodes and E is the set of edges between them. For every v_i in V_a, generated by

v_j in V_b, the authors define the below formula to use the second-order proximity:

$$P(v_i, v_j) = \frac{\exp(u_i^T \cdot u_j)}{\sum_{i' \in A} \exp(u_{i'}^T \cdot u_j)} \tag{3.7}$$

where u_i is the embedding vector of vertex $v_i \in V_A$, and u_j is the embedding vector of vertex $v_j \in V_B$. For each vertex $v_j \in V_B$, Equation (1) defines a conditional distribution $p(\cdot|v_j)$ over all of the vertices in the set V_A and for each pair of vertices $v_j, v_{j'}$, the second-order proximity is determined by their conditional distributions $p(\cdot|v_j)$, $p(\cdot|v_{j'})$ respectively. Here is the equation to preserve the 2_{nd}-order proximity:

$$O = \sum_{j \in B} \lambda_d(p(\cdot|v_j), p(\cdot|v_j)) \tag{3.8}$$

where $d(\cdot, \cdot)$ is the KL-divergence between two distributions, λ_j is the importance of vertex v_j in the network, which can be set as the degree $deg_j = \sum w_{ij}$, and the empirical distribution can be defined as $p(v_j|v_i) = w_{ij}/deg_j$. Omitting some constants, here is the simpler version of Equation (2):

$$O = -\sum_{(i,j) \in E} w_{ij} \cdot \log p(v_j|v_i) \tag{3.9}$$

We can optimize the above equation using gradient descent which uses edge sam-

pling and negative sampling.

We can embed our 3 bipartite networks using the above model. Next, we want to embed the heterogeneous text network which consists of three bipartite networks: word-word, word-document, and word-label networks. To learn the embeddings of the heterogeneous network, our approach is to collectively embed the three bipartite networks by the following equation:

$$O_{pte} = O_{ww} + O_{wd} + O_{wl} \tag{3.10}$$

where O_{ww}, O_{wd} and O_{wl} are calculated individually by Equation (3).

Once the word vectors are learned, the representation learning of any piece of text can be obtained by averaging the vectors of the words in that piece of text.

$$d = \frac{1}{n} \sum u_i \tag{3.11}$$

In general, PTE uses matrix factorization which makes it scalable and is designed to work with heterogenous networks. It can generate customized embeddings for specific downstream tasks, such as text classification or sentiment analysis. It also allows for edge weighting and multiple edge types. The limitations include failing to capture global or higher-order graph structures. Also, it depends on static graphs and does not handle dynamic or evolving graphs. It is best suited for het-

erogeneous information networks and applications requiring scalable, task-specific embeddings, but it may fall short for dynamic, highly complex, or attributed graphs.

3.3.5 Metapath2vec and Metapath2vec++ Dong et al. [2017]

In this section, we review a neural network-based representation learning algorithm. There are recent different types of algorithms which use neural networks for embedding the graphs; such as Node2vec, LINE, and DeepWalk which we discussed earlier. Although these methods have their privileges such as automatic discovery of latent features from the raw network, they can only be applied to homogeneous networks. However, a large number of social and information graphs are heterogeneous; which means they have multiple types of nodes and edges. Therefore, we need new algorithms to embed them.

Here, we review metapath2vec and metapath2vec++, which are representation learning methods for embedding heterogeneous networks.

Metapath2vec: is a representation learning method applicable to heterogeneous networks. The metapath2vec method develops metapath-based random walks to construct the neighborhood of a node and then feeds the achieved random walks to a skip-gram model to obtain node embeddings.

Method: As mentioned earlier, metapath2vec generates meta path-based ran-

dom walks from the nodes of the graph. The most straightforward fashion is to start the meta path-based walk at a random node and then move to the next random node. In this context, the probability of moving to the next node is $P(v_{i+1}|v_i)$ regardless of the types of nodes. However, the walks are biased toward a highly visible type of nodes Sun et al. [2011]. To overcome this issue, given a graph $G = (V, E, T)$ the authors design a meta-path scheme to guide the walks in this form:

$$\rho = V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} V_3 \xrightarrow{R_3} \dots \ V_t \xrightarrow{R_t} V_{t+1} \xrightarrow{R_{l-1}} V_l \tag{3.12}$$

Hence, the transition probability for moving to the next node is:

$$P(v^{i+1}|v_t^i, \rho) = \begin{cases} \frac{1}{|N_{t+1}(V_t^i)|} & (v^{i+1}, v_t^i) \in E, \phi(v^{i+1}) = t+1 \\ 0 & (v^{i+1}, v_t^i) \in E, \phi(v^{i+1}) \neq t+1 \\ 0 & (v^{i+1}, v_t^i) \in E \end{cases}$$

where $v_t^i \in V_t$ and $N_{t+1}(v_t^i)$ denote the V_{t+1} type of neighborhood of node v_t^i . Also, meta-paths are designed in a symmetric way; which means the first node denoted as V_1 is the same as the last one, V_l, and therefore, has the same

probability of being reached, which means:

$$p(v_{i+1}|v_t^i) = p(v_{i+1}|v_l^i), \text{ if } t = l \tag{3.13}$$

For example, consider "APA" and "APVPA" as meta-path schemes where the former represents the "co-author collaboration on a paper" and the latter represents "two authors publish papers in the same venue". In the next step, the algorithm inputs the achieved random walks to a heterogeneous skip-gram model to embed the nodes. Given a heterogeneous graph $G = (V, E, T)$, the objective of using a heterogeneous skip-gram model is to maximize the network probability in terms of local structure or $N_{t(v)}, t \in T_v$, i.e:

$$argmax \sum_{v \in V} \sum_{t \in T_V} \sum_{c_t \in N_t(v)} logp(c_t|v;\theta) \tag{3.14}$$

where $N_{t(v)}$ is neighborhood of v with the $t^(th)$ type of nodes, and $p(c_t|v;\theta)$ is a softmax function, that is $p(c_t|v;\theta) = e^{X_{c_t}.X_v} / \sum(u \in V) e^{X_u.X_v}$ where X_v is the Vth row of X, representing the embedding vector for node v.

Metapath2vec incorporates semantic context. While guiding meta-path-based random walks, it captures meaningful semantic relationships between nodes. For example, in a graph of authors, papers, and venues, a "Author-Paper-Author" meta-path captures co-authorship relationships. In addition, metapath2vec recognizes the context nodes of node v when constructing its neighborhood function

N_v based on their types. However, it ignores these types in the softmax function. For example, if a meta-path is $A - P - V$, during the embedding training, the method uses all sampled nodes (A, P, V) in the same embedding space. Therefore, we introduce a modified version of metapath2vec to enhance the results.

Metapath2vec++: is an extension of metapath2vec designed to improve the quality of the embeddings in heterogeneous graphs. MetaPath2Vec++ enhances metaPath2Vec by introducing type awareness during the embedding learning process, making it more suitable for heterogeneous graphs with rich node and edge semantics. In metapath2vec++, the softmax function is normalized with respect to the type of the context node c_t:

$$p(c_t|v;\theta) = \frac{e^{X_{ct} \cdot X_v}}{\sum_{u \in V} e^{X_u \cdot X_v}} \tag{3.15}$$

where X_{c_t} represents the embedding of c_t and $p(c_t|v;\theta)$ is adjusted to the node type t and V_t is the node set of type t. In this case, we will have one set of multinomial distributions for each type of the c_t in the output layer of the skip-gram model. Figure 4, illustrates the differences among embedding results of some models:

Overall, metapath2vec++ is an appropriate method of embedding heterogenuous networks because it preserves the semantic and structural roles of different node types and their interactions. In addition, it filters out irrelevant relationships by focusing on type-specific contexts during embedding learning. However,it

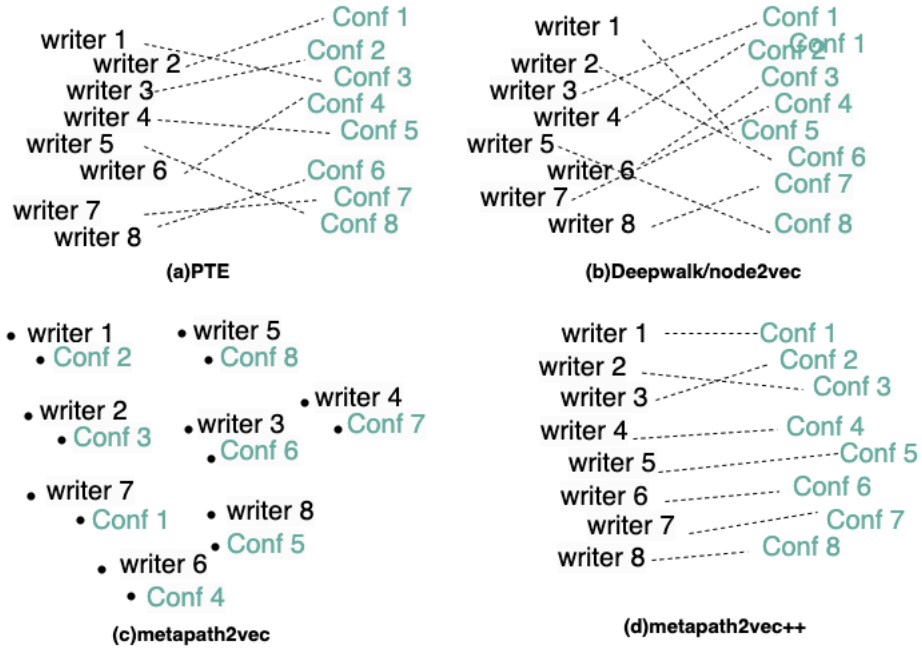


Figure 3.4: 2D PCA projections of the 128D embeddings of 16 top CS conferences and corresponding high-profile authors Dong et al. [2017].

comes with weaknesses, too. Type-aware context sampling and embedding learning makes this method computationally expensive compared to metapath2vec. Similar to metapath2vec, the quality of the embedding highly depends on the choice of metapath.

3.3.6 Regpattern2vec Keshavarzi et al. [2021]

In this section, we explain a method for embedding heterogeneous information networks (HINs). Regpattern2vec is an embedding algorithm which uses a fixed regular pattern (regular expression) to bias the random walks and then feeds the achieved walks to a modified version of the skip-gram to embed the walks. This fixed algorithm is helpful to speed up the random walk process especially on large graphs. In the original paper, the resulted embeddings are used for the link prediction task. Before explaining the regpattern2vec algorithm, we describe some preliminary concepts:

Regular expression: A regular expression or regular pattern defines a search pattern in a text in the form of a sequence of characters Rabin and Scott [1959]. In other words, a regular expression defines a set of strings matching it Friedl [2006]. Regular expressions can contain both ordinary and special characters. Ordinary characters refer to alphabets and numbers such as *A,b,5*. Special characters are non-alphabets and non-numbers such as *'('* and *'?'*. Special characters have special meanings in regular expressions. Some examples of regular expressions are *a3,5*, *^The.*Spain* , *H[^T] + HT*.

Regular pattern on knowledge graph: If $G = (V, E)$ is a knowledge graph which has a node type mapping function $\phi : V \to T$ and an edge type mapping function $\phi : E \to R$. A regular pattern r on G is formed over either set T or R as the alphabet.

Finite State Machine Lee and Yannakakis [1996]: also called Finite State Automaton, is a mathematical representation of computation which is an abstract concept but can also be implemented in software and hardware for different purposes; such as reducing the mathematical work in the theory of computation, pattern matching and lexical analysis. A Finite State Machine can be classified into two types: Deterministic Finite Automaton (DFA) and Non-Detereminsitic Finite Automaton (NDFA/NFA). Since our model is using the DFA, we only give a brief explanation of it.

As explained earlier, the codes accept a user input regex. Before running the random walk, we first want to make sure that the regex is valid and that random walks with that regex are applicable. We evaluate this validation via DFA. In this method, we have a DFA consisting of 5 states; in which we start from state 0 as the initial state and try to move through the states via the transition function, and then return to the final state (which is again state 0 in our case). If we arrive at the final state, the regex is valid; otherwise, the user should enter a different regex. If the regex is valid, the algorithm runs the walk based on this regex. Af-

ter the walk is finished, we feed the generated walks to a skip-gram for embedding.

Deterministic Finite Automata Lee and Yannakakis [1996]: or DFA, is a Finite State Machine that reads a string of symbols and either accepts or rejects it. For each input symbol, a state in the DFA is determined to which the machine moves. A DFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ in which: Q denotes the set of states, Σ (also called alphabet) denotes a finite set of symbols, δ is the transition function ($\delta : Q \times \Sigma \rightarrow Q$), q_0 denotes the initial state ($q_0 \in Q$) and F denotes the final state/states ($F \subseteq Q$). Basically, the machine works as follows: First, it takes the string (S) over the alphabet (Σ) as the input. Then, starting from the initial state (q_0), while reading each character of the string S, the machine moves to the next state by using the transition function. If the last alphabet of S makes the machine stop in F (the final state/any of the final states), the machine accepts the string, otherwise, rejects it.

Method: Here, we explain how regpattern2vec works. The algorithm runs on a fixed regular expression which is $r = H[\hat{}T] + HT$ based on the edges of the graph. Each of $(H, T, \hat{}T)$ denotes an edge type which has different sub-types. The user enters a regular expression based on the sub-types (r) of the edges and the algorithm uses DFA to check the validation of the entered regular expression. If the user-given regular expression matches the $r = H[\hat{}T] + HT$ format and the types of the chosen edges are compatible, the algorithm runs the walks based on this regular expression.

According to r, the random walk chooses the first edge randomly from any edges of type H, and the next edge is of any random edge from any type but T, and then chooses another available random edge of type H and then the next edge is of type T. In this case, the walk length is 4, however, if the walk length is more than 4, the algorithm repeats the walk in a back-and-forth fashion. This shows the walk moves backward and for the next edge, the walk chooses an edge of type H, and then chooses an edge of any type but T, then chooses type H and again moves forward afterward. The walk repeats the same thing until it reaches the walk length. At each step, the probability that a specific type of node is chosen is calculated as follows:

$$\sum_{i=1+1}^{l} P(v^{i+1}|v^i, M) = \begin{cases} \frac{\frac{1}{Nv_{i+1}}}{\sum_{i=1}^{n} \frac{1}{Nv}} & r \in G' \\ 0 & (v^i,\ r^i,\ v^{i+1}) \notin\ G' \end{cases}$$

Here, $|N_v|$ is the degree of node v, v_i indicates the current node, and v_{i+1} is the next candidate node.

Then, the random walks based on the defined regular expression are created. Next, we use a modified-version of the skip-gram to embed these walks. To capture the similarity of the edges based on their types and have close embeddings

in the latent space, our modified skip-gram takes into account the types of the edges. We use these biased walks as an input to the skip-gram and the output is the embeddings of these walks. In general, regpattern2vec is a flexible method due to supporting more complex and dynamic relationship patterns than traditional meta-path-based methods. Also, it can model intricate network traversal rules using regular expressions, accommodating diverse network structures. On the other hand, patterns used in it require expertise and can become more complex for large-scale networks. Also, the effectiveness of the embeddings relies greatly on the quality of the regpatterns chosen for the task.

3.3.7 Subgraph2vec Bozorgi et al. [2024]

Subgraph2vec is a representation learning technique that demonstrates the vector representation of the entities and relations of a knowledge graph in a low-dimensional space while maintaining their semantic meanings. The algorithm uses random walks and a modified version of the skip-gram to create the embeddings. In this method, the user enters a schema subgraph (i.e. subgraph of the complete schema graph), with the desire to bias the random walks on a specific part of the entire knowledge graph. The schema graph is in the form of an arbitrary set of integers based on the edges where each integer represents an edge. After the sub-graph is given, the algorithm chooses a random node inside the subgraph and starts the random walk. The next edge is chosen randomly inside the subgraph which moves to the next random node. The walk continues based on a parameter called *walk length*. Each chosen node/edge is valid only if it is within

the user-defined subgraph. This method is supposed to solve the deficiency of the previous random walk-based methods such as regpattern2vec and metapath2vec; where the walks are fixed and the experts have to define it.

By having a user-defined regular pattern, we focus the walks on important aspects of the graph rather than focusing it on any random part. In some of the previous mentioned methods, either the walk is biased on a fixed regular expression which is defined by experts or the user cannot give any pattern to guide the walk. However, this method is based on arbitrary random walks; as long as they are within the defined subgraph.

Method: The user enters a schema subgraph (S') in the form of integers, representing the edges of the schema subgraph. Let's, assume the user has entered this subgraph: $S' = x_1, x_2, x_3$, where each x denotes an edge in the graph. After the subgraph is given, a random node is chosen within this subgraph as the starting node. The walk starts at this node and in each step of the walk, a random edge is chosen. The chosen random edge is valid only if it is within the subgraph and invalid otherwise. The probability of choosing the next edge is calculated with this formula:

$$\sum_{i=1+1}^{l} P(r^{i+1}|r^i, S) =$$

$$\begin{cases} \frac{1}{r_{t_i}} \times \frac{1}{\sum_{i=1}^{n} t_i} & (r \in S) \\ \\ 0 & (r^i,\ r^i,\ r^{i+1}) \notin\ S' \end{cases}$$

where t_i denotes the type of each edge connected to the current node and r_{t_i} denotes the number of edges of each type. We choose the next edge from our valid set of edges according to its probability.

Subgraph2vec, is a flexible and customizable method becuase it depends on the user to define the subgraph rather than a fixed method and does not need expertise. By defining a subgraph, we can accelerate the walking process on the desired aspects of the knowledge graph. However, it may miss the global graph context and might face redundancy if the sugraphs overlap.

Method	Computational Efficiency	Embedding Quality	Adaptability Across Knowledge Graphs	Model Type	Strengths	Weaknesses
DeepWalk	High (scalable to large graphs)	Moderate (focus on local proximity)	Moderate (simple random walks limit versatility)	Random Walk-based	Simple, scalable to large graphs	Focused only on local relationships
LINE	High (supports large-scale graphs)	Moderate to high (captures first and second-order proximity)	High (suitable for diverse graph structures)	Proximity-preserving	Captures fine-grained proximity	Requires parameter tuning for balance
Node2Vec	Moderate to high (biased random walks)	High (flexible trade-off between local and global)	High (configurable for various graphs)	Random Walk-based	Flexible, captures diverse proximities	Higher computation cost than DeepWalk
PTE	High (designed for text-rich graphs)	Moderate (focus on heterogeneous networks)	Moderate (limited by domain-specific adaptations)	Text-based Graphs	Handles heterogeneous data well	Not ideal for generic graphs
Metapath2Vec	Moderate (metapath-based walks can be computationally expensive)	High (leverages heterogeneous graph semantics)	High (adaptable to heterogeneous graphs)	Random Walk-based	Captures semantic paths in heterogenous graphs	Expensive on large-scale graphs
Metapath2Vec++	Moderate (optimized metapath sampling)	High (improved over Metapath2Vec)	High (optimized for complex relationships)	Random Walk-based	More efficient than Metapath2Vec	Still limited by metapath complexity
RegPattern2Vec	Moderate (complex patterns increase cost)	High (focus on structural patterns)	Moderate (relies on predefined regular patterns)	Pattern-based	Captures structural nuances	Relies on predefined patterns
Subgraph2Vec	Moderate to low (subgraph isomorphism is expensive)	High (captures subgraph-level information)	Moderate (limited by subgraph definition)	Subgraph-based	Encodes subgraph-level features	Computationally expensive

Table 3.2: Analysis of Graph Embedding Methods with Full Features

3.4 Conclusion

In this work, we reviewed some methods for graph embedding based on the random-walk and deep learning. There are five different categories of methods for embedding graphs which include: matrix factorization, generative models, deep learning, graph kernels, and edge reconstruction-based optimization models. Each of these categories includes distinct subcategories and each subcategory contains different methods. We have provided examples and its relevant model type for each subcategory in Table 2. On the other hand, there are other methods that do not fall into any of these categories entirely. For example, there are methods that use random walks and a deep learning architecture. To the best of our knowledge, there appears to be a lack of survey papers specifically focusing on random walk-based embedding methods which combine a deep learning model.
We have summarized several recent random walk-based algorithms for embedding graphs which limit the amount of search space in a large graph. In addition, we have classified these methods based on the random walk technique in Table 1.
In addition, we have compared the methods explained in this paper in terms of computational efficieny, embedding quality, adaptability to graph structures, scalability and key strength and weakness. Table 3 summarizes these comparisons. The audience can choose the appropriate method for their work based on comparisons of the features in the tables. The future direction includes summarizing non-random walk based methods combining the deep learning category, a survey on the methods of other categories or a survey on methods for embedding dynamic graphs. Also, one can explore the applications of these methods in

real-world scenarios for different purposes such as: link prediction, visualization, clustering, anomaly detection, recommendation systems, fraud detection, etc. We hope that this survey gives better insights to the researchers in this field.

Chapter 4

Multi-hop Natural Language Question Answering on Graphs by Using LLMs

Widely used in different domains such as dialog interfaces and chat-bots, Question-Answering (QA) models are AI models that are often capable of answering questions given some context, as well as without any context, e.g. open-domain QA. LLMs are leveraged to enhance QA models, but only work well on one-hop questions (a type of question that can be answered by retrieving a single piece of information from one source or a single step of reasoning): the questions that can be answered with only one hop on their respective knowledge graph. In this work, we present a novel method to enable LLM enhanced QA models to answer multi-hop questions. Our proposed method consists of three steps: 1) in the first step,

we use LLMs for creating one-hop question answers for each edge of the graph schema and store these one-hop QAs in a Neo4j database, 2) next, the LLM uses these QAs to find the answer to the user's question, and 3) the LLM then generates Cypher query of the answers found in the previous step. We evaluate our method on two datasets in Neo4j. Our results indicate improvements in question answering with LLMs compared to the previous methods.

4.1 Introduction

Question answering (QA) on graphs has gained increasing popularity recently since it allows users to retrieve information from structured data. Using QA, the user does not need to have information about the underlying schema of the graph. QA is widely used in various fields such as chatbots (Dayal et al. [2023]), customer support services, personalized recommendations (Tai et al. [2021]), biomedical research and healthcare (Lin et al. [2024]), financial services (Tao et al. [2024]) and etc. While LLMs are widely used for simple question answering, there are still challenges in using them for multi-hop queries. In this work, we introduce a new method for question answering by using prompt engineering technique and build a novel system for multi-hop question answering.

LLMs are broadly used for question answering purposes and are trained on large datasets utilizing different techniques. Here, we discuss some of the training methods for LLMS:

- Retrieval-augmented generation (RAG). RAG is an innovative approach in

the field of NLP that enhances the quality of the generated text by combining the strengths of retrieval-based and generation-based models (Rizzi [2024]). In traditional LLMs, the generated responses are based on the data used for training the LLMs; which probably do not include the latest or accurate information in a specific field. RAG addresses this shortcoming by integrating retrieval-based models which grants access to the external databases, APIs or document repositories to the model (Gupta et al. [2024]).

- Fine-tuning. Fine-tuning is the process of taking a pre-trained model and training it further on a smaller, domain-specific dataset to adapt it to a specific task. It improves the accuracy of the model for specific tasks. Fine-tuning helps to reduce training time, resources and other computational expenses (VM et al. [2024]).

- Prompt engineering. A prompt is a natural language request that asks an LLM to perform a specific task or action. It gives the model context via tokens, and sidesteps the model's potential limitations, so that the model can give you a response. Prompt engineering in LLMs is the process of designing and optimizing prompts to obtain relevant, accurate and high-quality responses from language models (Zaghir et al. [2024]). There have been recent works on leveraging LLMs for multi-hop question answering. One recent work is (Shah and Tian [2024]), in which the authors use prompt engineering to train the LLM and use chain of hops reasoning for question answering.

- Another work in this field is Allemang and Sequeda [2024] in which the authors increase the accuracy of question answering on graphs by using ontologies and LLMs. By combining deterministic ontology checks with LLM-based repair, the approach significantly enhances reliability and correctness in QA systems, showcasing the power of knowledge graph semantics in boosting LLM performance.

- Few-Shot Knowledge Base Question Answering (FlexKBQA): Introduced by Li et al. [2024]; FlexKBQA improves knowledge base question answering by decomposing complex questions into interpretable components such as entity linking, relation extraction, and query structure prediction. These components are processed through a modular pipeline, allowing for flexibility, better error handling, and easier adaptation to new domains. The outputs are then combined to generate accurate SPARQL queries. This modular design enables the system to generalize effectively across different datasets and knowledge bases.

- Knowledge graph in the medical field (LLM-KGMQA), proposed in (Wang et al. [2025]). In this work, the authors leveraged LLMs to propose a method for multi-hop knowledge path reasoning and entity linking. Regarding the multi-hop knowledge path reasoning, the method proposed a three-step reasoning framework including an n-hop subgraph construction algorithm, a semantics-based knowledge pruning algorithm and a knowledge fusion algo-

rithm.

Problem: In this work, we use LLMs to generate a Cypher query for the user's multi-hop question in natural language. We cannot use LLMs directly to answer the user's question; because LLMs have challenges to answer multi-hop questions Huang et al. [2024]. For example, let's assume we have a graph dataset which represents an NFL player and their relevant information. The schema of this graph looks like this (Figure 1):

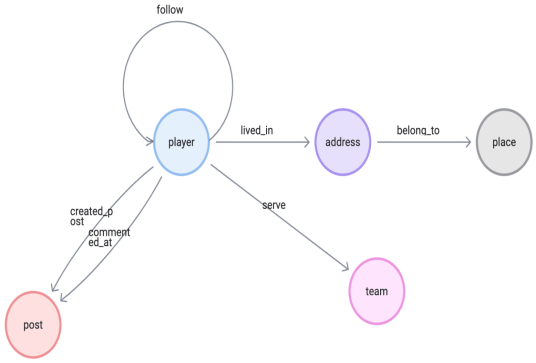


Figure 4.1: The Schema of the NFL Player and their information.

If the user asks a question about two connected nodes, such as nodes 'player' and 'address', the LLM can respond to the user's question properly. For example:

Q: Where does the player live in?

A: The player lives in Athens, GA.

However, if the user asks a question about two nodes that are not connected

with an edge such as 'place' and 'post', such as:

Q: What is the place of the player who posted on their twitter about the game?

the LLM has difficulties in answering the question for various reasons; which means that the LLM does not respond at all or respond with a wrong query. One reason is that the LLM cannot find the correct path between these two nodes. Another reason is that even by having the correct path, the LLM might not find all the relevant nodes; e.g if the player has several addresses assigned to them, the LLM might find only one of the addresses. Therefore, here we solve two problems:

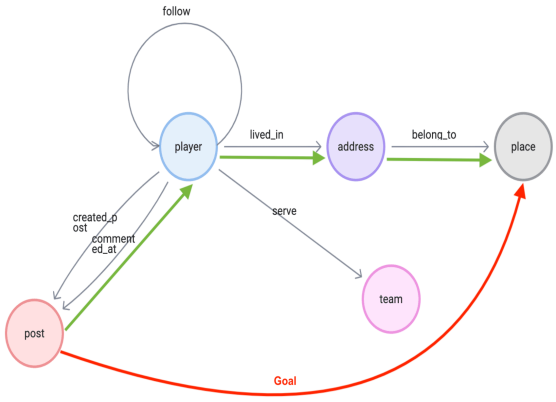


Figure 4.2: The Schema of the NFL Player and their information: The goal is to answer a question that connects the node 'post' to 'place'.

training LLMs to find a path for multi-hop question answering and generating a Cypher query of the user's question. We introduce a new method and leverage prompt engineering to train the LLM to find the right path for multi-hop question answering.

4.2 Related works

There are various approaches used for QA, and here is a list of some of the commonly used ones:

- *Semantic Parsing* Nguyen et al. [2024]; which involves converting a question in natural language into a machine readable language including query languages such as Cypher; which retrieves information from knowledge graphs. Another approach is *Deep Learning* Zhang et al. [2025], which uses trained neural networks to reason over knowledge graphs to answer questions or map questions directly to their corresponding answers.

- Using Neural Machine Translation (NMT); a method that uses machine learning to translate one language into another Stahlberg [2020]. In this case, NMTs can be leveraged to translate natural language to a structured language like Cypher.

- Large Language Models (LLM). LLMs are based on transformer architectures and trained on large volumes of data, and designed to handle human form of written language. The transformer model, which was introduced in 2017 in Zhang et al. [2024], is a type of neural network architecture that stands out at processing sequential data. As a result, they are mainly associated with LLMs but they are used in other fields such as computer vision Dubey and Singh [2024], time series Ni et al. [2024] and others. There are other well-known transformer-based architectures such as BERT and T5 which are used for natural language processing purposes including question

answering Chelliah et al. [2024], Yin et al. [2024] using different mechanisms. For example, T5 (Text-to-Text Transfer Transformer) developed by Google, converts all NLP tasks into a text-to-text format Grover et al. [2021]. On the other hand, BERT (Bidirectional Encoder Representations from Transformers) processes text bidirectionally, considering context from both left and right sides of a token Wang et al. [2024]. However, LLMs are generative, auto-regressive models that learn uni-directional (from left to right). LLMs have revolutionized NLP by using deep learning techniques to understand and generate human-like text. Due to their numerous benefits, various businesses take advantage of them in different domains such as customer service Larsen et al. [2024], healthcare Qiu et al. [2024], software development Manish [2024] and others.

The above mentioned research works are not accurate in answering quesitons that require multi-hop reasoning. In this work, we propose a multi-hop QA model to bridge this gap.

4.3 Methodology

In this work, we design an algorithm that translates user's multi-hop question from natural language into Cypher query. This helps users with no knowledge of the Cypher language to ask their questions in natural language and convert them to Cypher language in the chain-of-hops format. Then, the user runs this query on a Neo4j database to get the answer to their question. Let's explain the concept with

an example: Consider the Movies dataset where we have 6 different node types (Figure 4 (b)). The user wants to get more information about the genres of the movies and the actors acted in them. For example the user asks: "Which genres did (actor) Hugo Weaving act in?". This is an example of the multi-hop question since there is no direct edge between the nodes Actor (Hugo Weaving) and Genres.

Since it is a multi-hop question, the LLM on its own is typically not able to answer it. We train the LLM to find a path between the nodes "Actor" and "Genre" by prompt engineering. We teach it that the path is consisted of two steps: first find the connection between the nodes "Actor" and "Movie" and then "Movie" and "Genre". Next, we convert the user's question to a Cypher query by using LLMs and run the Cypher query on the Neo4j database to get the final answer. Our assumption is that the user does not have enough knowledge of the Cypher language to write the query on their own. This algorithm helps the user to obtain the Cypher format of their question and get the answer by running the Cypher query against the Neo4j database. Our algorithm is applicable on any graph dataset; however, in this paper, we get the results on two datasets: *Movies (neo4j-graph examples)* and *Northwind (Neo4j [2020])* . This method consists of three different steps (Figure 5 illustrates the structure of this method):

1. Let's assume we store the graph dataset and its schema in database (A) in Neo4j. We use LLM to generate one-hop QAs (in natural language format) for each edge type between any two nodes in the graph by using the data in database

A. We store the results in another Neo4j database (database B). For example, in the *Movies* dataset, we have 6 different node types (Figure 4 (b)). Considering the nodes *Actor: Hugo Weaving* and *Movie: Proof, Cloud Atlas, The Dressmaker, V for Vendetta, Babe, Matrix The, Interview The, Adventures of Priscilla, Queen of the Desert, The* and the edge type connecting them which is *Acted-In* (Figure 4), LLM generates a question answer (QA) like this:

Q: Did Hugo Weaving act in any movies?

A: Yes, Hugo Weaving acted in different movies such as Proof, Cloud Atlas, The Dressmaker, V for Vendetta (Figure 3).

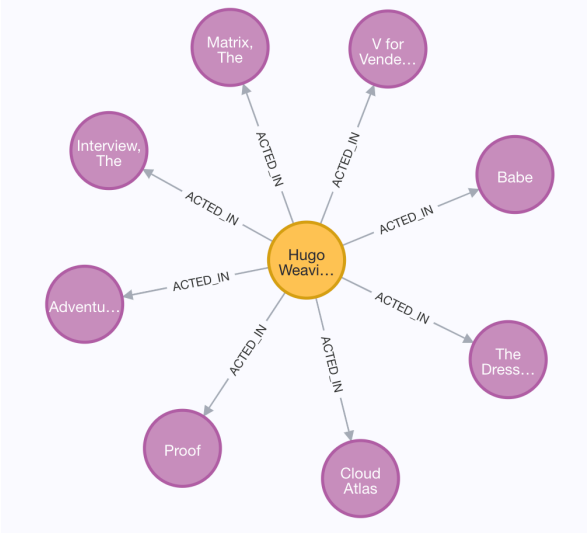
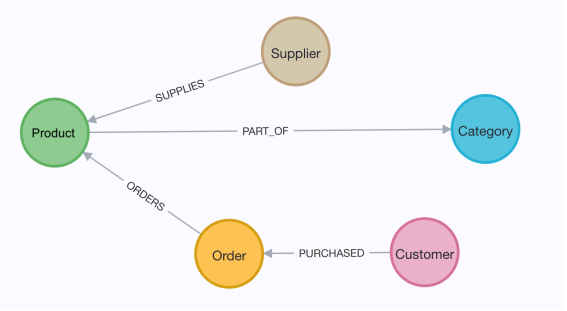
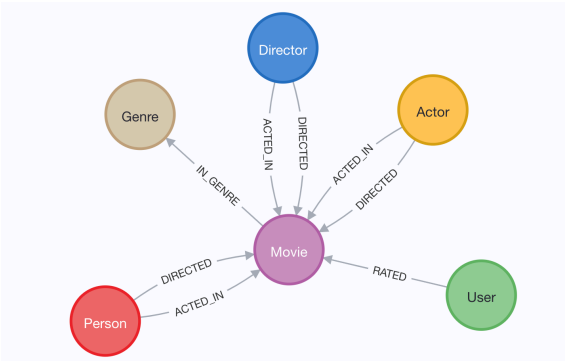


Figure 4.3: The movies which "Hugo Weaving" acted in.

2. Next, we run the algorithm which gets the user's question in natural lan-



(a) The schema of the Northwind dataset.



(b) The schemas of the dataset.

Figure 4.4: The schemas of the Movies and Northwind datasets.

guage and breaks down the user's question into smaller QAs by using database B. As mentioned earlier, the user's question is "Which genres did Hugo Weaving act in?". To find the answer, we train the LLM on how to find the path between the nodes "Actor" and "Genre". The path consists of two steps: 1. connecting nodes "Actor" and "Movie" 2. connecting the node "Movie" to "Genre". By leveraging prompt engineering, the LLM finds the path between the nodes "Actor" and "Genre" by finding the relevant QA from the database B: *User's question: Which genres did Hugo Weaving act in?* Using the few-shot training the LLM find the first step of the path from database B: *Q1: Did Hugo Weaving act in any movies? A1: Yes, Hugo Weaving acted in different movies such as Proof, Cloud Atlas, The Dressmaker, V for Vendetta*

Comparing Figure 3 and the LLM's response, the LLM has not found all the movies which Hugo Weaving acted in. That is one of the challenges of the LLM

to find all the connected Movie nodes to the Actor node. However, the good point is that it has found a connection between Hugo Weaving and the movies; which is the first step of the path. The second question that LLM finds from Database B is:

Q2: Are the genres of the movies Proof, Cloud Atlas, The Dressmaker, V for Vendetta available ? A2: Yes, the genres of these movies are drama, fiction, comedy in order.

3. In the last step, the code uses LLM and inputs the QAs in the previous step and generates Cypher queries for them. These queries are in format of Chain-of-hops. For example, considering the QAs in the previous step, here are the generated Cypher queries:

Query of the Q1:

MATCH (a:Actor name: "Hugo Weaving")-[r: Acted-In]-(m:Movie name: 'Proof, Cloud Atlas, The Dressmaker, V for Vendetta, Babe, Matrix The, Interview The, Adventures of Priscilla, Queen of the Desert, The')
RETURN m

Query of the Q2:

MATCH (g:Genre)-[]-(m:Movie name: 'Proof, Cloud Atlas, The Dressmaker, V

for Vendetta, Babe, Matrix The, Interview The, Adventures of Priscilla, Queen of the Desert, The')

RETURN g

The code later deletes the names of the movies from the query for a more accurate one resulting in:

MATCH (a:Actor name: "Hugo Weaving")-[r: Acted-In]-(m:Movie)

MATCH (g:Genre)-[]-(m:Movie)

RETURN g

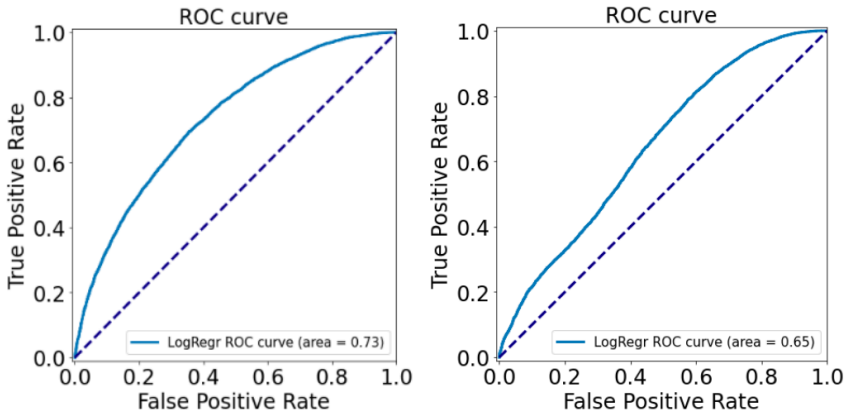


Figure 4.5: The framework used in the algorithm.

4.4 Experiments

In this section, we design a set of numerical experiments, in order to depict the efficiency and effectiveness of our proposed algorithm. To run the experiments, we used GPT-4o mini for all LLMs in the algorithm.

4.4.1 Datasets

For the experiments, we use two different datasets. One of them is the Northwind dataset; which is a dataset containing purchase history. This dataset simulates a fictitious company, "Northwind Traders", which deals with specialty food imports and exports. The database is widely used for educational purposes and practice; particularly with relational database management systems such as Neoj4 and SQL servers.

This dataset consists of 1,035 nodes and 3,139 edges. You can see the schema of this dataset as shown in Figure 4 (a). The second dataset is called "Movies" dataset; which is a graph of popular movies and actors. This dataset consists of 28,863 nodes and 166,261 edges. You can see the schema of this dataset as shown in Figure 4 (b).

4.4.2 Results

For the Northwind dataset, we ran our experiments for one-hop edge (between two nodes connected by one edge), two-hop edge (between three nodes connected by two edges). We implemented the algorithm on this dataset and shared the

results in Appendix A. Here are some of the results:

User Question: *What products did Antonio Moreno order?*

Answer:

MATCH (c:CustomercustomerName: 'Antonio Moreno')-[:placed]->(o:Order)

MATCH (o:Order)-[:contains]->(p:Product)

RETURN p.productName AS Products

User Question: *Which category does the supplier "New Orleans Cajun Delights" supply?*

Answer:

MATCH (s:SuppliercompanyName: 'New Orleans Cajun Delights')-[:supplies]->(p:Product)

MATCH (p:Product)-[:part-of]->(c:Category)

RETURN p.productName, c.categoryName

User Question: *Which category does supplier "Exotic liquids" supply?*

Answer:

MATCH (s:SuppliercompanyName: 'Exotic Liquids')-[:supplies]->(p:Product)

MATCH (p:Product)-[:part-of]->(c:Category)

RETURN p.productName, c.categoryName

User Question: Which category does supplier "Grandma's Boysenberry Spread"

supply?

Answer:

MATCH (s:SuppliercompanyName: ”Grandma Kelly’s Homestead”)-[:supplies]->(p:Product) MATCH (p:Product)-[:part-of]->(c:Category) RETURN c.categoryName AS c

User Question: *What product did Ana Trujillo order?*

Answer:

MATCH (c:CustomercustomerName: ’Ana Trujillo’)-[:placed]->(o:Order)

MATCH (o:Order) -(o)-[:contains]->(p:Product)

RETURN p.productName AS p

User Question: *What products did Maria Anders order?*

Answer:

MATCH (c:CustomercustomerName: ’Maria Anders’)-[:placed]->(o:Order)

MATCH (o:Order)-[:contains]->(p:Product)

RETURN p.productName AS p

User Question: *What categories are in order id 10250?*

Answer:

MATCH (o:OrderorderId: 10250)-[:contains]->(p:Product)

MATCH (p:Product)-[:part-of]->(c:Category)

RETURN p.productName AS Product, c.categoryName AS Category

User Question: *What categories are in order id 10248?*

Answer:

MATCH (o:OrderorderId: 10248)-[:contains]->(p:Product)

MATCH (p:Product)-[:part-of]->(c:Category)

RETURN p.productName AS Product, c.categoryName AS Category

User Question: *What categories did Thomas Hardy order from?*

Answer:

MATCH (c:CustomercustomerName: 'Thomas Hardy')-[:placed]->(o:Order)

MATCH (p:Product)-[:part-of]->(c:Category)

RETURN p.productName, c.categoryName AS Category

User Question: *What categories did Hanna Moos order from?*

Answer:

MATCH (c:CustomercustomerName: 'Hanna Moos')-[:placed]->(o:Order)

MATCH (o:Order)-[:contains]->(p:Product)

MATCH (p:Product)-[:part-of]->(c:Category)

RETURN p.productName, c.categoryName

Also, we implemented the algorithm and ran the code using the Movies dataset. We ran our experiments for one-hop edge (between two nodes), two-hop edge (between three nodes) and three-hop edge (between four nodes). Our results are

accurate for all the one-hop edge questions and in most cases for two-hop ones (above 80%). We shared the Cypher commands generated for ten QAs with this dataset in appendix B. Here are some of the results:

User Question: *Which actors acted in Susannah of the Mounties?*

Answer:

MATCH (m:Movie name: 'Susannah of the Mounties')<-[r:ACTED-IN]-(a:Actor)

RETURN a.name AS a"

User Question: *Who directed the movie "Just Around the Corner"?*

Answer:

MATCH (m:Movie name: 'Just Around the Corner')<-[r:DIRECTED]-(d:Director)

RETURN d.name AS d"

User Question: *What are the movies in Children genre?*

Answer:

MATCH (g:Genre name: 'Children')<-[r:IN-GENRE]-(m:Movie)

RETURN m.name AS m

User Question: *Which Genres did Channing Tatum act in?*

Answer:

MATCH (a:Actor name:'Channing Tatum')-[r:ACTED-IN]-(m:Movie)-[r1:IN-GENRE]-(g:Genre)

RETURN g

User Question: *Which genres did Ron Perlman act in?*

Answer:

MATCH (m:Movie)-[:IN-GENRE]->(g:Genre)

WHERE m.name IN ['Hellboy', 'Pacific Rim', 'Cronos']

RETURN m.name, collect(g.name) AS genres

User Question: *Which Genres did Donal Crisp direct?*

Answer:

MATCH (d:Director name:'Donal Crisp')-[r:DIRECTED]-(m:Movie)

MATCH (d:Director name:'Roberts Gannaway')-[r:DIRECTED]->(m:Movie)-[r2:IN-GENRE]->(g:Genre)

RETURN g.name AS Genre

User Question: *Which genres did Sam Raimi direct?*

Answer:

MATCH (m:Movie)-[:IN-GENRE]->(g:Genre)

RETURN m.name as Movie, collect(g.name) as Genres

User Question: *Which movies did George Davis act in?*

Answer:

MATCH (a:Actor name:'George Davis')-[r:ACTED-IN]-(m:Movie)

RETURN m.name AS m

User Question: *Which actors act in War genre?*

Answer:

MATCH (g:Genre name:'War')-[r:IN-GENRE]-(m:Movie)

(a:Actor)-[r:ACTED-IN]->(m:Movie)

WHERE m.name IN ['Saving Private Ryan', 'Black Hawk Down', 'Platoon', 'Full Metal Jacket']

RETURN a.name AS Actor, m.name AS Movie

4.5 Conclusion

In this paper, we introduced a new method for multi-hop question answering on graphs. Question answering is important since they are used in different domains such as chatbots, virtual assistants, information retrieval systems and etc. On the other hand, LLMs facilitate advancements in question answering by enabling rapid and accurate retrieval of information from extensive datasets, thereby streamlining both learning processes and decision-making.

As a result, we leverage them for question answering in our method. While LLMs facilitate QA, they are challenged for multi-hop queries; since LLMs cannot easily answer them. While there are recent articles introducing new methods for multi-hop question answering, this area is still challenging. Our method helps users to ask their multi-hop (or one-hop) question in natural language and translate it to

Cypher query. The user can run the resulted Cypher query on a Neo4j database and get the answer to their question.

Our method solves two issues: 1. translate the multi-hop question in natural language to Cypher query using LLMs 2. train the LLM using prompt engineering to find a path between two nodes of a graph.

The future work includes but not limited to: modifying the algorithm to make it more efficient, train the LLM on more complicated graphs, use other methods for training the LLM rather than prompt engineering, use other LLMs and compare the performance and results to the one we used.

Chapter 5

Conclusions and Future Work

In this knowledge, we focused on introducing and applying new machine learning methods for graph analysis. Graphs are fundamental data structures that capture relationships and connections in ways that traditional tabular data cannot. Here are some reasons why they're particularly important:

Natural representation of interconnected systems: Many real-world phenomena are inherently relational - social networks, biological systems, transportation networks, the internet, financial transactions, and knowledge graphs. Graphs preserve the structural information that gets lost when you flatten these relationships into tables.

Rich analytical capabilities: Graph datasets enable unique types of analysis that are impossible with other data structures. You can identify influential nodes, detect communities, find shortest paths, measure network resilience, and discover patterns of information flow. In social networks, this translates to understanding

how ideas spread, identifying key influencers.

Machine learning advantages: Graph neural networks and embedding techniques can leverage both node features and network structure simultaneously. This is crucial for tasks like recommendation systems (leveraging user-item and user-user connections), fraud detection (analyzing transaction patterns), or drug discovery (molecular structure analysis).

Scalability insights: Graphs reveal how systems behave at scale. Small changes in network structure can have dramatic effects on information propagation, system robustness, or computational complexity. Understanding these properties is essential for designing resilient systems.

Cross-domain applications: The same graph algorithms and insights apply across diverse fields - from analyzing protein interactions in biology to optimizing supply chains in logistics. This universality makes graph theory a powerful analytical lens. Social networks specifically showcase these benefits because human behavior emerges from complex social structures that graphs capture naturally - something that would be nearly impossible to model effectively with traditional data representations.

In this work, we presented different methods on how to take advantage of the information in graphs. In the first chapter, we introduce subgraph2vec, which is a method for embedding graphs. This method is based on random walks where the user has the authority to guide the walks. The user separates a subgraph (by entering the edge types) from the original graph and the random walks starts

within this subgraph. The obtained random walks are fed into a skip-gram for embedding and while the resulted embeddings can be used for various tasks, in this case they are used for link prediction using logistic regression.

In the next chapter, we discuss a survey some of the recent well-known methods for embedding graphs. In addition, we discuss their advantages and disadvantages and also compare subgraph2vec to them.

In the following chapter, we introduce a method to answer multi-hop questions on graphs using LLMs. We discuss how AI has affected people's lives and how one can benefit from LLMs for question answering. In general, question answering is used widely in different domains including chatbots, financial services, educational technology and others. As a result, we leverage LLMs to answer multi-hop questions by translating the user's question (in natural language) to Cypher format. Users can run this query on a Neo4j database and get the response to their question. We solve 2 challenges: 1. introducing a method to find a path between two unconnected nodes is that LLMs have challenges answering multi-hop questions. That is because they are often unable to find a path between two unconnected nodes. We address this issue by introducing a new method. 2. our method helps the users with little knowledge of Cypher to be able to use Neo4j database: Many people might not be familiar with the Cypher language but they may need to use Neo4j database; as a result we help these users to ask their questions in natural language while running the algorithm and get the equivalent of their question in

Cypher format. The user can run this Cypher query on a Neo4j database and get the answer to their question.

5.0.1 Future work

There are different ways to use information in graphs. While we introduced and discussed several ways in this work, one can focus on different ways for graph analysis. Regarding the graph embedding, one way is to apply subgraph2vec on larger graphs and modify the code for better efficency. Also, we ran our experiments on static graphs; while many graphs nowadays are dynamic. So, another possible direction is to run subgraph2vec on dynamic graphs. Also, for the multi-hop question answering, while we address the challenge of LLMs in finding a path in graphs, our method is not 100% accurate. As a result, the researcher can focus on how to improve the results by modifying the method or even introduce another fashion to solve this issue in different ways such as training the LLMs using other methods than prompt engineering.

Bibliography

U. Ahmed, G. Srivastava, Y. Djenouri, and J. C.-W. Lin. Knowledge graph based trajectory outlier detection in sustainable smart cities. *Sustainable Cities and Society*, 78:103580, 2022.

D. Allemang and J. Sequeda. Increasing the llm accuracy for question answering: Ontologies to the rescue! *arXiv preprint arXiv:2405.11706*, 2024.

A. Bordes, J. Weston, R. Collobert, and Y. Bengio. Learning structured embeddings of knowledge bases. In *Proceedings of the AAAI conference on artificial intelligence*, volume 25, pages 301–306, 2011.

A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

E. Bozorgi, S. Soleimani, S. K. Alqaiidi, H. R. Arabnia, and K. Kochut. Subgraph2vec: A random walk-based algorithm for embedding knowledge graphs. *arXiv preprint arXiv:2405.02240*, 2024.

S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

G. Buchgeher, D. Gabauer, J. Martinez-Gil, and L. Ehrlinger. Knowledge graphs in manufacturing and production: A systematic literature review. *IEEE Access*, 9:55537–55554, 2021.

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*, 30(9):1616–1637, 2018.

B. J. Chelliah, M. Hariharan, A. Prakash, B. Manoharan, and A. Senthilselvi. Harnessing t5 large language model for enhanced pdf text comprehension and q&a generation. In *2024 International Conference on Computing and Data Science (ICCDS)*, pages 1–6. IEEE, 2024.

X. Chen, Z. Hu, and Y. Sun. Fuzzy logic based logical query answering on knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3939–3948, 2022.

Z. Chen, Y. Wang, B. Zhao, J. Cheng, X. Zhao, and Z. Duan. Knowledge graph completion: A review. *Ieee Access*, 8:192435–192456, 2020.

P. Chirita, S. Costache, W. Nejdl, and S. Handschuh. Www'07: Proceedings of the 16th international conference on world wide web, 2007.

K. W. Church. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017.

R. Dayal, P. Nangia, S. Vijh, S. Kumar, S. Agarwal, and S. Saxena. Development of chatbot retrieving fact-based information using knowledge graph. In *Proceedings of International Conference on Recent Innovations in Computing: ICRIC 2022, Volume 1*, pages 153–164. Springer, 2023.

Y. Dong, N. V. Chawla, and A. Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 135–144, 2017.

S. R. Dubey and S. K. Singh. Transformer-based generative adversarial networks in computer vision: A comprehensive survey. *IEEE Transactions on Artificial Intelligence*, 2024.

F. Fan, G. Wu, Y. Yang, F. Liu, Y. Qian, Q. Yu, H. Ren, and J. Geng. A graph neural network model with a transparent decision-making process defines the applicability domain for environmental estrogen screening. *Environmental Science & Technology*, 57(46):18236–18245, 2023.

S. Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

J. E. Friedl. *Mastering regular expressions*. " O'Reilly Media, Inc.", 2006.

E. Gavagsaz and A. Souri. Triangle-induced and degree-wise sampling over large graphs in social networks. *The Journal of Supercomputing*, 81(1):1–29, 2025.

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

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*, pages 855–864, 2016.

K. Grover, K. Kaur, K. Tiwari, Rupali, and P. Kumar. Deep learning based question generation using t5 transformer. In *Advanced Computing: 10th International Conference, IACC 2020, Panaji, Goa, India, December 5–6, 2020, Revised Selected Papers, Part I 10*, pages 243–255. Springer, 2021.

S. Gupta, R. Ranjan, and S. N. Singh. A comprehensive survey of retrieval-augmented generation (rag): Evolution, current landscape and future directions. *arXiv preprint arXiv:2410.12837*, 2024.

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

K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1231–1239, 2012.

Z. Hou, X. Liu, Y. Cen, Y. Dong, H. Yang, C. Wang, and J. Tang. Graphmae: Self-supervised masked graph autoencoders, 2022. URL `https://arxiv.org/abs/2205.10803`.

E. Huaman. Steps to knowledge graphs quality assessment. *arXiv preprint arXiv:2208.07779*, 2022.

Q. Huang, F. Huang, D. Tao, Y. Zhao, B. Wang, and Y. Huang. Coq: An empirical framework for multi-hop question answering empowered by large language models. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 11566–11570. IEEE, 2024.

Z. Huang, W. Chung, and H. Chen. A graph model for e-commerce recommender systems. *Journal of the American Society for information science and technology*, 55(3):259–274, 2004.

A. Keshavarzi, N. Kannan, and K. Kochut. Regpattern2vec: link prediction in knowledge graphs. In *2021 IEEE international IOT, electronics and mechatronics conference (IEMTRONICS)*, pages 1–7. IEEE, 2021.

Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. doi: 10.1109/MC.2009.263.

K. Krinkin, I. Kulikov, A. Vodyaho, and N. Zhukova. Architecture of a telecommunications network monitoring system based on a knowledge graph. In *2020 26th Conference of Open Innovations Association (FRUCT)*, pages 231–239. IEEE, 2020.

A. G. Larsen, M. Skjuve, K. Kvale, and A. Følstad. Llm-powered conversational ai in customer service: Users' expectations and anticipated use. In *International*

Symposium on Chatbots and Human-Centered AI, pages 217–233. Springer, 2024.

Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

D. Lee and M. Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

M. Ley and P. Reuther. Maintaining an online bibliographical database: The problem of data quality. In *EGC*, pages 5–10. Citeseer, 2006.

J.-c. Li, D.-l. Zhao, B.-F. Ge, K.-W. Yang, and Y.-W. Chen. A link prediction method for heterogeneous networks based on bp neural network. *Physica A: Statistical Mechanics and its Applications*, 495:1–17, 2018.

Z. Li, S. Fan, Y. Gu, X. Li, Z. Duan, B. Dong, N. Liu, and J. Wang. Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 18608–18616, 2024.

X. Lin, S. Ma, J. Shan, X. Zhang, S. X. Hu, T. Guo, S. Z. Li, and K. Yu. Biokgbench: A knowledge graph checking benchmark of ai agent for biomedical science. *arXiv preprint arXiv:2407.00466*, 2024.

Y. Liu, J. Ding, Y. Fu, and Y. Li. Urbankg: An urban knowledge graph system. *ACM Transactions on Intelligent Systems and Technology*, 14(4):1–25, 2023.

B. Luo, R. C. Wilson, and E. R. Hancock. Spectral embedding of graphs. *Pattern recognition*, 36(10):2213–2230, 2003.

X. Lv, L. Hou, J. Li, and Z. Liu. Differentiating concepts and instances for knowledge graph embedding. *arXiv preprint arXiv:1811.04588*, 2018.

R. Ma, G. Pang, L. Chen, and A. van den Hengel. Deep graph-level anomaly detection by glocal knowledge distillation. In *Proceedings of the fifteenth ACM international conference on web search and data mining*, pages 704–714, 2022.

X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu. A comprehensive survey on graph anomaly detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12012–12038, 2021.

S. Manchanda, S. Gupta, S. Ranu, and S. J. Bedathur. Generative modeling of labeled graphs under data scarcity. In *Learning on Graphs Conference*, pages 32–1. PMLR, 2024.

S. Manish. An autonomous multi-agent llm framework for agile software development. *International Journal of Trend in Scientific Research and Development*, 8(5):892–898, 2024.

X. Mao, H. Sun, X. Zhu, and J. Li. Financial fraud detection using the related-party transaction knowledge graph. *Procedia Computer Science*, 199:733–740, 2022.

S. F. Mohsin, S. I. Jami, S. Wasi, and M. S. Siddiqui. An automated information

extraction system from the knowledge graph based annual financial reports. *PeerJ Computer Science*, 10:e2004, 2024.

Neo4j. Northwind graph example. `https://github.com/neo4j-graph-examples/northwind`, 2020. Accessed: 2025-07-14.

neo4j-graph examples. recommendations. `https://github.com/neo4j-graph-examples/recommendations`. Accessed: 2025-07-12.

L.-M. Nguyen, L.-N. Khang, K. Q. Anh, N. D. Hien, and Y. Nagai. Semantic parsing for question and answering over scholarly knowledge graph with large language models. In *JSAI International Symposium on Artificial Intelligence*, pages 284–298. Springer, 2024.

H. Ni, S. Meng, X. Geng, P. Li, Z. Li, X. Chen, X. Wang, and S. Zhang. Time series modeling for heart rate prediction: From arima to transformers. In *2024 6th International Conference on Electronic Engineering and Informatics (EEI)*, pages 584–589. IEEE, 2024.

G. Nikolentzos, G. Siglidis, and M. Vazirgiannis. Graph kernels: A survey. *Journal of Artificial Intelligence Research*, 72:943–1027, 2021.

R. C. Perez, S. Da Veiga, J. Garnier, and B. Staber. Gaussian process regression with sliced wasserstein weisfeiler-lehman graph kernels. In *International Conference on Artificial Intelligence and Statistics*, pages 1297–1305. PMLR, 2024.

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*, pages 701–710, 2014.

A. Polleres, R. Pernisch, A. Bonifati, D. Dell'Aglio, D. Dobriy, S. Dumbrava, L. Etcheverry, N. Ferranti, K. Hose, E. Jiménez-Ruiz, et al. How does knowledge evolve in open knowledge graphs? *Transactions on Graph Data and Knowledge*, 1(1):11–1, 2023.

J. Qiu, K. Lam, G. Li, A. Acharya, T. Y. Wong, A. Darzi, W. Yuan, and E. J. Topol. Llm-based agentic systems in medicine and healthcare. *Nature Machine Intelligence*, 6(12):1418–1420, 2024.

M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959. doi: 10.1147/rd.32.0114.

A. Rajvanshi, K. Sikka, X. Lin, B. Lee, H.-P. Chiu, and A. Velasquez. Saynav: Grounding large language models for dynamic planning to navigation in new environments. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 464–474, 2024.

R. Reinanda, E. Meij, M. de Rijke, et al. Knowledge graphs: An information retrieval perspective. *Foundations and Trends® in Information Retrieval*, 14 (4):289–444, 2020.

F. Rizzi. *Developing an Enterprise Chatbot using Machine Learning Models: A RAG and NLP based approach*. PhD thesis, Politecnico di Torino, 2024.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. URL `https://api.semanticscholar.org/CorpusID:205001834`.

L. Ruthotto and E. Haber. An introduction to deep generative modeling. *CoRR*, abs/2103.05180, 2021. URL `https://arxiv.org/abs/2103.05180`.

F. Saadati, I. Chehreh, and E. Ansari. The role of social media platforms in spreading misinformation targeting specific racial and ethnic groups: A brief review. In *Proceedings of the 36th Conference of Open Innovations Association FRUCT, Helsinki, Finland*, 2024.

A. Saxena, A. Tripathi, and P. Talukdar. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 4498–4507, 2020.

M. Shah and J. Tian. Improving llm-based kgqa for multi-hop question answering with implicit reasoning in few-shot examples. In *Proceedings of the 1st Workshop on Knowledge Graphs and Large Language Models (KaLLM 2024)*, pages 125–135, 2024.

N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learn-*

ing Research, 12(77):2539–2561, 2011. URL `http://jmlr.org/papers/v12/shervashidze11a.html`.

M. Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1): 27–29, 1996.

M. Sipser. Introduction to the theory of computation. cengage learning. *International edition*, 2012.

S. Soleymani, N. Gravel, L.-C. Huang, W. Yeung, E. Bozorgi, N. G. Bendzunas, K. J. Kochut, and N. Kannan. Dark kinase annotation, mining, and visualization using the protein kinase ontology. *PeerJ*, 11:e16087, 2023.

Y. Song, H. Luo, S. Pi, C. Gui, and B. Sun. Graph kernel based clustering algorithm in manets. *IEEE Access*, 8:107650–107660, 2020. doi: 10.1109/ACCESS.2020.3001137.

F. Stahlberg. Neural machine translation: A review. *Journal of Artificial Intelligence Research*, 69:343–418, 2020.

T. Steiner, R. Verborgh, R. Troncy, J. Gabarro, and R. Van de Walle. Adding realtime coverage to the google knowledge graph. In *11th International Semantic Web Conference (ISWC 2012)*, volume 914, pages 65–68. Citeseer, 2012.

Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003, 2011.

C.-Y. Tai, L.-Y. Huang, C.-K. Huang, and L.-W. Ku. User-centric path reasoning towards explainable recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 879–889, 2021.

J. Tang, M. Qu, and Q. Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1165–1174, 2015a.

J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015b.

L. Tang and H. Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 817–826, 2009.

L. Tang and H. Liu. Leveraging social media networks for classification. *Data mining and knowledge discovery*, 23:447–478, 2011.

W. Tao, H. Zhu, K. Tan, J. Wang, Y. Liang, H. Jiang, P. Yuan, and Y. Lan. Finqa: A training-free dynamic knowledge graph question answering system in finance with llm-based revision. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 418–423. Springer, 2024.

X. Tao, T. Pham, J. Zhang, J. Yong, W. P. Goh, W. Zhang, and Y. Cai. Mining

health knowledge graph for health risk prediction. *World Wide Web*, 23:2341–2362, 2020.

M. D. Tezerjani, D. Carrillo, D. Qu, S. Dhakal, A. Mirzaeinia, and Q. Yang. Real-time motion planning for autonomous vehicles in dynamic environments. *arXiv preprint arXiv:2406.02916*, 2024.

A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller. Graph clustering with graph neural networks. *Journal of Machine Learning Research*, 24(127):1–21, 2023.

R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, and P. Colpaert. Triple pattern fragments: a low-cost knowledge graph interface for the web. *Journal of Web Semantics*, 37: 184–206, 2016.

K. VM, H. Warrier, Y. Gupta, et al. Fine tuning llm for enterprise: Practical guidelines and recommendations. *arXiv preprint arXiv:2404.10779*, 2024.

F. Wang, D. Shi, J. Aguilar, X. Cui, J. Jiang, L. Shen, and M. Li. Llm-kgmqa: large language model-augmented multi-hop question-answering system based on knowledge graph in medical field. *Knowledge and Information Systems*, pages 1–43, 2025.

J. Wang, J. X. Huang, X. Tu, J. Wang, A. J. Huang, M. T. R. Laskar, and A. Bhuiyan. Utilizing bert for information retrieval: Survey, applications, resources, and challenges. *ACM Computing Surveys*, 56(7):1–33, 2024.

X. Wang, D. Bo, C. Shi, S. Fan, Y. Ye, and S. Y. Philip. A survey on heterogeneous graph embedding: methods, techniques, applications and sources. *IEEE Transactions on Big Data*, 9(2):415–436, 2022.

Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi. Mixup for node and graph classification. In *Proceedings of the Web Conference 2021*, pages 3663–3674, 2021.

Wikipedia contributors. Path (graph theory) — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Path_(graph_theory)&oldid=1191420024`, 2023. [Online; accessed 16-February-2024].

X. Xian, T. Wu, X. Ma, S. Qiao, Y. Shao, C. Wang, L. Yuan, and Y. Wu. Generative graph neural networks for link prediction, 2022. URL `https://arxiv.org/abs/2301.00169`.

C. Yang, Y. Feng, P. Li, Y. Shi, and J. Han. Meta-graph based hin spectral embedding: Methods, analyses, and insights. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 657–666. IEEE, 2018.

J. Yang and J. Leskovec. Overlapping communities explain core–periphery organization of networks. *Proceedings of the IEEE*, 102(12):1892–1902, 2014.

L. Yao, C. Mao, and Y. Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7370–7377, 2019.

L. Yin, L. Wang, Z. Cai, S. Lu, R. Wang, A. AlSanad, S. A. AlQahtani, X. Chen, Z. Yin, X. Li, et al. Dpal-bert: A faster and lighter question answering model. *CMES-Computer Modeling in Engineering & Sciences*, 141(1), 2024.

J. Zaghir, M. Naguib, M. Bjelogrlic, A. Névéol, X. Tannier, and C. Lovis. Prompt engineering paradigms for medical applications: Scoping review. *Journal of Medical Internet Research*, 26:e60501, 2024.

L. Zhang, W. Zhao, Z. Cheng, Y. Jiang, K. Tian, J. Shi, Z. Jiang, and Y. Hua. Osteosarcoma kgqa system: deep learning-based knowledge graph and large language model fusion. *Intelligent Medicine*, 2025.

M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34:9061–9073, 2021.

Y. Zhang, C. Liu, M. Liu, T. Liu, H. Lin, C.-B. Huang, and L. Ning. Attention is all you need: utilizing attention in ai-enabled drug discovery. *Briefings in bioinformatics*, 25(1):bbad467, 2024.