TOWARDS META-DATA DISCOVERY AND KNOWLEDGE DISCOVERY ON KNOWLEDGE GRAPHS

by

ABBAS KESHAVARZI

(Under the Direction of Krzysztof J. Kochut)

ABSTRACT

A knowledge graph (KG) provides a framework for data representation, integration, analytics by expressing sets of linked descriptions of entities and places data in a context via semantic metadata, and it helps to enrich the data with computer-processable semantics. In many domains, the KG aids researchers blend related information to a single source for effortless and efficient investigations. External resources and datasets, usually Web documents, are acquired by software programs for the purpose of creating or evolving a KG create or evolve a KG. New findings lead to changes in the original data sources relentlessly; therefore, the generated KG should comply with the changes. The introduced changes can range from individual entities and their relations to more significant changes in the KG schema. In both cases, the domain expert or KG engineer should employ mechanisms to track them and take proper actions. The structure and connectivity among entities in graph-like data make researchers curious about finding new associations by visualizing or querying the data. With the rise of diverse machine learning techniques, this process can be more efficient and achievable. Thus, the link prediction task becomes one of the priorities on KGs, especially in domains such as biology, social networks, and

recommender systems. It generally aims to discover unknown linkage between existing entities in the KG. Machine learning techniques for link prediction have become popular solutions for link prediction, especially deep learning (DL) methods. The scalability issue of these approaches for large graphs calls for an alternative direction. Toward addressing these issues, this dissertation investigates scalable approaches for evaluating and using KG for knowledge discovery. First, we present our work, KGdiff, for tracking the evolution of KGs by discovering meta-data information from KGs and then we introduce RegPattern2Vec for the link prediction problem and its successful application on a large biological dataset.

INDEX WORDS: knowledge graph, link prediction, ontology, OWL, machine learning, random walk, knowledge discovery, meta-data discovery

TOWARDS META-DATA DISCOVERY AND KNOWLEDGE DISCOVERY ON KNOWLEDGE GRAPHS

by

ABBAS KESHAVARZI

B.S., North Tehran Azad Islamic University, Iran, 2004

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

2021

TOWARDS META-DATA DISCOVERY AND KNOWLEDGE DISCOVERY ON KNOWLEDGE GRAPHS

by

ABBAS KESHAVARZI

Major Professor:	Krzysztof J. Kochut
Committee:	Hamid R. Arabnia
	Ismailcem B. Arpinar

Electronic Version Approved:

Ron Walcott
Vice Provost for Graduate Education and Dean of the Graduate School
The University of Georgia
August 2021

To my wife and my parents

ACKNOWLEDGEMENTS

I would like to acknowledge my major advisor Prof. Krzysztof J. Kochut who shed a light on my path with his great guidance and support and motivated me through my study.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Storing and representing the increasing amount of data, produced by millions of users, introduced great challenges for scientists in different domains. The data should be stored in a way that it can be queried effortlessly and efficiently to better help experts discover new knowledge. On the other hand, representation of data based on the desired task should consider machine-readability or human understandability. The mid-point would be desirable when representation consider both aspects. Another import consideration is, as the data might have different forms from text to documents, the architect of the data store should be flexible enough to contain such a diversity. One of the most popular solutions is to use knowledge graphs. Knowledge graphs are a way of storing data in a form of a graph, where entities from different types might have a variety of relationships with other entities. These knowledge graphs can be encoded in different forms, which we discuss next.

One of the most popular ways is to use Web Ontology Language[1] (OWL) to express the data and define relations among the entities in a KG. The expressivity of OWL gives is a huge advantage to explain complex relationships and the light-weighted vocabulary make it possible to store or even transfer huge amount of data. The more recent choice is to use graph databases. A graph database, such as Neo4j[2], offers major advantages, such as

[1] https://www.w3.org/TR/owl-features/
[2] https://neo4j.com/

performance, flexibility and agility. For graph data, it performs much better than relational databases and the flexible schema gives users the ability to keep up with new demands in evolution of the data. As important graph algorithms are often built in some of the graph databases, such as Neo4j, they became the source of extensive popularity of KGs among researchers in different domains, such as social networks, biomedicine, health sciences, and many more.

KGs are usually populated using a variety of external data sources, such as Web documents, articles, and other diverse datasets. As the data evolve due to advancement of science or acquiring more data, domain experts need to understand the changes introduced to the KG, which sometimes cause massive structural changes, like transformation in the schema of KG. The domain experts and knowledge engineers need to be informed of these changes and validate their generated KGs. However, manual version comparison is a time-consuming process and thus impractical.

On the other hand, since these KGs enable their users to identify interesting patterns, the demand for automated methods has emerged, largely due to the significant size of data. Tasks such as node clustering, KG completion, node classification, and link prediction are commonly used within different domains and a variety of approaches to tackle them exist. Machine learning proved to be a good solution to many of these tasks, as it has been shown to have successful applications in a variety of tasks such as image and video processing, natural language processing, text mining, and many others.

One of the interesting tasks on graphs is link prediction where researchers try to infer previously unknown relationships between entities in data [1]. Link prediction can be applied in many domains. For example, in social networks, where the goal is to suggest

possible friends for a user based on people they know, and in biological networks, where scientists try to discover relations between two biological entities, based on their relations with other biological entities. To perform such a task, one should use properties and existing links in data to locate semantic relationships between entities and use them as features for machine learning algorithms in order to learn and eventually predict unknown relationships between entities in the KG.

In this dissertation, we first introduce a framework to track the evolution of a KG, where it incrementally collects meta-data about instances and also the schema of KGs encoded in OWL. We evaluate our framework on various KGs and ontologies to show the advantages of using such a tool. The discovery of the schema and the schema-in-use is beneficial for any KGs, which might not have a well-defined schema. We explain how the framework compares a KG to its previous versions and gives a comprehensive report about its evolution. It highlights the statistics of the data and the main compartment of the KG to the user.

Then we present our link prediction approach, which samples a large KG and captures semantic relationships between the entities with minimum prior knowledge and human involvements. We use well-established benchmarks to evaluate our method and compare to results of similar graph embedding approaches as our baselines.

Further, we applied our method to a specific biological KG as a case study, where domain experts and external data sources evaluated the predictions. We created a KG based on ProKinO project about protein kinases and enriched it with external datasets focusing on under-studied (dark) protein kinases to predict dark kinase and pathway possible associations. Although RegPattern2Vec was initially designed to select the informative

part of the KG, it also performed better in the case that the relative subgraph is the whole KG, as shown in Chapter 5.

CHAPTER 2

BACKGROUND

In this chapter, we introduce some of the concepts and definitions that have been used through this work. Then, we move to explain our goals by defining the evolution of the knowledge graph and the link prediction task.

1.1 Ontologies

According to W3C[3] ontologies (vocabularies) are defined as:

"An ontology is a specification of a conceptualization in specific domain."

The conceptualization is the main component of representing knowledge. It can be seen as a simplified view of the entities, concepts, and their relations to each other in the domain of interest. The common ingredients of an ontology are Individuals, Classes, Attributes, Relations, Restrictions, Axioms, and others. Developing an ontology, enables reusability of the knowledge and sharing mutual thought among different disciplines, and machines and humans and give more flexibility to software to adjust to changes in underlying assumptions in domain knowledge. The two most popular standard languages to encode these specifications are RDF, RDFS, and OWL. They offer different levels of expressivity, as needed for the conceptualizations. To get a better understanding of these differences, we provide the definition of each of them in the following sections.

[3] https://www.w3.org/standards/semanticweb/ontology

2.1 Resource Description Framework [4] (RDF)

The simplest model to exchange data on the web is RDF. It is a graph model that uses Uniform Resource Identifiers (URI) to identify and distinguish the resources across the web. In a nutshell, RDF is a triple that consists of a subject, a predicate, and an object.

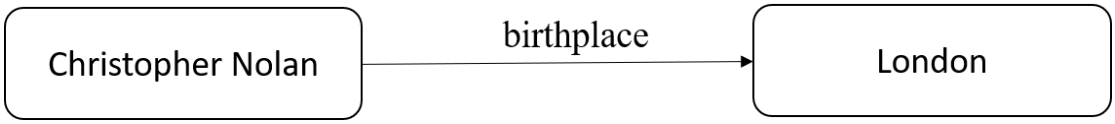

Figure 2. 1. A graph structure example of RDF for triple the (Christopher_Nolan, birthplace, London)

As shown in Figure 2.1, the example triple can be viewed as a directed edge in a graph structure, where the head node is the subject, the connection (edge) is the predicate, and the object is the tail node. In RDF, a subject should be either a URI, or a blank node, a predicate must be a URI, and an object can be in the form of a URI, literal, or a black node, which are referred to as RDF terms.

2.2 RDF Schema (RDFS)

RDFS is the vocabulary for RDF data to be modeled. It provides the ability to define classes, properties and to describe a related group of resources either as a hierarchy or an association to explain the RDF data. We can summarize RDFS into two groups of vocabulary terms: RDF classes and RDF properties. As shown in Figure 2.2, RDFS helps

[4] https://www.w3.org/TR/rdf11-concepts/

to provide more clear specifications to a document by providing characteristics of different terms defined in the document.

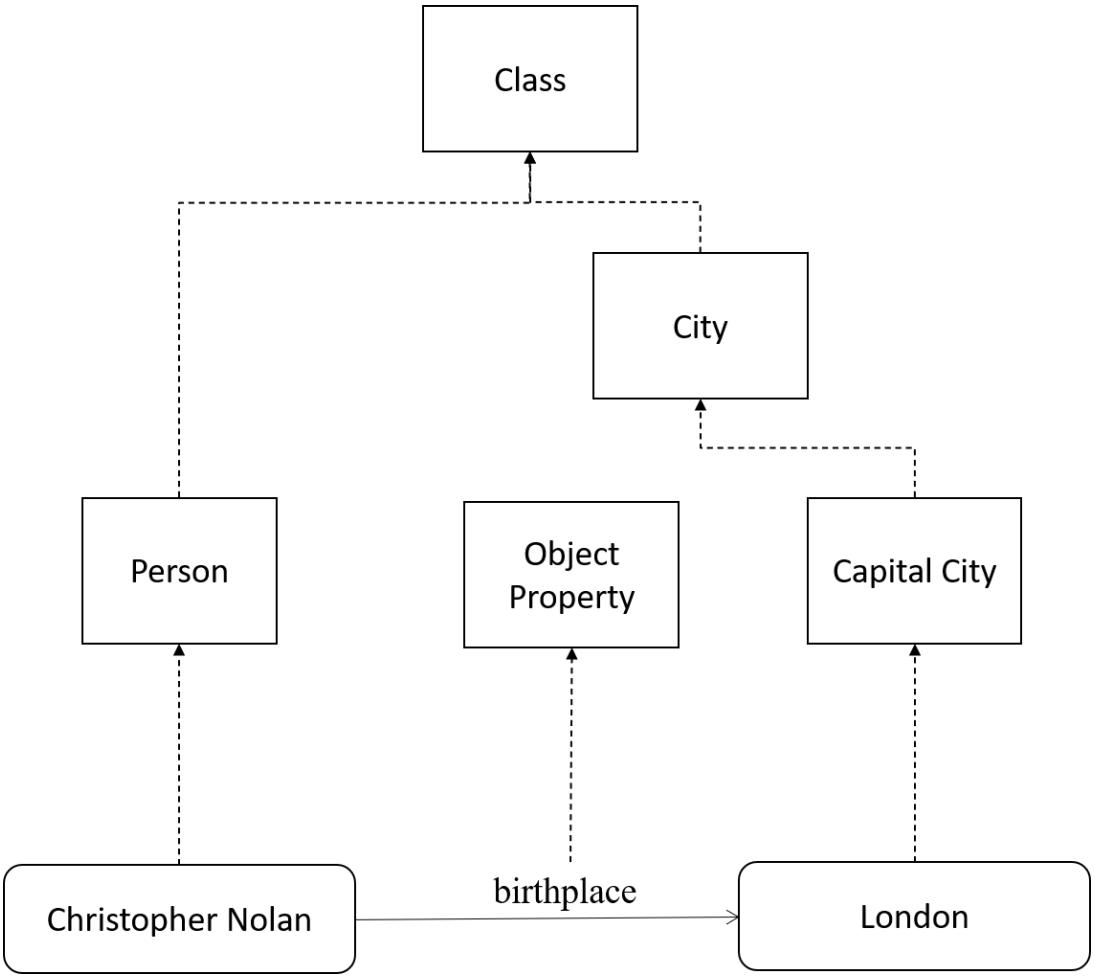

Figure 2. 2. An UML diagram showing the extension of semantic for RDF data using RDFS extension.

2.3 OWL

Web Ontology Language (OWL) is the extension to RDFS to enable formulating data expression for classes, properties, individuals, datatypes, and annotations. It has been used to formally provide the meaning to the Semantic Web [1]. Here is the formal definition of OWL by W3C[5]:

"OWL is a computational logic-based language such that knowledge expressed in OWL can be exploited by computer programs, e.g., to verify the consistency of that knowledge or to make implicit knowledge explicit."

2.4 SPARQL

SPARQL is the query language for RDF data. The basic building block of SPARQL queries is the triple pattern. A triple pattern is similar to an RDF triple, but in place of each triple element, one can place a variable. A triple pattern is then used to retrieve any matching number of the RDF data by matching triples in the data set to the triple pattern, substituting variables in the pattern by entities and properties found in the matched data triples.

[5] https://www.w3.org/OWL/#:~:text=Overview,things%2C%20and%20relations%20between%20things.

Figure 2.3. SPARQL query example and the result. a) example SPARQL query b) result of the query.

As illustrated in Figure 2.3, variables can be used in the triple patterns to retrieve the information needed. There are other query forms available in SPARQL, such as CONSTRUCT, ASK, and DESCRIBE, as well. The "WHERE" clause is where a triple pattern should be specified, and it will be used for matching against the RDF data. The namespace definition and also solution sequence modifiers (such as ORDER BY, LIMIT, etc.) are optional. The SPARQL syntax is straightforward, and it is the basis for other standard query syntaxes, as explained later.

2.5 Triple Stores

To store RDF data or triples, we need a data store that has been optimized for storage and retrieval of triples, and like relational databases, queries are used to store and retrieve data. Generally, triple stores are built to store and retrieve RDF data via semantic queries.

Some of the popular triple stores are AllegroGraph[6], OpenLink Virtuoso[7], Jena TDB[8], and the Oracle Graph database[9].

Note that, graph databases are more general stores as compared to purpose-built triple stores, where graph structure with nodes, edges are used to store the data. This might lead to faster lookups by using index-free adjacency structure. Most of the previously mentioned data stores are graph databases capable of storing RDF directly. Typically, graph databases, such Neo4j[10], do not accept RDF natively and SPARQL queries need to be converted to Cypher queries (Neo4j query language), beforehand. Recently, plugins to load RDF/RDFS into Neo4j became available [2] .

2.6 Knowledge Graphs

From 1972, when the term knowledge graph (KG) was first introduced, researchers focused on designing semantic networks to capture semantic relationships between entities in their specific domains and projects such as Wordnet [3] remained active until now. On the other hand, projects such as DBPedia [4] and Freebase [4] were designed for unspecialized knowledge in 2007 although they have never considered themselves as KG. But the term gained popularity in 2012, when Google introduced their Knowledge Graph [11] to incorporate the semantics in search engine, e.g., NELL[2] and YAGO[3]. There are multiple definitions for knowledge graphs in the literature but here we present the graph-based definition to better differentiate them from Heterogeneous Information Networks.

[6] https://franz.com/
[7] https://virtuoso.openlinksw.com/
[8] https://jena.apache.org/documentation/tdb/
[9] https://www.oracle.com/database/technologies/spatialandgraph.html
[10] https://neo4j.com/
[11] https://blog.google/products/search/introducing-knowledge-graph-things-not/

A knowledge graph (KG) is a directed graph G: (V, E) whose nodes $v_i \in V$ are entities and edges $e_i \in E$ are relations connecting the entities. For each node in V, we have a type mapping function $\phi : V \rightarrow T$, where T denotes a node type set, and edges have an associated type mapping function $\varphi : E \rightarrow R$ where R denotes a relation type set. Thus, a triple (v_i, e, v_j) forms an edge which implies the relation e between two nodes $v_i$and v_j.

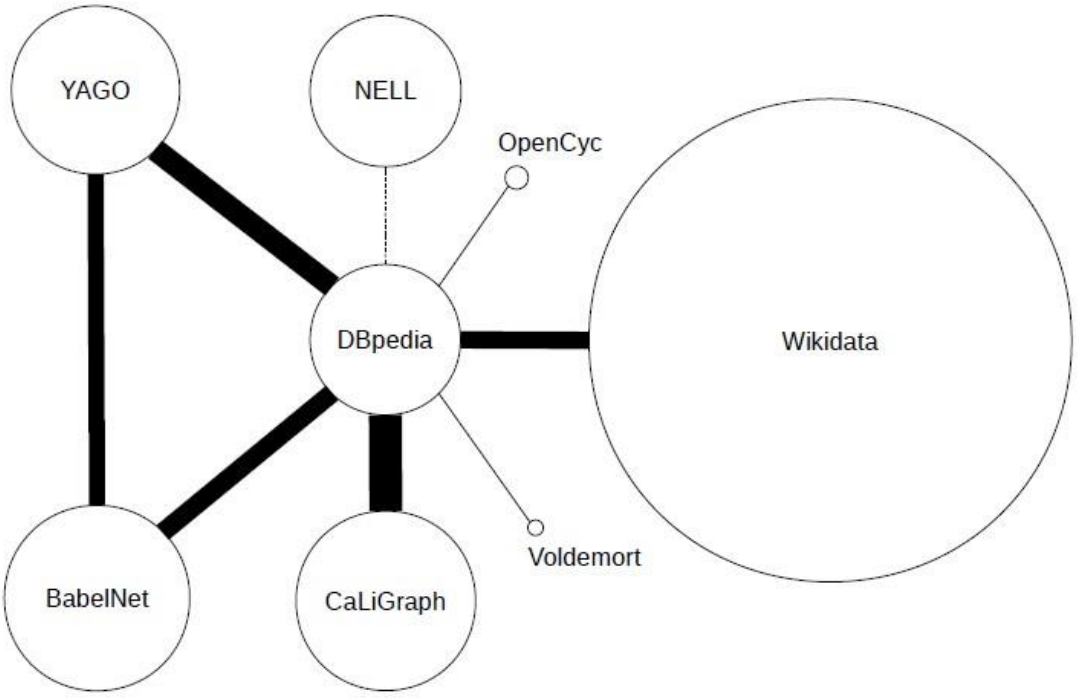

Figure 2.4. Size and linkage degree of publicly available knowledge graphs on the Web. [source ISBN:*9781643680811*]

2.7 KG Evolution

A critical consideration for a domain expert or knowledge engineer is how a KG evolves over time. Since most of the KG population software uses different datasets or even Web documents to generate/update a KG, it is necessary to validate the resulting KG before making it available to the users. Even in the most extreme cases, the changes

in those datasets may require software to be adjusted or modified, therefore a knowledge engineer needs to monitor such processes. The basic way is to check the logs of the software along with running some fixed number of queries to compare their results to the expected results manually.

On the other hand, domain experts need to be notified of the changes in the reference data sets when populating a new version. That is because the sciences are evolving each day, and monitoring these changes helps them to get a better understanding of the data and even make necessary decisions to utilize the information to their benefit. As mentioned before, querying the data is time-consuming and usually insufficient when dealing with large KGs.

Chapter three presents a method to automatically gather meta-information of different versions of KGs to monitor changes in the numbers of nodes and edges as well as structural changes and produce comprehensive reports.

2.8 Link Prediction

The link prediction task on a directed/undirected graph is defined as inferring the subset of missing relations in $time_{t+1}$ given a snapshot of the graph in $time_t$, where nodes in the graph represent entities and edges represent relationships between the entities [1]. There are different ways to formulate a link prediction task on a graph, but the most popular way is to express it as a ranking problem, where a threshold is set on the likelihood of the presence of edges between each pair of nodes. The most comprehensive taxonomy of different approaches for link prediction is provided by [5] and improved in [1], where the authors group the methods into four main categories: similarity-based

methods, probabilistic and statistical model, algorithmic methods, and preprocessing method.

Link prediction has a large number of applications in different domains, such as predicting protein-protein interactions in biology, social network analysis, entity resolution [5], and many more. In chapter 5, we present a case study in the biological domain, where the domain experts are interested in discovering unknown associations between proteins and pathways in which the proteins participate.

2.9 Knowledge discovery in KGs

As mentioned earlier, there are numerous knowledge graphs in different fields that contain hidden knowledge to be discovered. One primary way to achieve this is to use link prediction to propose new associations within the entities in KG. The variety of entities and edges between them make it hard to realize the schema of the KG. These entities and relations might come from other resources or even more general-purpose KGs or ontologies that have been used to populate the new KG and the schema of those resources might not be known. This makes the link prediction problem even harder and less efficient, sometimes less accurate.

Furthermore, the newly populated KG version might introduce changes in the schema and the instances of the data, and this causes the previously successful data mining methods to perform poorly on the new version of KG. Here is where the KGdiff, our method to track the evolution of KG comes to play. It discovers the schema and schema in-use of the KG and gives the better understanding of KG to the engineers and domain

experts and help them realize the overall schema and actual relations between different types of entities in KG to variety of purposes such as link prediction.

The RegPattern2Vec is a powerful tool for large KGs where their schema in-use is known to experts, and it takes advantage of the known schema to perform faster and more accurate by selecting the sub-graph from KG that is more informative for such prediction without expending lots of time tracing the whole KG. RegPattern2Vec has shown its very good performance on the gold-standard datasets, but we did not stop there. Many other methods perform well on these datasets but they are not able to perform as well in real-life tasks. To show that RegPattern2Vec is applicable to real-life datasets, we applied to a biological KG where it can predict association that are crucial in that domain. We have shown that some of the predictions have literature support when the associations were not known in our data and even propose new associations that are promising.

This dissertation proposes a knowledge discovery pipeline for domain experts where a huge data in a form of a KG is available and discovery of knowledge is desired.

CHAPTER 3

KGdiff: TRACKING THE EVOLUTION OF KNOWLEDGE GRAPHS[12]

[12] Abbas Keshavarzi and Krys Kochut. To be submitted to IEEE Transactions on Knowledge and Data Engineering Journal.
* Presented in 21st IEEE International Conference on Information Reuse and Integration for Data Science (IRI), 2020.

Abstract

A Knowledge Graph (KG) is a machine-readable, labeled graph-like representation of human knowledge. As the main goal of KG is to represent data by enriching it with computer-processable semantics, the knowledge graph creation usually involves acquiring data from external resources and datasets. In many domains, especially in biomedicine, the data sources continuously evolve, and KG engineers and domain experts must not only track the changes in KG entities and their interconnections but introduce changes to the KG schema and the graph population software. We present a framework to track the KG evolution both in terms of the schema and individuals. KGdiff is a software tool that incrementally collects the relevant meta-data information from a KG and compares it to a prior version the KG. The KG is represented in OWL/RDF/RDFS and the meta-data is collected using domain-independent queries. We evaluate our method on different RDF/OWL data sets (ontologies).

3.1 Introduction

Recently, knowledge graphs have gained considerable interest among researchers. They play an important role in various software systems, including recommendation systems, information retrieval, search engines, and many others. Therefore, creating, managing, and evaluating them is critically important.

A knowledge graph (KG) is a machine-readable, labeled graph-like representation of human knowledge. This definition is similar to that of ontology, as they share largely the same terminology and specifications, and they are used interchangeably in the literature. Often, they both use the Resource Description Framework (RDF) to store, transfer, and represent data. Broadly speaking, RDF is a set of triples, composed of a subject, predicate, and object. It offers a very simple, yet powerful way to make data easier to process, transfer and store by a variety of software systems. RDF Schema (RDFS) and the Web Ontology Language (OWL) can be used to create a schema for the knowledge graph data. In this regard, OWL is more expressive, which makes it possible to define many constraints and restrictions to closely follow the meaning of the KG data. Consequently, RDF/RDFS and OWL are some of the most popular specifications to represent both ontologies and knowledge graphs.

Despite the aforementioned similarity between ontologies and knowledge graphs, there are some fundamental differences between the two. In an ontology, the main focus is on conceptualizing a given domain as accurately as possible and so the accuracy of the knowledge modeling is at the forefront. An ontology should faithfully capture all of the concepts and relationships in a given domain and appropriately represent them in its

schema. Often, only a limited number of instances, if any, are included, often as an illustration.

On the other hand, knowledge graphs, while often including a schema as well, focus more on the data represented as individuals (graph nodes) and interconnections among them (graph edges). Usually, a major goal is to represent a large amount of data in a graph form with an aim to leverage the data in many tasks requiring a semantics-based approach. As previously mentioned, KG may be used to enhance graph-based search capabilities or to support the core functions of recommendation systems. A KG can be created using various resources, ranging from semi-structured to structured data, possibly including unstructured natural language text. The acquired data is often curated and interconnections are identified to form a knowledge graph. The resources may be evolving over time and the KG construction process should be flexible enough to accommodate the changes. Thus, a framework is needed to track the changes and offer the KG engineers a good grasp of the accumulated changes in the KG. In this work, we present a framework to track KG evolution from the perspective of both the schema and the individuals (instances). KGdiff retrieves meta-data and other important information about the knowledge graph, which can be analyzed by KG engineers and domain experts. Also, it can be used for comparison to other (past or future) versions of the same KG. KGdiff uses SPARQL endpoints to execute a number of SPARQL queries in order to get the necessary meta-data. Therefore, it does not need to load the whole KG to memory. The queries are independent of the knowledge graph schema and comprehensive enough to gather all of the necessary information for tracking KG version evolution.

3.2 Related work

Some of the original knowledge graph comparison methods were motivated by the UNIX diff command, but they have been proven to be unsuitable [6].

[7] is one of the pioneering works for ontology versioning. Although the main purpose was to match different versions of ontologies together, similarly to version control systems in software development, the authors managed to use different heuristic matchers to find a delta of two ontology versions. Another way to tackle the problem was reported in [8], where the authors convert each version of an ontology to a rooted directed acyclic graph and compare them to establish differences. Atomic changes such as addition and deletion were aggregated into more complex changes in [9]. Zeginis et al. [10] tried to find a set of changes that could transform a previous version of ontology into a newer one. Although these approaches sounded promising, in reality, their complexity of the types of reports render them unsuitable for a comprehensive comparison of KGs.

In [11], the authors categorize changes in OWL ontologies as effectual or ineffectual by applying their approach to 88 versions of the same ontology. The basic change operations in [12], along with a rule-based approach helped the authors to determine semantically relevant concepts in different versions. Similarity measures were used in [6] to detect semantic differences in versions of an OWL ontology along with syntactic differences using OWL syntax. [13] also used low-level changes and simple changes to detect more complex ontology changes, but it is not clear if the authors consider changes of the concepts or not. SPARQL select queries were used to retrieve simple changes from a Virtuoso server used as an RDF datastore. Some other approaches include [14], and an online ontology browser with a diff tool called Bubastis [14], which simply tracks the

changes in class entities within different versions of the same ontology. However, the comparison of classes in Bubastis (classes added, removed or changed) is not sufficient to track the evolution of an complex ontology or a knowledge graph.

In general, tracking changes in a knowledge graph evolution is rooted in area of network science, where measuring the structural similarity/distance provides a metric to tackle important challenges in graph analysis, such as graph matching, network comparison and network and ontology alignment. For example, in [15], the authors investigated approaches to determine the exact or inexact matches and used global and/or local measures to determine if the networks are deterministic or random graphs.

Most recently, in [16], the authors studied the evolution of a knowledge hierarchy using an approximation of Katz similarity measure to capture the concept hierarchy and relationship importance to track the evolution of a hierarchy in large knowledge graphs. In another work, researchers attempted to express changes in an ontology due to evolution based on an ontology log and to determine whether the revalidation of existing alignment is necessary or not [17].

Recently, graph databases become one of the options for storing Knowledge Graphs. Often, the evolution of graph data is considered as a tool for tracking the history of data, such as the recent work on the Neo4j[13] database [18]. The similarity of the RDF data to graph database data models makes it easy to store a KG in any graph database. Neo4j gained significant popularity because of its compatibility with Java and its ease of use. Its query language, Cypher, is easy to understand and has a lot in common with other well-established query languages such as SPARQL and SQL. Having no schema requirement in

[13] https://neo4j.com/neo4j-graph-database/

Neo4j databases is a plus, especially in the development phase, which imposes no restriction on structuring the data. However, in the production phase, there is a need for a schema in order to restrict the structure of the data stored in the database, typically, to avoid errors. The schema is also helpful in understanding the structure of the graph data when dealing it for the first time. Recently, some efforts in this regard have been made in the graph database community, including Data Profiling (using Cypher's built-in db.schema[14]) and Database Analyzer[15] . To the best of our knowledge, there are no efforts to analyze the evolution of a knowledge graph represented in a graph database. This offers an opportunity for us to expand our approach to graph databases, in the future.

3.3 Motivating Example

The Protein Kinase Ontology (ProKinO) [19], [20] is a Knowledge Graph, encoded in OWL, containing a large amount of comprehensive data on protein kinases. Protein kinases play an important role in many different types of cancer and have been a focus of intensive research. At present, ProKinO has 829 classes, 81 relationships (properties) and close to two million individuals. Its schema has been jointly developed by kinase scientists and ontology engineers and the included hierarchy of classes, object and datatype properties and other constructs define a comprehensive domain of knowledge on protein kinases. The ProKinO knowledge graph is automatically populated by custom-built

[14] https://neo4j.com/blog/data-profiling-holistic-view-neo4j
[15] https://medium.com/neo4j/introducing-the-neo4j-database-analyzer-a989b85e4026

software and uses a number of external data sources, such as COSMIC [21], UniProt [22], Reactome [23] and Kinbase. However, much of the data included in ProKinO has been created at the lab of Dr. Kannan at the University of Georgia.

The hierarchical structure of the ProKinO classes has been designed by kinase specialists and ontology engineers to closely follow the biologist's view of protein kinases, their structure and function. The kinase classification into groups, families and subfamilies has been established by kinase scientists at many institutions. However, classification of the pseudokinases has been created largely at the University of Georgia [24]. The ProKinO knowledge graph population software loads the schema and the kinase classification files and then populates the individuals and links among them using resources mentioned above to form the whole ProKinO knowledge graph.

The population process is repeated at regular intervals to keep ProKinO up-to-date (usually bi-monthly) as the data resources release new versions, regularly. Consequently, these regular updates result in new versions of the ProKinO knowledge graph. Obviously, as the resources change, the ProKinO versions continually accumulate differences, which are important both for the knowledge engineers and the domain experts who are the intended knowledge graph users. Changes in the individuals in various classes and the numbers of links between those individuals are crucial for the scientists, as they query the ProKinO data and perform various kinase analysis tasks. With each new version, the scientists are forced to execute numerous test queries or even manually verify the newly generated knowledge graph to discover how the new data changed in comparison the previous ProKinO version.

The ontology schema evolves over time, albeit not frequently, and the automatic knowledge graph population software must be modified, as well. Consequently, the knowledge graph engineers must evaluate the populated ProKinO Knowledge Graph and check the new version for accuracy and verify that the population process was completed correctly. The verification queries check if the numbers of instances in each class follow the general growth trends and if a change in a specific class or a property causes any issues for the populated data. These verification steps typically consume a significant amount of time and resources but are absolutely necessary to maintain high quality of various protein kinase analysis tasks. KGdiff has been designed as a comprehensive solution for monitoring the evolution of a Knowledge Graph. To the best of our knowledge, there are no other software tools available that are capable of monitoring the above concerns. As shown in Figure 3.1, KGdiff displays information about a single version of a KG and also the result of comparing two versions of a KG. We will present details about its design, implementation and capabilities in the following sections.

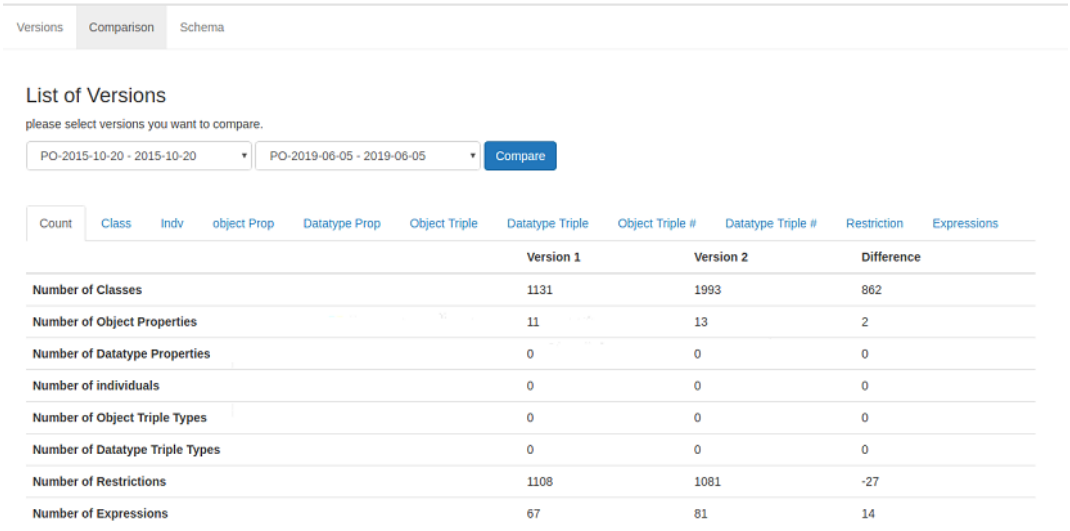

Figure 3. 1. Screenshot of KGdiff

To evaluate KGdiff further, we selected a set of knowledge graphs and ontologies in different domains. Then, using their different versions, we showed that the KGdiff is able to track knowledge graph evolution and can provide important information for the knowledge engineers and domain experts. We will discuss the experiments in the section on evaluation.

3.4 Difference of Knowledge Graph versions

KGdiff is a software system which identifies and reports changes in an evolving knowledge graph represented in RDF/RDFS or OWL. KGdiff is capable of identifying changes both in the KG structure (at the schema level, sometimes referred to as the TBox) and among the individuals and connections among them included in the graph (sometimes referred to as the ABox). However, identified changes among the individuals only refer to their counts, and not the actual individuals. The system provides a summary report in terms of the statistics about the number of modified and retained classes, object and datatype properties and other important concepts in a knowledge graph. Subsequently, the user can evaluate the changes as either expected, e.g., new classes and properties, or any typical increases in the numbers of class individuals, or unexpected, perhaps due to population process errors or unintended class or property modifications. The system is divided into two components: (1) the meta-data acquisition and (2) the graph difference evaluator. We assume that the knowledge graph is accessible by SPARQL endpoints, one for each KG version to be compared. At the end, a summary report is presented to the user.

The meta-data acquisition component executes a series of specific SPARQL queries against the two endpoints to acquire the necessary information for each version of the RDF/RDFS/OWL knowledge graph. Subsequently, the comprehensive meta-data

information about the graphs is stored in a MySQL database. The graph difference evaluator uses the meta-data information for two versions stored in the database and computes the differences between the versions. The final summary is prepared and presented to the use.

As the difference evaluator is focused on the differences between the versions of knowledge graphs, it does not report the complete meta-data for both versions. However, as a convenience, the system also allows the user to view the entire meta-data information identified for one KG version, which shows all graph entities (resources) and their statistics.

3.4.1 Meta-data discovery

The meta-data for a knowledge graph includes primarily all classes of the resource in the graph (nodes) and their hierarchical organization, relationships (properties) and their hierarchical organization, as well as all types of edges (patterns of triples) in the graph. For OWL-encoded knowledge graphs, it may also include class expressions and restrictions, as defined in the graph.

A fixed set of SPARQL queries is executed. Due to the space limitations, we are not showing them here, but an outline of one such query is shown in Figure 3.2. Classes are retrieved first. For each class in the graph, KGdiff obtains the class's URI, its labels (rdfs:label), parent classes (rdfs:subClassOf), if they exist, and the count of its individuals (instances). Here, we assume that individuals are classified using the rdf:type property. The number of individuals for a class includes only the direct (immediate) instances. For example, if class RoseWine has 100 instances, none of them are counted as instances of

```
SELECT DISTINCT ?domain ?property ?range WHERE {
        ?property a owl:ObjectProperty.
        ?subject ?property ?object.
        ?subject a ?domain.
        ?object a ?range.
}
ORDER BY ?domain ?property ?range
```

Figure 3. 2. A sample SPARQL query

the parent class Wine (if the Wine class has no direct instances, its number of instances is set to zero).

Object and datatype properties are discovered next. In RDF/RDFS and OWL they are first-class objects, and KGdiff collects their labels and parent properties (rdfs:subPropertyOf). Classes defined as set expressions on other classes are discovered, as well. OWL restrictions (including value and cardinality) are discovered, too, if present. After all the classes have been collected, KGdiff retrieves the edge (triple) types. For each triple, it determines the predicate, which is either an object property or a datatype property. Furthermore, it obtains the class of the triple's subject and the class of its object (for object properties), or the XSD type (for datatype properties). SPARQL queries also establish the counts of the instances for all triple types. Another important aspect of meta-data discovery concerns discovery of additional class descriptions[16].

[16] https://www.w3.org/TR/owl-ref/

3.4.2 Meta-data store

A relational database has been selected to store the retrieved meta-data information from Knowledge Graphs. Fig. 3.3 illustrates an outline of the database organization as a UML diagram. There are 19 classes and several associations created to capture all of the important graph elements. Note that this is only an outline of the meta data organization, and the actual database schema is considerably more comprehensive. For simplicity, the diagram does not provide association multiplicities and many other details.

The central classes are Class, ResourceType, and EdgeType. EdgeType represents all of the types of edges in a given Knowledge Graph and represents the property, the class of the subject and the class of the object (for object properties). Note that blank nodes (RDF resources without assigned URIs) can be used as subject or objects in edges. For graphs represented in OWL, ClassExpression captures classes defined using set expressions, including owl:unionOf, owl:intersectionOf and owl:complementOf. OWL restrictions are represented by the Restriction class, which can be either value or cardinality restrictions. Please note that a Restriction can be a part of a class set expression, as in a "classic" example of a the RedWine definition as an intersection of the class Wine and all individuals with the hasColor property restricted to color red[17].

3.4.3 Defined schema vs. schema "in-use"

RDF graphs do not have to have a defined schema in RDFS or OWL. In fact, an RDF knowledge graph may only include resources (individuals) and links among them in the

[17] https://www.w3.org/TR/owl-test/misc-000-guide

form of triples. First, KGdiff identifies the defined schema by retrieving the explicitly specified classes (rdfs:Class, owl:Class), properties (rdfs:Property, owl:DatatypeProperty, owl:ObjectProperty), class expressions, restrictions, and other constructs. In addition, KGdiff recognizes how object and datatype properties are used in a graph, where the ranges and domains have not been explicitly specified, effectively recognizing what we call the schema "in use".

Consequently, there are two main types of information that KGdiff gathers from an endpoint. In summary, a graph schema is retrieved by running SPARQL queries to retrieve all concepts specified as classes, object properties and datatype properties, their domains and ranges, expressions, restrictions, etc. Second, all edge types (triple types) which are not explicitly defined in the schema but occur in the graph are recognized and stored as ObjectEdges and DatatypeEdges, as well (see Figure 3.3 for an example).

The class of individuals participating as subjects (or objects) in any triple can be a subclass of the actual domain and range defined for the property. This means the class of the subject and object in an instantiated triple can be different from the domain and range defined for that property. For example, in the ProKinO ontology, the class Mutation has been defined as the domain for an object property named locatedIn, while the Motif class has been defined as the range. The Mutation class has multiple subclasses, including Insertion, Deletion, etc., and the actual individuals are populated from these classes, not the Mutation class. So, in this case, EdgTypes can be <Insertion, locatedIn, Motif>, <Deletion, locatedIn, Motif>, etc., instead of <Mutation, locatedIn, Motif> which was defined in the schema.

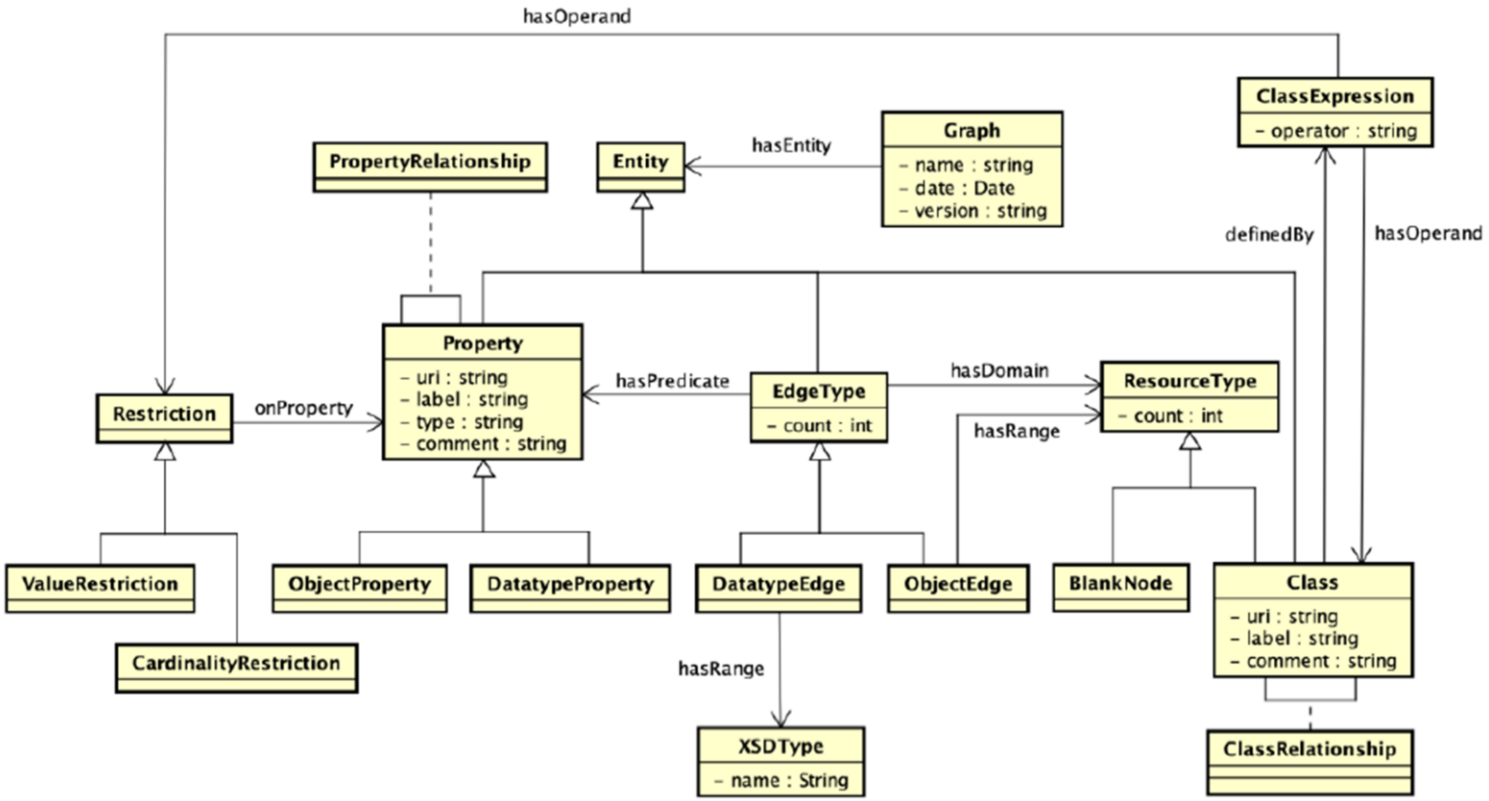

Figure 3. 3. Meta-data UML Class Diagram

Therefore, in evolving KGs, tracking the actual instances of the inferred schema is as important as the changes in the defined schema. KGdiff is able to retrieve all object and datatype edge types from an RDF or OWL knowledge graph and compare it to another version of the graph for verification and to detect any changes for further analysis.

3.4.4 Implementation

KGdiff is an open-source[18] web application coded in Java that uses both Apache Jena[19] and HTTP REST requests to retrieve a comprehensive knowledge graph meta-data, accessible from a SPARQL endpoint. It uses a SPARQL endpoint and the target graph name and executes a number of SPARQL queries against the endpoint to gather various aspects of the knowledge graph meta-data and store them in the database. Finally, the complete structure of restrictions and class expressions are identified using a recursive function and appropriate queries. Note that the restriction and class expressions are defined using blank nodes which are resources with no URIs and no defined classes. Each RDF/OWL datastore treats blank nodes differently. Although we gather information about them in the database, KGdiff does not count them as classes but represents them as a separate type.

As previously discussed, KGdiff requires Java, Jena, MySQL, Apache Tomcat[20] and can be used from a typical Web browser. Please note that Apache Tomcat can be easily

[18] https://github.com/abbask/KGdiff
[19] https://jena.apache.org
[20] http://tomcat.apache.org

replaced by a different Java-based application server. Similarly, another relational database accessible by JDBC can be used in place of MySQL.

3.5 Evaluation

We selected a number of ontologies and knowledge graphs in the domains of biology, geography, linguistics, and agriculture. We tested KGdiff using different versions of each knowledge graph. For some of the graphs, we used the published release notes or other forms of descriptions to determine what has been changed in comparison to the previous version. We used them to evaluate our results.

3.5.1 Datasets

We have collected five datasets, which will be discussed in more detail. For ProKinO, we obtained several versions and compared them with the results reported by KGdiff. Detailed change logs may not be available for some KGs, as they might be created and populated automatically by software systems. In such cases, KGdiff may be used to generate and record change logs for future reference.

The Ontology for Biomedical Investigations (OBI) [25] defines terms and protocols to describe an investigation in the biomedical domain. We chose OBI due to its size and the schema complexity. It can be used as a good benchmark to test our method's performance and correctness. OBI is a fairly large ontology, with numerous class and property definitions. Many of them are defined using complex expressions and restrictions using

the OWL vocabulary. We used different versions of the OBI graph available on BioPortal[21].

Plant Ontology (PO) links the anatomy of plants to genomic data by defining common vocabulary for anatomy, morphology, and development of plants. We have used two versions of PO for our experiment, versions 2015/10/20 and 2019/05/06[22].

To evaluate KGdiff further, we selected the GeoNames[23] ontology and compared versions 3.0 and 3.2, using the provided release notes as a reference.

Gold[24] is an ontology for descriptive linguistics. It is an effort to systematize the general knowledge in this field. For the evaluation purposes, we selected three different versions of this ontology.

The BioPortal uses a tool called Bubastis [14], which provides some brief description about what classes have been added and removed and the total number of differences between the current and previous versions of an ontology.

3.5.2 Results

Table 3.1 shows the results of KGdiff for datasets mentioned earlier. The counting for several aspect of the KG that KGdiff considers are listed for two different version of them. Along with these counts, KGdiff retrieves and shows separate reports, listing entities, numbers of instances and their parent(s). The differences are based on the changes in the entity's name, their parent entities, and their definitions. For this work, we considered

[21] https://bioportal.bioontology.org
[22] https://www.ebi.ac.uk/ols/ontologies/po
[23] http://www.geonames.org/ontology/documentation.html
[24] http://linguistics-ontology.org

classes, object properties, datatype properties, object edges, datatype edges, restrictions and class expressions. Others, including class axioms, functional properties and annotations are left for future work.

It is worth mentioning that an ontology such as OBI has a large number of classes defined by restrictions on other classes and object properties, and their definitions use blank nodes to represent them. Blank nodes play a special role of graph nodes and are treated differently in various RDF datastores. Since they do not have any definitions, they are not clear to ontology users. Consequently, we decided not to count them in overall statistics in the current version of KGdiff which is a practice follow by other tools such as Protege[25]. Each graph version was retrieved using its own endpoint and the meta-data information was stored in the database for further comparison. KGdiff has no limitation on the number of the processed graph versions. All versions are stored independently from the endpoints can be used offline. For the purpose of the evaluation, we selected only two versions of each KG to evaluate KGdiff.

[25] https://protege.stanford.edu/

Table 3. 1. Results by applying datasets to KGdiff.

Dataset	ProKinO		OBI		PO		GeoName		Gold	
Version	2019-10	2020-01	2018-02	2018-05	2015-10	2019-06	3.0	3.2	2008	2009
Classes	829	832	3343	3381	1131	1993	11	9	505	506
Object Properties	29	29	95	96	11	13	19	16	76	76
Data type Properties	51	53	8	8	0	0	13	16	7	7
Individuals	1787139	3133965	604	604	0	0	689	702	589	84
Object Triple Types	2537	3412	20	20	0	0	1	0	0	0
Data Type Triple Types	2254	2265	0	0	0	0	3	3	0	0
Restrictions	0	0	3992	8028	1108	1081	7	16	0	0
Expressions	6	7	1751	3524	67	81	0	0	0	0

3.5.3 Results evaluation

In the case of ProKinO, the changes detected by KGdiff were exactly as expected by the KG engineers. All of the changes in the new version of ProKinO were successfully identified by KGdiff and the correctness of KG population was verified. Obviously, such an exact change log is available for a KG population process managed by knowledge engineers, where some part of the process is semi-automatic. KGdiff is beneficial in verifying the resulting KG or ontology created by an automatic or semi-automatic population process. It shows a comparison of meta-data snapshots of the compared versions.

KGdiff is also very useful when analyzing large RDF and OWL knowledge graphs, such as OBI, with hundreds or thousands of classes and complex restrictions and class expressions. In our experiment, the comparison results produced by KGdiff showed the expected modifications, as compared to what has been provided in the change log. KGdiff identified many additional changes in class expressions and restrictions, which were verified by our analysis of the serialized ontology versions.

When a class (or a property) is replaced by a different class (or a property), it will not be reflected in the changed counts of entities, as previously discussed. For that reason, we have created an additional KGdiff module handling modifications for each entity types of concern. For instance, the Gene class in the ProKinO ontology has recently been replaced by the Protein class (to better model protein motifs and other specific information not associated with genes). KGdiff, in its classes view notifies he user that the Gene class, which existed in the previous version, does not exist in the new version of the ontology

graph, and that the Protein class did not exist in the previous version but exists in the new version. KGdiff identified and reported all such changes in ProKinO.

In the GeoName graph dataset, the newly added GeonamesFeature class and the geonamesID property were correctly identified along with all deleted entities. As mentioned previously, the changes to the parent properties are also detected by KGdiff. Table 1 shows the most important aspect of the changes in different versions of this ontology. Note that, due to the extensive changes in the feature code entity, the vast number of restrictions introduced in version 3.2, were skipped here due to the size limitations.

The versions of the GOLD ontology show a changing number of classes in versions 2008 (505) and 2009 (506). The class Greater Plural was removed and the classes Thing, Circumfix were added. Some other classes, such as Closed and Salience Propety were modified to Close and Salienc eProperty. The other observed changed between these two versions was that 505 individuals of type Class were removed from the ontology. The other release that we used was version 2010 (not shown here). We compared it to version 2009 to show the evolution of the ontology. 12 classes were removed namely Version, Proverb, Predicative, Small Paucal Number, Several Number, ConVerb ,FreeUnit, Morphological Unit, Inflectional Unit, Recent Tense, FunctionalUnit, DerivationalUnit and 8 new classes were added, including InflectionalMorpheme, BoundMorpheme, Complement, Morpheme, ProVerb, Converb, FreeMorpheme, and DerivationalMorpheme. Except for the ProVerb class that seems to have been just renamed, others seem to have been added/removed due to the change of concept. This shows that a domain expert can greatly benefit from this tool to recognize and understand the evolution of the ontology or track its changes over time.

The GOLD ontology does not include any individuals, so the triple types for the two graph versions in this experiment resulted in empty lists. We observed the growth in the number of classes from the 2015 to 2019 versions by 1131 classes (not shown here) and in the number of object properties by 11. The number of restrictions and expressions is also increased by 1108 and 67 respectively.

It is worth mentioning that the results from the Bubastis difference tool was different for this ontology. Bubastis reported that there were 329 classes that have been changed and 271 classes have been deleted, which indicates that the number of classes have been reduced over the period of four years. According to the Ontology Lookup Service, none of the classes have been deleted in this period of time. It might have occurred due to uses of external terms and definitions in Plant Ontology or, perhaps, Bubastis just uses the ontology namespace to find classes and is limited to the internal class definitions and does not identify all existing classes within the ontology.

3.6 Conclusions and future work

In this work, we presented a software solution, KGdiff, for tracking the evolution of knowledge graphs and ontologies. KGdiff uses generic SPARQL queries to extract all concepts and entities from an RDF/OWL graph and stores them in a relational database. The process of acquisition is a one-time process and the information gathered can be used to compare a graph version to its future versions as many times as necessary. The report consists of data about a variety of concepts in the knowledge graph along with the overview of its schema and schema “in-use”. The simplicity and interpretability of the results relieve the knowledge engineers of time-consuming manual version comparison. Using examples of knowledge graphs and sizeable ontologies, we have shown that this work accurately

identifies all of the changes in the schema and in the numbers of instances, which is an important part of the knowledge graph evolution processes.

Currently, KGdiff does not handle all of the OWL constructs. However, it focuses on the most important aspects of RDF/OWL knowledge graphs to help KG engineers and domain experts to verify the correctness of the ontology/KG population and to track the evolution of the graph. In the future, we plan to add the handling of the remaining OWL constructs, such as Annotations, Class Axioms and some of the property descriptions, such as owl:sameAs, owl:inverseOf, and others. Also, we plan to provide better visualization to help the users understand different aspect of the knowledge graph changes and provide more efficient ways to verify the data.

Another avenue for us is to explore the knowledge graphs that are stored in graph databases, instead of RDF or OWL, as many knowledge engineers turn to graph databases for knowledge graph representation. KGdiff can be adapted to identifying changes to KGs represented in graph databases (SPARQL queries would have to be replaced by suitable queries in Cypher, GSQL, or other graph database query language). Otherwise, much of the KGdiff software could be reused.

CHAPTER 4

RegPattern2Vec: LINK PREDICTION IN KNOWLEDGE GRAPHS[26]

[26] Abbas Keshavarzi, Natarajan Kannan and Krys Kochut. To be submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence.
* Presented in 2021 IEEE International IOT, Electronics, and Mechatronics Conference (IEMTRONICS).

Abstract

Link prediction is an important task in many domains, including health sciences, biology, recommender systems, social networks, and many more. It is one of the problems residing within the intersection of knowledge graphs and machine learning. Link prediction aims to discover unknown links between entities in a graph using various techniques. However, due to the size of knowledge graphs today and their complexity, it is a challenging and time-consuming task. In this work, we present RegPattern2Vec, a method to effectively sample a large knowledge graph to learn node embeddings, while capturing the semantic relationships between graph nodes with minimum prior knowledge and human involvement. Our results show that the link prediction using RegPattern2Vec outperforms related graph embedding approaches on large-scale and complex knowledge graphs.

4.1 Introduction

Today, knowledge graphs (KG) are gaining popularity in many domains. KGs can be stored and represented using standardized vocabularies, including Resource Description Framework (RDF), Web Ontology Language (OWL), and various graph databases that are rapidly gaining popularity, nowadays. Today, KGs often are very large and represent vast amounts of actionable data and researchers and computing practitioners have turned to graph data mining to leverage the KG data even further.

Recently, machine learning (ML) has been gaining popularity due to numerous successful applications in many different domains, including graph mining [26], image and

video processing [27], [28], text mining [29], reinforcement learning [30], and many others. It is also applicable to data mining of KGs. Software systems can automatically discover interesting patterns in KGs, while not being explicitly programmed to achieve that task. There are various types of learning algorithms, such as supervised and unsupervised learning, reinforcement learning, and many more. Classification and clustering are some of the most popular algorithms for machine learning on KGs. Experts take advantage of different algorithms to create recommendation systems for social media networks, entertainment libraries, find similarities in bibliographic networks, and many mor e.

KGs in nature are similar to Heterogeneous Information Networks (HIN) [31], where a variety of node and/or relation types between are used to represent data. This diversity of types provides benefits for learning, as compared to Homogenous Network, where types of nodes and relations are uniform. Finding appropriate methods to capture this extra information and use it in ML algorithms is a significant challenge.

Another major area, in the intersection of machine learning and KGs, is KG completion and, more specifically, link prediction. This problem has two aspects: predicting the missing links. KGs are usually populated automatically using variety of internal and external resources and due to the incompleteness of those resources, there might be some known links that are missing in populated KG. Here, one task of machine learning methods is to find those missing links and suggest that there should be connections between them.

As the graph analytics techniques are computationally expensive, especially on large graphs, researchers often aim to reduce the dimensionality of a graph into a low dimensional space. Graph embedding aims to preserve the structure of the graph while representing it as low dimensional vectors [32].

Based on [32], there are six different categories to generate vectors from a graph. These include matrix factorization, deep learning, edge reconstruction-based optimization, graph kernels, generative models, and hybrid models. In this work, we use a deep learning approach, in which random walks are used to sample the graph. This approach is based on a family of models from Natural Language Processing (NLP) called word2vec [33]; we specifically use the modified version of skip-gram model [34] in producing the vector embeddings. Skip-gram attempts to find the semantic similarity between words in a context by learning a meaningful representation for each word used in sentences in a corpus or documents. The main intuition is that we can discover the meaning of a word by understanding other words appearing close in a sentence. In the basic word2vec approach, the algorithm accepts a sentence and considers a window, usually of size 5 to 10, around the word of interest (center word) and generates training examples for a simple Neural Network (NN) with one hidden layer. The training examples are pairs of the center word and each of the words within the window size (context words). Then it trains the Neural Network to maximize the probability of a context word, given a center word. Then, the weights in the trained network are used as embeddings for each word in the corpus dictionary.

In this paper, we adopted this NLP method to our novel graph flattening approach using regular expressions to produce vector representation for the nodes in the graph. We formulate the link prediction as classification problem, using a model trained on the vector embeddings of the pairs of nodes connected by the link of interest. Our method, which we call RegPattern2Vec, shows high accuracy and discovers interesting possible links between unlinked nodes in the graph.

4.2 Related Work

Past research includes several approaches to capturing the semantic relationship between graph nodes. Matrix factorization-based methods generate embeddings by factoring the matrix that represents the relations between nodes [35]. The matrix can be an adjacency or a Laplacian matrix, among other methods. Another technique that is used to generate vector embeddings, is Graph Kernels. Graph kernels are a measure of similarity of pairs of graphs. For example, [36] uses graph kernels for subtrees and similarity of instances in the original graph by counting common structures. Intuitively, vector embeddings of nodes with similar structure in a subtree are closer to each other.

Generative models are also popular as a graph representation learning method. The generative and discriminative models play the minmax game, where the generator approximates the connectivity of a graph and a discriminator calculates the probability of edge's existence. They are used to perform link prediction and node classification [37]. Finally, we discuss two approaches that can be classified as Deep Learning (DL) methods. DLs using random walks, such as metapath2vec [34], and DLs not relying on random walks [38] and [39], utilizing other techniques for computing vector embeddings. Whether employing minimization of Margin-Based Ranking Loss for entities or constructing a multilayer graph with structural similarity of all nodes in level of hierarchy, their goal is to translate the graph to a low dimensional space, where it can be used for applications such as link prediction or node classification. It is worth mentioning that there are other techniques such as using Convolutional Networks [40] and Autoencoders [41], which we

do not discuss in this section because they fall into an entirely different type of methods. Generally, most GCN approaches suffer from scalability problems when the graph is large and dense due to the number of parameters and so are impractical to use. Therefore, different type of approach with a smaller number of parameters and hyperparameters will be required to large KGs such as approaches to sample the graph and learn representation in more efficient way, which will be discussed next.

Since in our method we rely on deep learning using random walks, we will focus on similar approaches in greater detail. As the computation of all possible walks on a graph is computationally expensive, researchers tend to choose random walks on the graph using some probability distribution. This would be sufficient for walks on homogenous networks. However, in heterogeneous networks with multiple types of nodes and edges, we need to differentiate types of nodes/edges when selecting the next node. Metapath2vec++ [34] is an approach that considers a fixed path of node types, which is called a meta-path. For instance, on a DBLP computer science bibliographic dataset [42], the meta-paths APA, APVPA, and OAPVPAO were chosen, where A represents the author, P paper, O the organization, and V the venue. These meta-paths are used to bias the random selection of the next node with the appropriate type in a random walk. Although some results of automatically discovering meta-paths have been published [43], usually domain experts are needed to choose the meta-paths of their interest for random walks. A domain expert should fully understand the KG organization. Although some tools for a KG schema discovery exist, such as KGdiff [44], due to the complexity of KGs and the hierarchy of concepts in them, it becomes difficult and time consuming to create appropriate meta-paths. Their selection should consider several aspects, e.g., the problem we want to solve

(node classification, clustering, etc.), either the selected subgraph or the whole KG, and the meta-path coverage within it (number of nodes that can be reached using meta-paths/meta-graphs). Although the meta-path approaches on HINs were often used for various tasks, they are not useful for capturing more complex relations among entities. Each type of node must be explicitly defined or the meta-path does not capture the variation of attributes linked to the nodes. [45] proposed meta-graphs, which in a nutshell are meaningful combinations of meta-paths. For instance, if there are two meta-paths as APA, AVA, a possible meta-graph would be A-[P/V]-A. The use of meta-graphs as constrained to random walks was tested in [46], but the choice of meta-graphs where they can improve the overall model poses another challenge. To overcome these weaknesses, we introduce RegPattern2Vec, where a regular expression guides the random walks to sample sequences of nodes in a more efficient way, especially for large KGs. where other methods fail due to the lack of scalability. The embeddings produced by our representation learning captures all of the necessary characteristics of each node to be used for high accuracy link prediction.

4.3 Preliminaries and Problem Definition

In this section, we first introduce some preliminary concepts and then define the problem of Link prediction on KGs using Random Walks constrained by Regular Expressions. As of this writing, a single, commonly accepted definition of a knowledge graph does not exist, yet, and many researchers provide their own definitions. A good

analysis is of KG definitions has been presented in [47], [48]. Here, we will use graph-based definition.

Knowledge graphs. A knowledge graph (KG) is a directed graph $\mathcal{G}$: $(\mathcal{V}, \mathcal{E})$ whose nodes $v_i \in \mathcal{V}$ are entities and edges $e_i \in \mathcal{E}$ are relations connecting the entities. Edges, usually referred to as triples of the form (v_i, e, v_j), represent some type of semantic dependence between the connected entities. Nodes have an associated type mapping function $\phi : \mathcal{V} \rightarrow \mathcal{T}$, where $\mathcal{T}$ denotes a node type, while edges have an associated type mapping function $\varphi : \mathcal{E} \rightarrow \mathcal{R}$ where $\mathcal{R}$ denotes a relation type set.

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 set of all such meta edges for G is called a *schema graph* (sometime referred to as *meta-template*). In fact, schema graph is a directed graph defined over node types $\mathcal{T}$, with edges from $\mathcal{R}$, denoted as $G_S = (\mathcal{T}, \mathcal{R})$ [49].

Knowledge graphs are often represented as RDF [50] datasets, where nodes (entities) and relationships are represented using Uniform Resource Identifiers (URI). Nodes and relationships have assigned types, given as URIs, as well. Furthermore, these types may form type hierarchies. RDFS [51] is often used to define a schema for an RDF knowledge graph. Knowledge graphs are closely related to Heterogeneous Information Networks (HIN). In HINs, object (node) and relationship *types* both contain more than one element, that is, there are multiple labels for graph nodes and multiple labels for edges. In case that type sets are singletons, the Information Network is called a Homogeneous Information Network (all nodes in the network are of the same type and all edges are of the same type).

Despite the obvious similarity of KGs and HINs, there exist important differences between them. An important distinction is that in HINs, a relation of type R ∈ $\mathcal{R}$ uniquely determines the types of source and target nodes that can be connected by the relation R. In knowledge graphs, however, a relation of type R may connect nodes of many different source types and target types. Many other differences KGs and HINs exist, but they are not important for the research presented in this paper.

It has been shown that the results of various graph-embedding tasks are sensitive to the selection of a specific meta-paths [52]. In this paper, we propose a method of using regular expressions as a specification of a wide range of semantic relationships to be incorporated in random walks.

Regular Patterns on KGs. Let G be a knowledge graph, $G = (\mathcal{V}, \mathcal{E})$, with a node type mapping function $\phi : \mathcal{V} \rightarrow \mathcal{T}$ and an edge type mapping function $\varphi : \mathcal{E} \rightarrow \mathcal{R}$. A Regular Pattern on G is a regular expression (pattern) [49] r formed over either set $\mathcal{T}$ or $\mathcal{R}$ as the alphabet.

We will not formally define regular expressions, since they are commonly used in computing, today. Briefly, a regular expression defines a set of strings (sequences, or words) over an alphabet; it defines a regular set [49]. We assume a standard format of regular expressions used in many programming languages today, for example in Python [53]. Here, we will only use a subset of possible regular expression constructs, including the concatenation, the alternative (|), repetitions of zero or more times (the Kleene star *), one or more times (+), specified number of times ({n, m} n through m, and {n,} at least n), and the complement matching [^xy] (any symbol other than x or y). Note that a meta-path, as defined in the metapath2vec algorithm [34], can be regarded as a regular expression over

$\mathcal{T}$ (or $\mathcal{R}$) since it can be regarded as a concatenation of node (or relation) types placed on the meta-path (again, a relation in HINs uniquely determines the source and target node types and vice-versa).

As an example, given node types $T_i \in \mathcal{T}$ in a KG, we could formulate a variety regular patterns over node types, for example, $T_1\ T_2\ T_3$, $T_1\ (T_2 \mid T_3)\ T_4$, $(T_1 \mid T_2)\ T_3{+}\ T_4$, any many others. Similarly, given edge types $E_i \in \mathcal{R}$ in a KG, we could create regular patterns over edge types, such as $[\hat{}R_1]\ R_2\ R_3{*}\ R_4$ or $R_1\ (R_2 \mid R_3){*}\ R_4$. Intuitively, a regular pattern defines a set of node (or edge) type sequences (a regular set), which we use to bias random walks on a KG to follow semantically relevant data.

Many knowledge graphs utilize complex hierarchies of node (entity) types such as Yago[54], DBpedia[55], and NELL[56]. Consequently, defining regular patterns based on node types is impractical, as they would require costly type inference (node types in actual sampled walks could be subtypes of those included in the defined regular pattern). Hence, using regular patterns on edge types may be a better choice.

In this paper, we focus on link prediction and so we rely on a specific general format of regular patterns for biasing the random walks. Assume that given a KG, we need to predict an edge (h, r, t), where $h, t \in \mathcal{V}$, $r \in \mathcal{E}$, $\phi(h) = H$, $\phi(t) = T$, and $\varphi(r) = R$. If the KG has a simple (non-hierarchical) structure of node types, our expression pattern can be based on node types and have a general format $\boldsymbol{H}[\hat{}\boldsymbol{T}]{+}\ \boldsymbol{H}\ \boldsymbol{T}$. However, in a KG with a large node type hierarchy, we use edge-based patterns with the general format of $[\hat{}R]\{2,\}\ R$. That is, at least 2 edges with relation types different than the one to be predicted followed by the edge relation type to be predicted.

The intuition behind the above regular (expression) pattern formats primarily comes from the observations of meta-paths and meta-graphs, where the similarity of two nodes is calculated based on the number of paths between them that follow a specific meta-path [57]. While some works [34], [46], [58] use symmetric meta-paths to calculate similarity between nodes of the same type, others [59] use more complex meta-paths for a different types of nodes. RegPattern2Vec follows the latter idea of finding the similarity of nodes with different types, but meta-paths and meta-graphs must be explicitly designed by domain experts and each such meta-path needs to be used in a separate experiment. In general, individual meta-paths cannot capture all possible semantically relevant connections between the nodes of interest. RegPattern2Vec, due to its use of regular patterns cover a large set of meta-path-like connections and takes advantage of a multitude of such semantic connections in one experiment.

We can explain the RegPattern2Vec using a simple example shown in Figure 4.1. The graph contains 4 different relations: R1, R2 R3, and R4 (red, green, blue and yellow, respectively). The link that we want to predict is R2. Following the regular pattern, [^R2]{2,}R2, we have R1, R3, R4 ∈ [^R2]. An example walk following the pattern is $a1 \overset{R1}{\rightarrow} c1 \overset{R1}{\rightarrow} a2 \overset{R2}{\rightarrow} d1$. The intuition is that if we found two nodes (a1 and a2) where they link to another common node (c1), they are semantically related and node a1 might have the relationship R2 to d1, as well, which is useful for the link prediction task.

In the example nodes a1 and a2 have two common nodes b1 and c1 and plus the above path, $a1 \overset{R4}{\rightarrow} b1 \overset{R4}{\rightarrow} a2 \overset{R2}{\rightarrow} d1$ is also allowed based on the regular pattern. It is possible that the intermediate nodes have a relationship with other nodes, such as nodes c1, c2, and c3. These links might be considered as loops within the same type of nodes. If the nodes are the same or different type, a regular pattern can find such path and there is hyperparameter to control number of such possible loop. If this parameter is 1 the path $a1 \overset{R1}{\rightarrow} c1 \overset{R3}{\rightarrow} c2 \overset{R1}{\rightarrow} a3 \overset{R2}{\rightarrow} d2$ is allowed and if it was 2, a walk from node c1 can reach node c3 and the path $a1 \overset{R1}{\rightarrow} c1 \overset{R3}{\rightarrow} c2 \overset{R3}{\rightarrow} c3 \overset{R1}{\rightarrow} a4 \overset{R2}{\rightarrow} d3$ is also permitted. Here, a1 and a2 are more similar than a1 and a4 but it can be beneficial to capture those paths, as well. By this logic, a random walk constrained by a regular pattern can reach different paths to capture more links within the graph, if necessary.

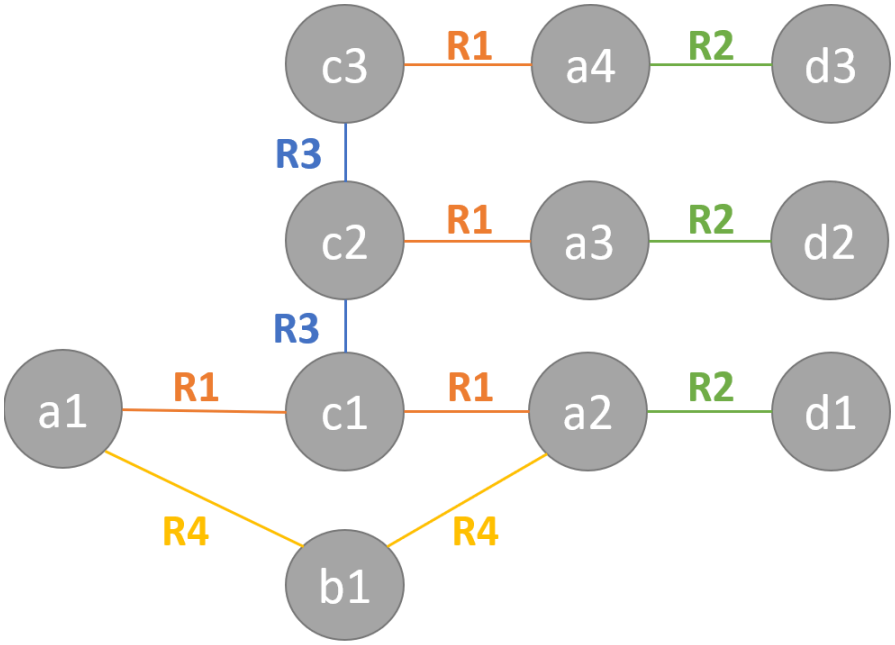

Figure 4. 1. Illustration of random walks using regular expression.

4.4 RegPattern2Vec

RegPattern2Vec relies on random walks to produce graph embeddings. Random walks on knowledge graphs are constrained to those matching a defined regular pattern.

4.4.1 Random Walks

Random walks in RegPattern2Vec are designed to sample an arbitrary number of walks. Their number can be controlled by parameters, such as “walk length” and “number of walks” (per starting node). Even though the knowledge graphs we use are defined as directed graphs, here, we treat them as undirected. We do this to be able to sample paths from all possible paths according to a defined distribution. Having an undirected knowledge graph and a regular expression pattern, a random walk can be started from any instance of the starting edge (or node) type in the pattern.

A regular pattern is converted to an equivalent Deterministic Finite Automaton (DFA) M [49] with the same input alphabet as the one used in the regular expression pattern definition. We will not introduce a formal definition of a DFA here but simply state that a DFA has a finite set of states, an input alphabet, a transition function δ, a starting state, and a set of final states. In our case, the state transition function is defined as $\delta: S\times\mathcal{R} \rightarrow S$ (or $\delta: S\times\mathcal{T} \rightarrow S$) and specifies state transitions based on edge (or node) types, depending on the regular pattern expression. We assume that δ is a partial function and for some states, transitions on some relation (or node) types may be undefined. We use the transition function δ of M to define the probabilities of node selections in our random walks.

It is obvious that if we repeat the walk from each node, we will discover more paths as the node might link to multiple nodes, which are allowed based on the regular pattern. We will call this parameter "number of walks". We will discuss how to choose the parameter and analysis of their impacts in the next section. As in each step there might be multiple choices, randomization will help the random walk to select a next node in each step. In the scale-free networks, where the degree distribution follows the power law, some nodes, often referred to as hubs, have high number of incoming/outgoing edges. It seems that such nodes would dominate the random walks as they have a higher probability of being reached when the next node selection follows the normal distribution. As the frequency of nodes appeared in the walk is the key point of representation learning, we bias the walks while using a regular expression pattern and its equivalent DFA *M*, using the formula below:

$$P\left(v^{i+1} \mid v^i, M\right) = \begin{cases} \dfrac{\frac{1}{\left|N_{v^{i+1}}\right|}}{\sum_{v \in N_{v^i}} \frac{1}{|N_v|}} & \left(v^i, r^i, v^{i+1}\right) \in \mathcal{E}, M \text{ is in state } s_i \text{ and} \\ & \delta\left(s_i, \varphi(r^i)\right) = s_{i+1} \\ 0 & \left(v^i, r^i, v^{i+1}\right) \in \mathcal{E}, \text{but } \delta\left(s_i, \varphi(r^i)\right) \text{ is undefined} \\ 0 & \left(v^i, r^i, v^{i+1}\right) \notin \mathcal{E} \end{cases}$$

Here, $|N_v|$ is the of degree of node v, v^i indicates the current node and $v^{i+1} \in N_{v^i}$ is the next candidate node, where N_{v^i} is the entire neighborhood of node v^i. Furthermore, s_i is the current state of *M*, that is, after processing the sequence of edge types $\varphi(r^1) \ldots \varphi(r^i)$, and *M*'s transition function from state s_i on edge type $\varphi(r^i)$ is defined and leads to state s_{i+1}. We can easily create a similar formula using a DFA *M* with node types, instead of relation types, as shown in (1). By fine tuning the previously mentioned parameters, this probability distribution will

be sufficient to reach as many nodes as possible (of reachable nodes) to be included in walks, which results in more accurate vector embeddings.

4.4.2 Representation Learning

RegPattern2Vec converts the graph into the sequences of nodes and, from this point, we treat the nodes as words in sentences, as produced by random walks. These sentences are used as input to a model, similar to the one used in metapth2vec++ [9], for generating node embeddings. This model is an improved version of the original skip-gram model, as it takes into consideration types of edges (nodes). This allows the embeddings to capture the similarity of edges (nodes) based on their types (often considered as classes) along with their appearance of closely connected nodes, as required by the pattern.

4.4.3 Link Prediction

RegPattern2Vec formulates Link Prediction in KGs as a classification problem. Each existing link (or edge) of interest is represented as a vector of real numbers and is treated as a positive example for training the model. We can combine two vectors using Hadamard product and used the resulted vector as features for machine learning algorithm with label as positive. As the negative examples are typically not included in knowledge graphs, we create combinations of pairs of nodes that are not connected by edges in the graph and use them as negative examples, which as a common approach in the published research in this area. RegPattern2Vec uses an element-wise multiplication of vectors as the combination

operation, which transforms the pairs of nodes to another space. These examples are used to train a classification model, such as Logistic Regression, which can be used for link prediction.

4.5 Experiments

4.5.1 Datasets

In our experiments, we used two popular datasets, YAGO39K [35] and NELL [5]. YAGO39K contains a subset of the YAGO knowledge base [29], which includes data extracted from Wikipedia, WordNet and GeoNames. This subset contains 123,182 unique entities (nodes) and 1,084,040 edges, using 37 different relation types. A histogram of relation type distribution in the YAGO39K dataset is shown in Figure 4.2. NELL is a knowledge graph mined from millions of web documents and contains 49,869 unique nodes, 296,013 edges, using 827 relation types. In contrast to the Heterogeneous Information Networks, both datasets include many edges with the same relation type connecting nodes with many source types and/or many target node types.

4.5.2 Link Prediction Experiments

Following the work on link prediction on the YAGO dataset [35], we chose three different relation types namely isLocatedIn, isCitizenOf, and isLeaderOf. Based on the relation to be predicted, we split the KG data, as reported in Table 4.1. We need to extract some number of edges from each of the three types into three different test sets for three

different tasks. To do so, we utilized minimum spanning tree to capture the minimum number of nodes that can be added to the test set while having the nodes in the training set. It is necessary, because our method requires that the node exist in the training data, although the node does not necessarily need to have the edge of interest in the training set. However, it can have other relations with other types of nodes. So, for each task, we extract the maximum number of edges of interest from the graph as the test set, while the remaining edges of interest and instances of other relations are combined to form the training data.

Table 4. 1. Statistics of split of data for different experiments

	isLocatedIn	isCitizenOf	IsLeaderOf
Train Set Graph	1,039,499	1,080,570	1,083,079
Train set edges	44,542	3,128	855
Test set edges	44,541	342	106

We have already discussed how to use the edge of interest in creating the training data with positive and negative examples for a binary classification model. We can follow the same process to make examples for testing in order to evaluate the performance of our method. Following the work described in [5], we chose two relations *CompetesWith* and *playsAgainst* for our link prediction experiments with the NELL dataset. The cited work reported the best metapaths used to predict these relations and

used these metapaths for comparison. We performed a similar process (as described above) to split the data into a train and test sets.

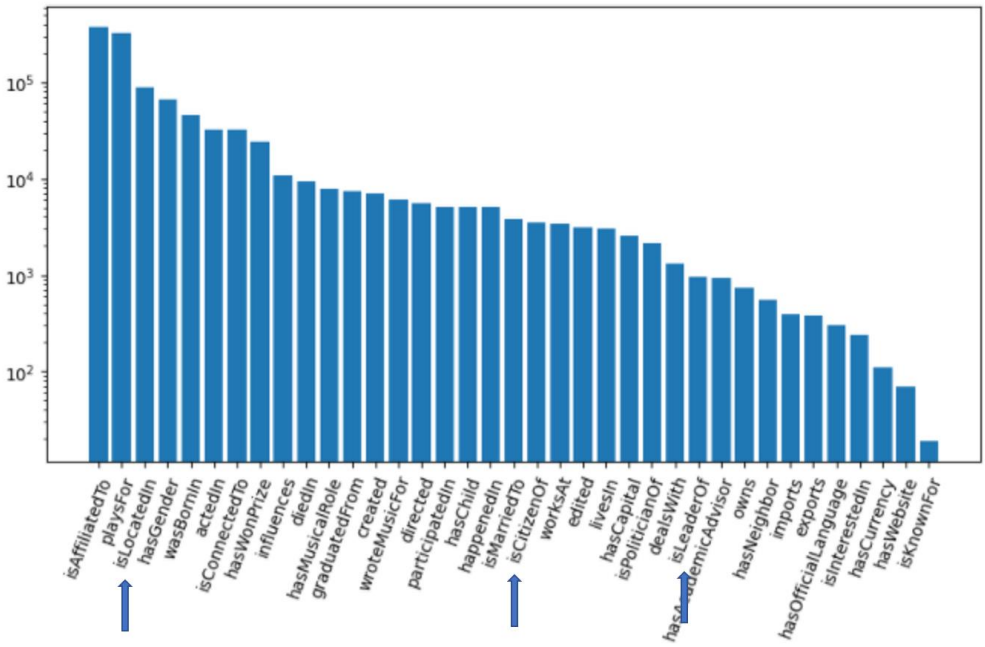

Figure 4. 2. Distribution of relation types in the YAGO39K dataset.

4.5.3 Performance

Having approximately a balance training set we train a logistic regression for binary classification. We evaluate our model using 10-fold cross-validation and test it using unseen set that we extracted from the graph.

To evaluate the performance of RegPattern2Vec on YAGO39, we chose metapath2vec as our baseline, shown in Figure 4.3, to demonstrate that RegPattern2Vec can cover more meta-paths without explicitly defining them and perform better, too. To get the best meta-paths for *isLocatedIn* and *isCitizenOf* relations, we chose the ones that achieved the best scores reported in the literature. However, were not able to find the best metapaths for

isLeaderOf, and we designed them ourselves. After our experiments, Person $\xrightarrow{\text{isLeaderOf}}$ city $\xrightarrow{\text{isLocatedIn}}$ country $\xleftarrow{\text{isLocated}}$ city was the best meta-path. For example, a leader of state, is specified as leader of cities with the state. That information suggesting the earlier meta-paths to perform better than any other meta-paths. The regular pattern for three aforementioned links were defined as follow:

$$[\hat{}\,isLeaderOf\{2,\}\,isLeaderOf]$$

$$[\hat{}isLocatedIn]\{2,\}\,isLocatedIn]$$

$$[\hat{}isLocatedIn]\{2,\}\,isLocatedIn]$$

To run the experiments, we kept the same parameter settings for each of the method, when working on a specific relation prediction. The settings included the number of walks from each node, the maximum walk lengths, the Logistic Regression parameters, and the metrics to evaluate their performance. To select the best algorithm for binary classification we examine two famous and popular algorithms, Logistic Regression and Random Forest, and we tested several experiments with both, and it seems that logistic Regression in our case the best performing method.

Therefore, all the experiments are performed with logistic Regression for evaluation purposes. RegPattern2Vec shown superior performance over metapath2vec method with best meta-paths possible. It is expected that RegPattern2Vec would perform better in link prediction tasks where it can obtain more semantics by exploring different path within the knowledge graph. In the case of the *isLeaderOf* relation, as the data does not contain information useful for predicting this relation, the performance is lower than for other relations.

For the evaluation of RegPattern2Vec on NELL datasets, we chose metapath2vec and used the best meta-paths reported in [30] for each of the relations. For *CompetesWith*, there were top five meta-paths. Our experiment showed that *HeadQuarteredIn* performs the best although it was ranked as the third in top 5. And for the *PlaysAgainst* relation, we selected two best meta-paths for evaluation purposes. We used the following regular expression patterns:

$$[\hat{}CompetesWith]\{2,\}\,CompetesWith$$

$$[\hat{}PlaysAgainst]\{2,\}\,PlaysAgainst$$

As shown in Figure. 4.3, RegPattern2Vec outperforms metapath2vec with different meta-paths for both relations in correctly predicting the unseen links between different nodes. The ROC shows that across most of the threshold the performance of RegPattern2Vec is significantly higher than metapath2vec with different metapaths.

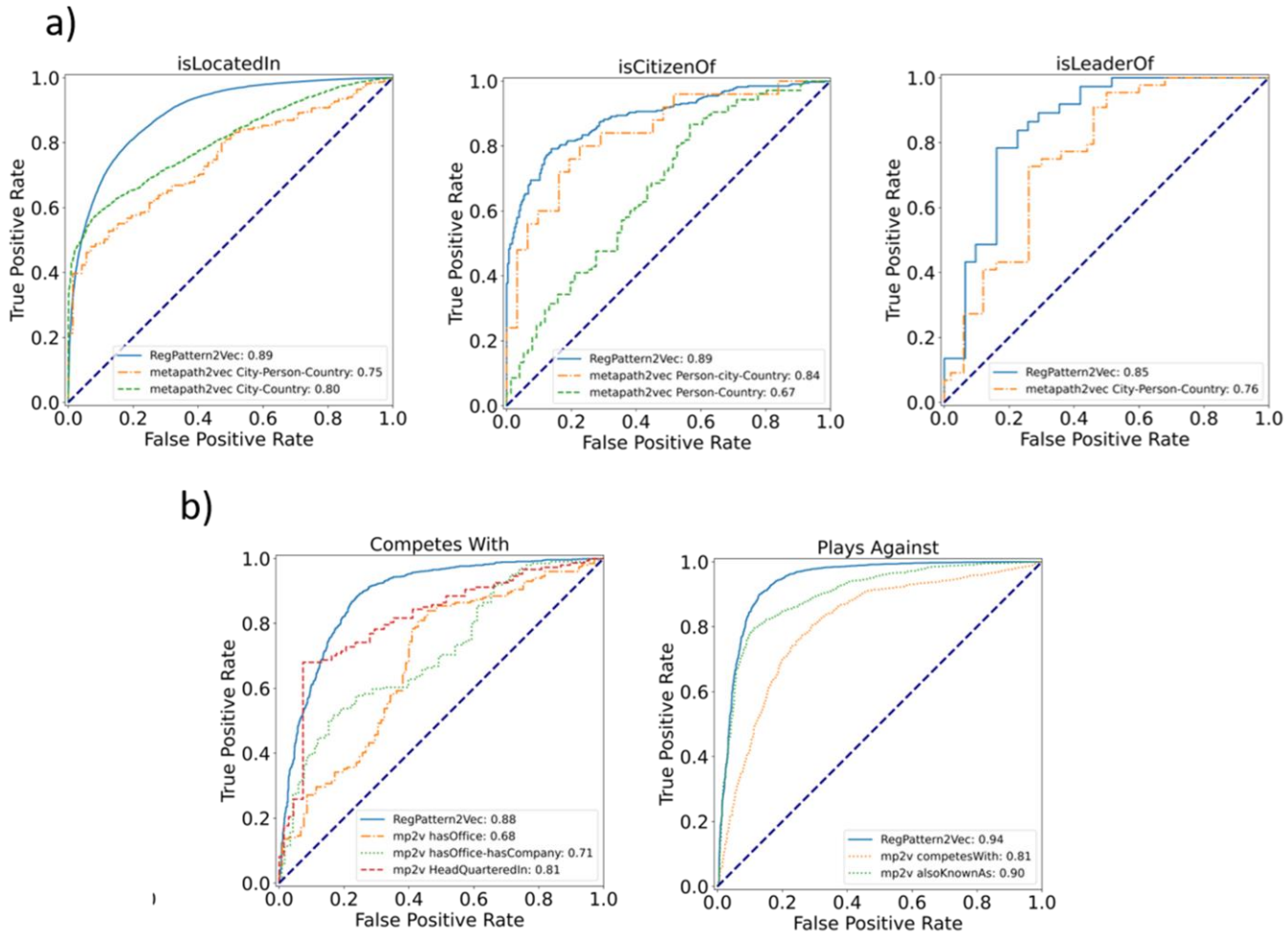

Figure 4. 3. Comparing ROC of RegPattern2Vec with baseline on two datasets. a) Three relations on YAGO39K b) Two relations on NELL

4.5.4 Patterns Discovered of Random walk guided by Regular Pattern

Based on the dataset and underlying schema, RegPattern2Vec discovers different patterns in the data and uses them to accurately predict possible links in the KG. Figure 4.4 shows the top 30 frequent meta-paths capture with RegPattern2Vec without explicitly specifying them. Although the number of possible relation sequences is very high, especially allowing for repetitions, some of the sequences can be seen frequently, based on the graph and they might significantly influence the vector embeddings. So, it is important to have a way to capture most of the patterns in the graph and consequently all path instances to allow the representation learning model to produce more accurate embeddings. This cannot be achieved by meta-paths as prior knowledge is needed to design each meta-path and they can be easily missed, especially when the schema of KG is unambiguous or not well-defined. Although just the frequency of each pattern in the data does not provide a good measure of how the pattern is important or useful for the link prediction problem, it can provide information about the graph itself and what are the frequent patterns in the graph. Then, the representation learning model can decide how frequent two specific nodes appear in the same context and provide a closer embedding for each of them, based on their local structures and neighbors.

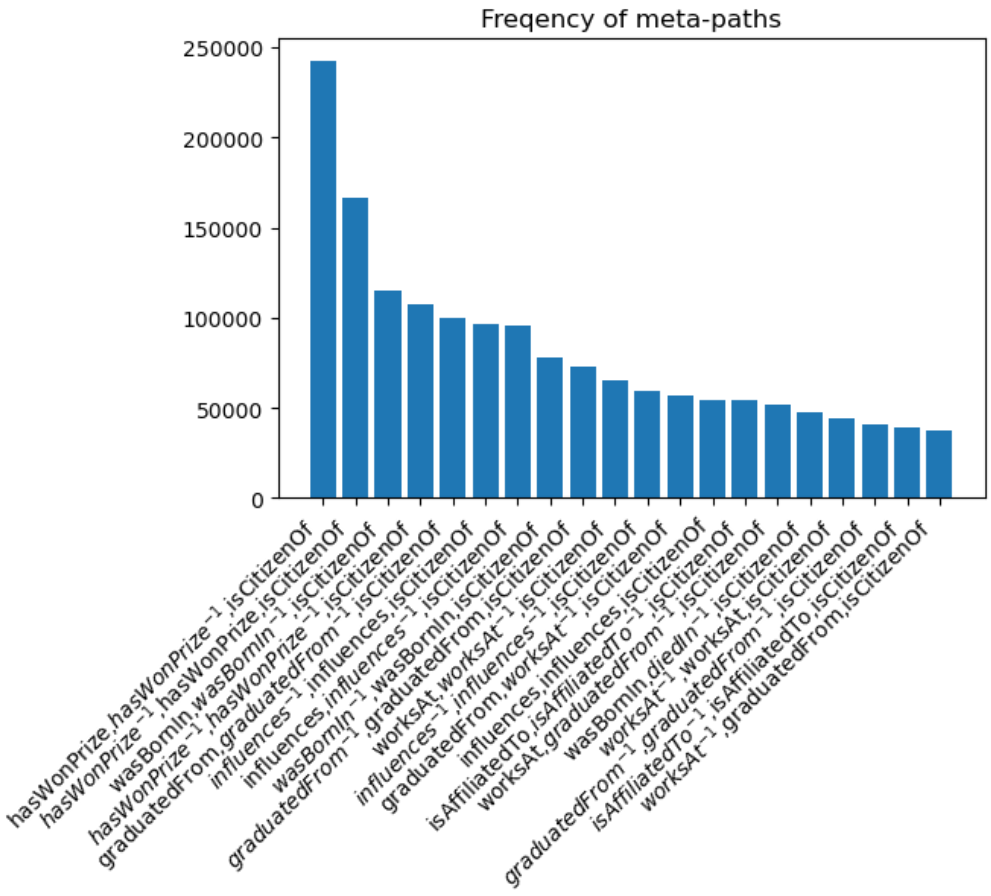

Figure 4. 4. The top 30 most frequent relationship patterns discovered by RegPattern2Vec for isCitizenOf relation.

Effect of Hyper-parameters

In this section we studied the effect of two important hyper-parameters in the random walks on performance and elapse time namely the number of walk and length of walks. The effect of different choices of these two on the AUC ROC is demonstrated in Fig. 4.5a for *playsAgainst* relation on NELL dataset. As more nodes are connected to each other, we need to sample more paths by increasing the number of walks or walk length. On the other side, due to the large size of KGs, one of the challenges of learning the embeddings is

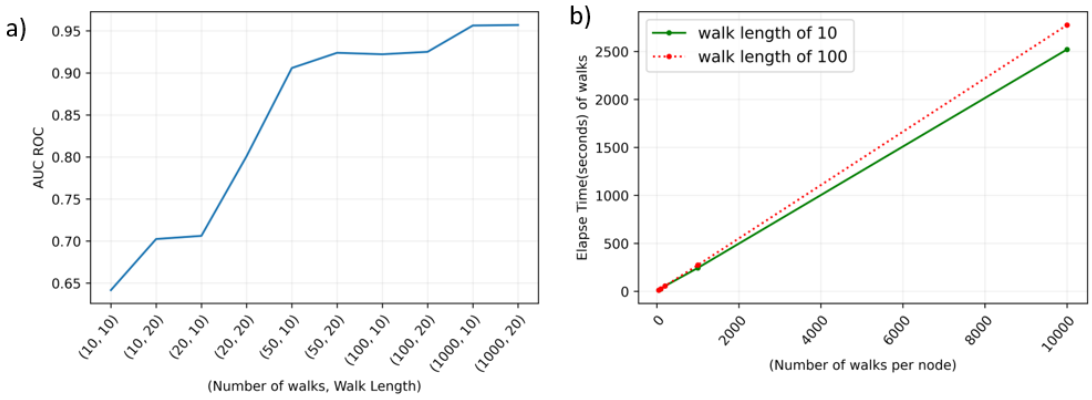

Figure 4. 5. a) Effect of number of walks and walk length on AUC ROC b) Effect of number of walks per node on elapsed time of random walks with two different walk lengths 10 and 100.

scalability and efficiency. we showed that increasing the number of walks can improve the performance, as the random walks are able to trace more paths in the data. Figure 4.5b shows how the increase of this parameter affects the elapsed time of the random walk, in this case when experimenting with the *competesWith* relation on the NELL dataset in two cases of walk length 10, and 100. As demonstrated the elapsed time is linearly related to the hyper-parameters.

4.6 Conclusion and Future work

In this work, we presented RegPattern2Vec, where a regular pattern guides the random walks in a knowledge graph to efficiently sample sequences of nodes to learn high quality embeddings for link prediction. We demonstrated link prediction using relation types, where the schema of knowledge graph is unknown, or node type hierarchy is complex. As a future direction for our work, we want to explore how to bias the random walks to favor the nodes or relations that might contribute more accurate link prediction results. Also, if the most frequent relations found in the patterns of walks would improve the results. We plan to achieve this by automatically tuning the bias score for each type of node (or relation), while training on the graph and checking the link prediction results to adjust the scores. The number of parameters to tune is related to the number of types or relations that we have in KG, which seems to be practical to a model to work with.

CHAPTER 5

PREDICTING PATHWAY ASSOCIATIONS FOR UNDERSTUDIED DARK KINASES USING A PATTERN-BASED GRAPH EMBEDDING ON HETEROGENOUS KNOWLEDGE GRAPHS [27]

[27] Abbas Keshavarzi, Liang-Chin Huang, Krzysztof J. Kochut, and Natarajan Kannan, to be submitted to *Computational Biology*.

Abstract

Given a Knowledge Graph (KG) Link Prediction is a task of predicting links between nodes in the graph. Capturing the structure of the graph and the characteristics of neighboring nodes may offer critical information for predicting possible links within the graph. Although link prediction can be applied to networks from different domains, it is particularly interesting in biological networks, where many important associations can be discovered based on graph structure. In this work, we propose a new approach to sampling a large knowledge graph and using an embedding technique adapted from one often used in Natural Language Processing. We were able to use this approach to accurately predict known associations between kinases and biological pathways and predict new putative associations between under studied kinases and pathways.

5.1 Introduction

Given a Knowledge Graph (KG), the Link Prediction task is predicting the next most probable link in the network [52], [60] This task differs from finding missing links that occur frequently due to incompleteness of KGs. To this end, capturing the structure of data, the characteristics of neighboring nodes might result in predicting the possible links within the data. Although the link prediction can be applied to networks from different domains, it is particularly interesting in biological networks where some associations are to be discovered [61], [62]. As the analytics techniques are computationally expensive especially on huge graphs, researchers tend to reduce the dimensionality of a graph into the low dimension space. Graph embedding aims to preserve the structure of the graph while representing it into low dimensional vectors [32]. Based on the work mentioned earlier, there are six different categories to generate vectors from a graph namely, matrix factorization, deep learning, edge reconstruction-based optimization, graph kernels, generative models, and hybrid models. There are two main categories of Deep Learning approaches to learn representation of graphs. The DL with random walks such as metapath2vec [34] and DL without random walks where other techniques used to compute the vector embeddings such as [38] and [39].

In this work, we focus on deep learning approach in which random walks are used to sample the graph. This approach is based on a family of models from Natural Language Processing (NLP) called word2vec [33] and we specifically used the skip-gram model in producing the vector embeddings. Skip-gram tries to find the semantic similarity between words in a context by learning a meaningful representation for each word in a corpus of sentences or documents. The main intuition is that we can imply the meaning of a word by

the understanding of other words in a sentence. In the basic word2vec approach, the algorithm took a sentence and consider a window (usually of size 5 to 10) around the word of interest (center word) and generate training examples for a simple Neural Network (NN) with one hidden layer. The training examples would be the pair of the center word and each of the words within the window size (context words). Then it trains the Neural Network to maximize the probability of context word, given the center word. Then the weights in the trained network are used as embeddings for each word in the dictionary.

5.2 Materials and Methods

5.2.1 Creation of curated kinase knowledge graph for kinase-pathway link prediction

Knowledge Graphs are very similar to Heterogeneous Information Networks. An *Information Network* is a directed graph G = (V, E), composed of vertices and edges, with an associated object type mapping function $\phi : V \rightarrow A$ and an edge type mapping function $\psi : E \rightarrow R$ [31]. Each object $v \in V$ belongs to one particular object type in the object type set A: $\phi(v) \in A$, and each edge $e \in E$ belongs to a particular relation type in the relation type set R: $\psi(e) \in R$. An Information Network is called a *Heterogeneous Information Network* if the sets A and R both contain *more than one element*, that is, there are multiple labels for graph nodes and multiple labels for edges. In case that the sets A and R are singletons, the Information Network is called Homogeneous Information Network (all nodes in the network are of the same type and all edges are of the same type). While Heterogeneous Information Networks require that if two edges belong to the same relation type, the two edges share the same starting object type as well as the ending object type,

Knowledge Graphs do not. That is, the same relation (edge) type can be applied to different starting object types and different ending types. It is the case in the Resource Description Framework (RDF), a notation often used to represent Knowledge Graphs. RDF is based on the notion of triples of the form subject-predicate-object [50] representing edges connecting entities in the graph.

For example, in a knowledge graph representing information about protein kinases, a node (entity) representing a protein kinase *EGFR* can be connected by an edge labeled *participatesIn* to a node representing a pathway *Signaling by EGFR in Cancer*, which represents the knowledge that *EGFR* participates in the pathway *Signaling by EGFR in Cancer*. In such a knowledge graph, *EGFR* may be connected to other nodes (entities) using different labels (predicates), such as, *EGFR – contains—Furin-like domain*, *EGFR – isClassifiedAs—EGFR family* and *EGFR – isLocatedIn—Cell membrane*. Here, edges have multiple labels and destination nodes are of different types, which indicates that it is a Heterogeneous Knowledge Graph.

Given a Heterogeneous Information Network G = (V, E) (as defined above), the network's schema is a directed graph, S = (A, R), based on G's object type mapping $\phi : V \rightarrow A$ and its link type mapping $\psi : E \rightarrow R$. S is a directed graph defined over object types A, with edges as relations from R. Similarly, the 0schema of a Knowledge Graph represented in RDF is represented in RDFS (RDF Schema [51]). For example, given the EGFR examples above, the schema would contain the edges *Protein —participatesIn — Pathway*, *Protein – isClassifiedAs—Kinase Family*, and *Protein — isLocatedIn — Cellular Location*. The schema (sometimes referred to as a meta-knowledge graph, or meta graph) specifies constraints on the use edge (property) labels to certain types of starting and ending

nodes (subjects and objects in RDF). Also, given a Knowledge Graph Schema, we can create a Knowledge Graph (instance) conforming to the schema containing many individuals (entities).

Our kinase-pathway link prediction knowledge graph was populated by individuals from a variety of relevant types. Edges interconnecting them were populated based on the data obtained from various sources. The list of human protein kinases was obtained from [63]. The protein-protein interation network was retrieved from STRING [64] and we only included interaction with score more than 700 based on laboratory experiments. For pathway association we used the Reactome [65] v.76 dataset and split the associations to two sets of manually curated (evidence=TAS) and predicted association (evidence=IEA). We include the manually curated in our KG and saved the predicted pathway associations for future evaluation purposes. We filtered out high-level pathways and removed their association until no protein that has some number of pathway association left without any association. We also removed the predicted pathways with low confidence under hierarchy of disease pathway except infectious diseases. The Gene Ontology annotations [66] (v2019_11) was used to enriched the proteins with molecular functions, cellular components and biological processes. We excluded the associations that were not manually curated (IEA evidence) and removed the high level ones when they have more than 5000 connections. All the functional domains for proteins from the Pfam [67] (version 33.1) were used except protein kinase domains (“Pkinase” and “Pkinase_Tyr”). The[68] PTMs and the assoications extracted from iPTMnet [18] (version 5) with confidence score more than 1.0 were included in the graph. The protein-chemical and protein-disease associations

were extracted from the Comparative Toxicogenomics Database [69]. Table 5.1 shows basic statistic of different edges and their source and target nodes.

Table 5. 1. Number of associations between different type of nodes.

Association (source node-target node)	Edge	Source node	Target node
Protein-Pathway	68,856	7298	1974
Protein-Functional Domain	1,711,017	990,504	7834
Protein-PTM	872	308	677
Protein-Molecular Function	743,539	166,338	4,528
Protein-Biological Process	2,621,261	188,237	17,670
Protein-Cellular Component	466,057	124,988	2,152
Protein-Protein interaction	64,517	11,091	44,819
Protein-Chemical	436,831	9,426	14,397
Protein-Disease	27,282	6,806	5,234
Chemical-Disease	74,757	5,251	1,731
Chemical-Molecular Function	3,274	1,154	229
Chemical-Biological Process	60,974	5,357	3,976
Chemical-Cellular Component	136	92	54

Therefore, the populated KG has 11 types of nodes (meta-nodes) and 13 types of edges (meta-edges), where 6,279,373 edges are connecting the 1,064,097unique nodes of different types. The total unique number of each type of nodes along with their average degrees are shown in Table 5.2.

As discussed above, the KG may contain hierarchical data. For example, *Pathways* are often organized into a parent-child relationship among pathways. Similarly, data for *Molecular Function*, *Biological Process* and *Cellular component* contain hierarchical data, as, for example, there are many subtypes of *Molecular Function*. The lower-level nodes are necessary and useful to our approach, but such higher-level nodes have some

disadvantages. First, having hierarchical nodes in the graph introduces redundancy in the data, as edges connecting lower-level nodes should have mirror edges for the corresponding parent nodes. Second, nodes in the upper level of the hierarchy might connect to thousands of nodes (often referred to as *hubs*) which makes the vector embeddings (used in our method) more general than they should be. In fact, we want vector embeddings for nodes to be encoded for more specific semantics. Consequently, we removed the high-level nodes of types *Molecular Function*, *Biological Process* and *Cellular component*, from the graph which led to more accurate predictions.

Table 5. 2. The unique number of node types with their average degree in the KG.

Node Type	Unique # of nodes	Average degree
Well-studied Kinases	390	85
Under-studied Kinases	151	29
Protein	1014061	6
Pathway	1974	34
PTM	677	1
Functional Domain	7834	218
Biological Process	17670	151
Molecular Function	4528	164
Cellular Component	2152	216
Chemical	9426	61
Disease	5234	19

The knowledge graph used in our link predition experiments described here contains several types of entities (meta nodes) such as *Protein*, GO Terms (*Biological Process*, *Molecular Function*, and *Cellular component), Disease*, and other types. The schema organization is shown in Figure 5.1. The dotted line represents the schema edge

Protein —participatesIn — Pathway, which represents the known (existing) edges, retrieved from Reactome [65] and included in our knowedge graph, and unknown (previously not reported) links that we are predicting. In addition, the *Protein* type includes the understudies kinases (referred to as Dark Kinases), well-studied kinases (referred to as Light Kinases), and other human proteins. This distinction is important, as it gives us the ability to make predictions for a subset of proteins. The meta-edges (edges in the schema) are not labeled here, as we do not consider them in our method described here. Instead, we only rely on the types of source and target nodes. The loop edge going back to the *Protein* type indicates the *Protein-Protein* interaction relation, which is included in our knowledge graph.

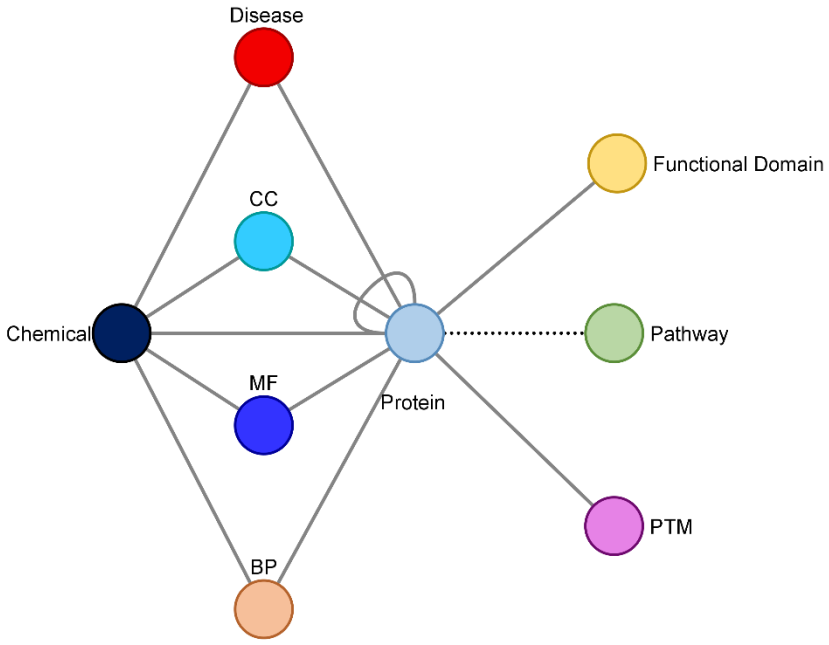

Figure 5. 1. Schema of the Knowledge Graph. The dotted line is the link of interest.

5.2.2 Predicting Links

In the experiments and results presented in this paper, we used our novel graph embedding approach, called RegPattern2Vec. It is used as the first step in our link prediction process to produce vector representations for the nodes in the graph and formulate the link prediction as a classification problem. A Machine Learning model was trained using the combined vectors of existing pairs of nodes connected by an edge (link) of interest. The outline of the method is illustrated in figure 5.2. We discuss each step of our link prediction method in the subsequent sections.

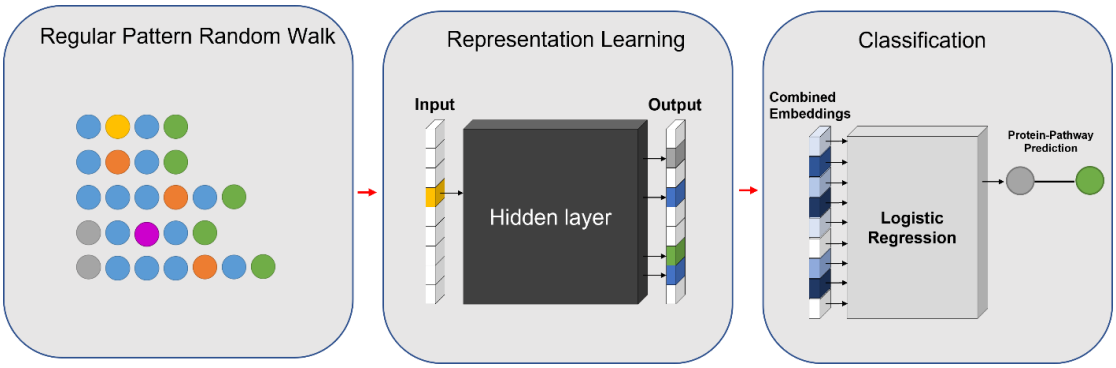

Figure 5. 2. RegPattern2Vec overview.

5.2.3 Regular pattern selection and its usage in random walks

To encode the nodes in the KG to vectors suitable to be used in Machine Learning, we need a way to "flatten" the graph into sequences of nodes. Performing random walks on a graph is one way to generate such sequences and previous approaches have constrained random walks using metapaths [34] which contrain the walk by a fixed number of user defined hops (the length of a meta-path). However, creating such metapaths for complex

Heterogeneous Knowledge Graphs, often with no well-defined schema, is challenging and time consuming. A number of previous attempts to automatically generate meta-paths have been reported in the literature [30], [58], [70] but they all rely on fixed-length meta-paths which seems to be a major limitation for vector representations, when we just look into their immediate neighbors. A regular pattern approach introduced in this work, allows numerous sequences of meta-nodes based on the pattern and is a generalization of the meta-paths and meta-graphs (figure 5.3b).

The regular pattern is defined as schema subgraph that should include only the meta-nodes that are suitable for a specific problem. An example regular pattern subgraph is shown in figure 5.3.b. Unlike in the Node2Vec method [71], the schema graph pattern represents several paths (walks) that can be used to predict missing links, in our case, "Proteins" participating in "Pathways". Another way to think of several possible walks is to represent them as a regular expression that connects Protein and Pathway. For example, the meta-graph pattern includes a path (walk) represented as a regular expression $\boldsymbol{H}[\hat{}\boldsymbol{T}]+ \boldsymbol{H}\ \boldsymbol{T}$, where H is a set of source node type of edge of interest and T is a set of target node type of edge of interest and it guides random walks, as demonstrated in Figure 3c. As mentioned before, as the link of our interest is Protein-Pathway, we have defined the regular pattern as $Protein\,[\hat{}Pathway] + Protein\,Pathway$ and each walk instance should satisfy it. Each walk starts by randomly selecting a Protein node. Selecting the next nodes on a walk is based on the existing graph nodes and matching the neighbor's type in the regular pattern. When a walk reaches a pathway node, it follows the pattern in reverse

order and does so until a certain number of steps (nodes) in the walk are reached or when it reaches a termination node when there is no neighbor to follow the graph pattern.

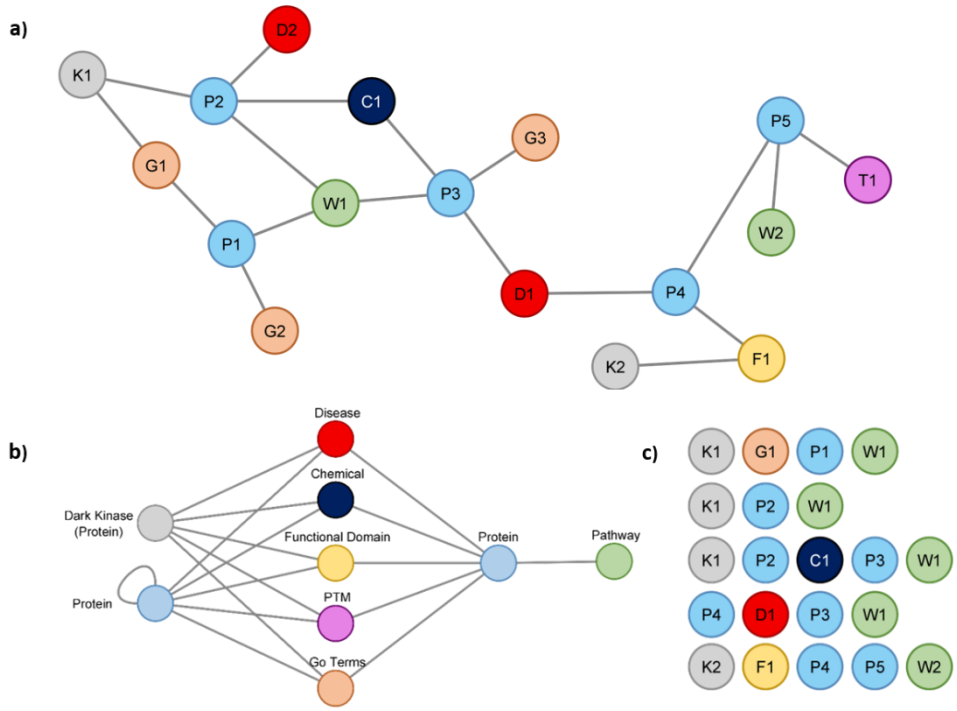

Figure 5. 3. (a) a hypothetical subgraph of KG. (b) The regular pattern (c) the flattened node of the subgraph.

5.2.4 Capturing semantic relationships between Proteins and Pathways in Deep Learning

RegPattern2Vec uses a modified skip-gram model presented in [34] to generate vector representations for the nodes of the Knowledge Graph. The random walks generate sequences of nodes, which resemble natural language sentences. The machine learning model simultaneously captures the local structure of the graph and types of the nodes and encodes them as vector representations. Figure 5.4. shows learned vector representation of

node in the vector space using dimension reduction technique called Principal Component Analysis (PCA). As our goal is to predict the protein-pathway associations, we just consider the nodes to be of three types: "*Protein"*, "*Pathway"* and "others", when learning representation for the nodes. The separation of nodes in PCA shows the encoded types of nodes and other dimensions of vector capture their local structure in graph.

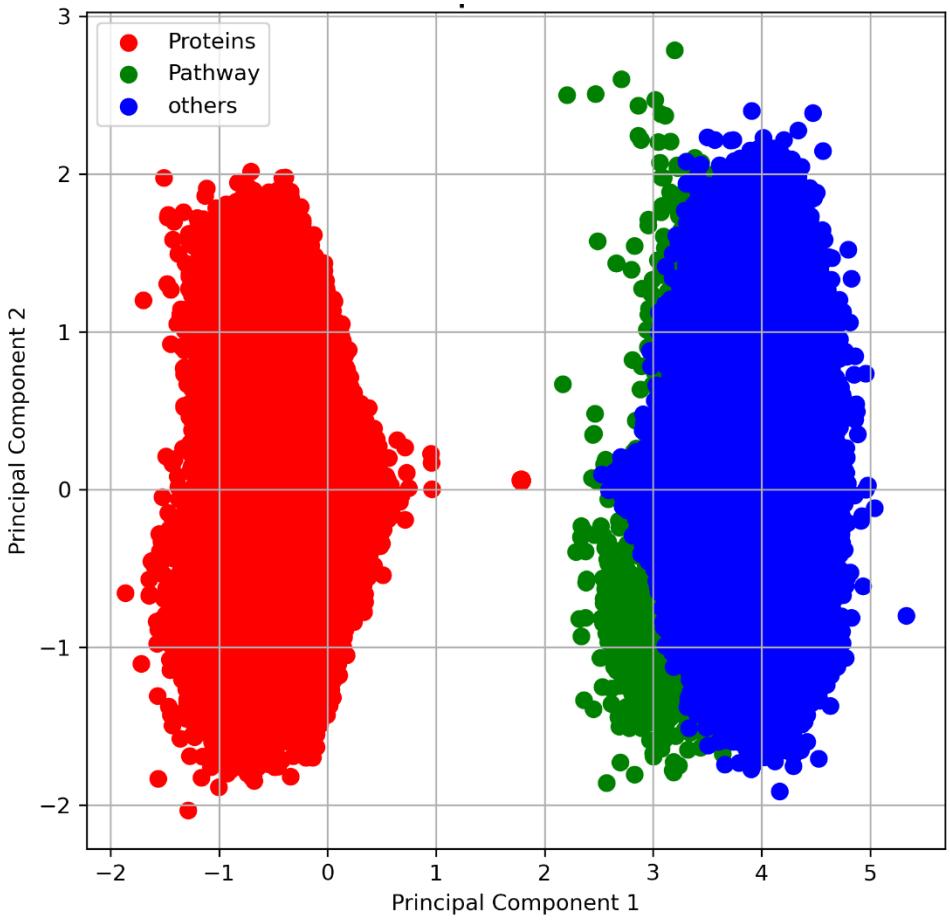

Figure 5. 4. PCA of vector embeddings of all nodes generated by machine learning model.

5.2.5 Regular-pattern definition

Given $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ denotes a node set with type mapping function $\phi : \mathcal{V} \rightarrow \mathcal{T}$, where $\mathcal{T}$ denotes a node type set and $\mathcal{E}$ is an edge set with type mapping function $\varphi : \mathcal{E} \rightarrow \mathcal{R}$ where $\mathcal{R}$ denotes a relation type set. if (h, t, r) is edge of interest, where h, t $\in \mathcal{V}$ and r $\in \mathcal{E}$ and $\varphi(h) = H, \varphi(\mathrm{t}) = \mathrm{T}$, The regular pattern is defined as $\boldsymbol{H[\hat{}T]+ H\ T}$.

5.2.6 Biased random walk constrained by regular pattern

As this work considered undirect graphs, enumerating all the paths of a given graph is impossible. The solution is to sample some paths from all possible path according to some distribution. The random walk with regular pattern is selected to generate arbitrary number of paths. This number can be controlled by "walk length" parameter. Having the undirected heterogeneous network, and selected regular pattern, the random walk can be started from each instance of a starting node type in the pattern. As we want all the nodes to appear in our walk, iterating over all of them would be desirable. It is obvious that if we repeat the walk from each node, we will discover more path as the node might link to multiple nodes of the same type. We will call this parameter "number of walks". We will discuss how to choose the parameter and analysis of their impacts in the next section. The next step for each node is to select a node from the adjacent nodes based on the pattern that we are considering. This might result in multiple choices and this is where the randomization comes to play. RegPattern2vec can get distribution by user-defined function to generate same probability for all the node or arbitrary distribution. A regular pattern is converted to a Deterministic Finite Automata (DFA) and each of the possible transitions are mapped to

the DFA, denoted by M. The DFA M is responsible to check if transitions are allowed (an edge between two nodes) thus, disallowed transition gets zero probability and were not used in the random walks.

On the other hand, in scale-free networks where the degree distribution follows the power law, there are some "hub" nodes that have high degree of income/outgoing edges. Because such high degree nodes can dominate random walks and, consequently, representation learning, one popular way is bias the walk by inverse of degrees of nodes [71] where probability of choosing the node v_{i+1} from v_i is calculated by normalizing the inverse of degrees of all neighbors of v_i. Although this approach prevents the random walk from selecting high degree nodes , it biases the random walk toward low-degree nodes . To accomplish better distribution and avoid both biases, we proposed the formula below:

$$P(v^{i+1} \mid v^i, M) = \begin{cases} g(r^i)\dfrac{\frac{1}{|N_{v^{i+1}}|}}{\sum_{v \in N_{v^i}} \frac{1}{|N_v|}} & (v^i, r^i, v^{i+1}) \text{ is an edge in the graph and the transition from } v^i \text{ to } v^{i+1} \text{ is allowed in } M. \\ 0 & (v^i, r^i, v^{i+1}) \text{ is an edge, but the transition is undefined} \\ 0 & (v^i, r^i, v^{i+1}) \text{ is not exists in the graph} \end{cases}$$

Where $g(r^i)$ is the proportion of r^iamong all outgoing edge types, $|N_v|$ is the of degree of node v, v^i denotes the current node and the candidate node for next step is $v^{i+1} \in N_{v^i}$, and N_{v^i} is the set of all the neighbors of node v^i. Therefor we randomly choose one relation type (independently) and then use the probability distribution by inverse of node degrees to select the next node in the walk. After applying the proposed formula above, the nodes with high and also low degrees are visited moderately and suggest the better exploration of nodes as demonstrated in figure 5.5.

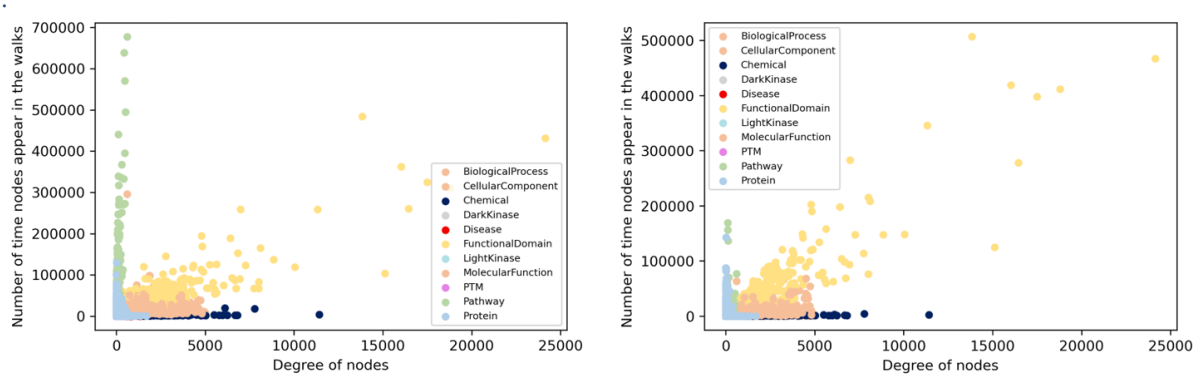

Figure 5. 5. Left figure shows the number of times a node appears in the random walk only using the inverse of degrees. The Right figure shows the effect of using our proposed formula for finding proper probability distribution for selecting the next node.

5.3 Link Prediction as a classification problem

For each pair of protein-pathways, we combined their vector embedding using a widely used Hadamard product [72]–[74] and resulting vector is used as features to train a Logistic Regression model as positive examples. Generating negative examples is the next issue. The open-world assumption for knowledge representation defines that the true values are the statement whether that is known or not. In contrast, the close-world assumption, determines the statement are known to be true which is the basis for generating the negative examples. Having an edge of interest, a random node from head nodes of the edge along with another node from tail nodes of the edge are selected. If they don't form a link in the data, we combined their vector embeddings and count them as negative example. And we do this process until the same number of negative examples are generated.

5.4 Results and Discussion

5.4.1 Comparison of RegPatter2vec and other graph embedding methods in Protein-Pathway prediction

To evaluate our method, we generated a test set by excluding 50% of known Protein-Pathway associations from the training set. The training examples are generated using the process mentioned in section “Link Prediction as a classification problem” and a logistic regression algorithm is selected as the binary classification model for the link prediction task. A 10-fold cross-validation applied to make sure that model does not have overfit, then the train model acquired f1-score of 0.859 and AUC ROC of 0.9383 on the test set. We selected two methods as baseline to compare their performance to RegPattern2Vec method. The node2vec [73] which has been shown to perform well on many datasets and metapath2vec [34] which inspired this work are selected for comparison. To select a specific meta-path that is required by metapath2vec approach, we tested several meta-paths and selected the “Protein-Functional Domain-Protein-Pathway”, which showed the best performance. As shown in figure 6 RegPattern2vec achieves better or comparable accuracy to Node2Vec. Both Node2Vec and RegPatter2Vec perform significantly better than metapath2vec as determined by AUC-ROC shown in figure 6.

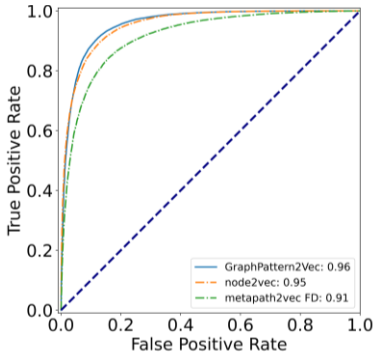

Figure 5. 6. Comparing ROC of RegPattern2Vec with baselines, node2vec and metapath2vec with best metapath (Protein-FunctionalDomain-Protein-Pathway)

5.4.2 Predicting possible association of Dark Kinases and pathways

We next used the two high performing methods (RegPattern2Vec and Node2Vec) to predict pathway associations for under-studied (dark) Kinase nodes in the graph.. The optimal threshold is extracted from the model when it maximizes both the precision and recall, and it has been used to make predictions. Among all 319,214 possible pairs, RegPattern2Vec predicted 44,142 new possible associations and node2vec predicted 33,409 associations. The top 50 overlap predictions of possible protein-pathway associations by both methods are shown in the figure 5.7.

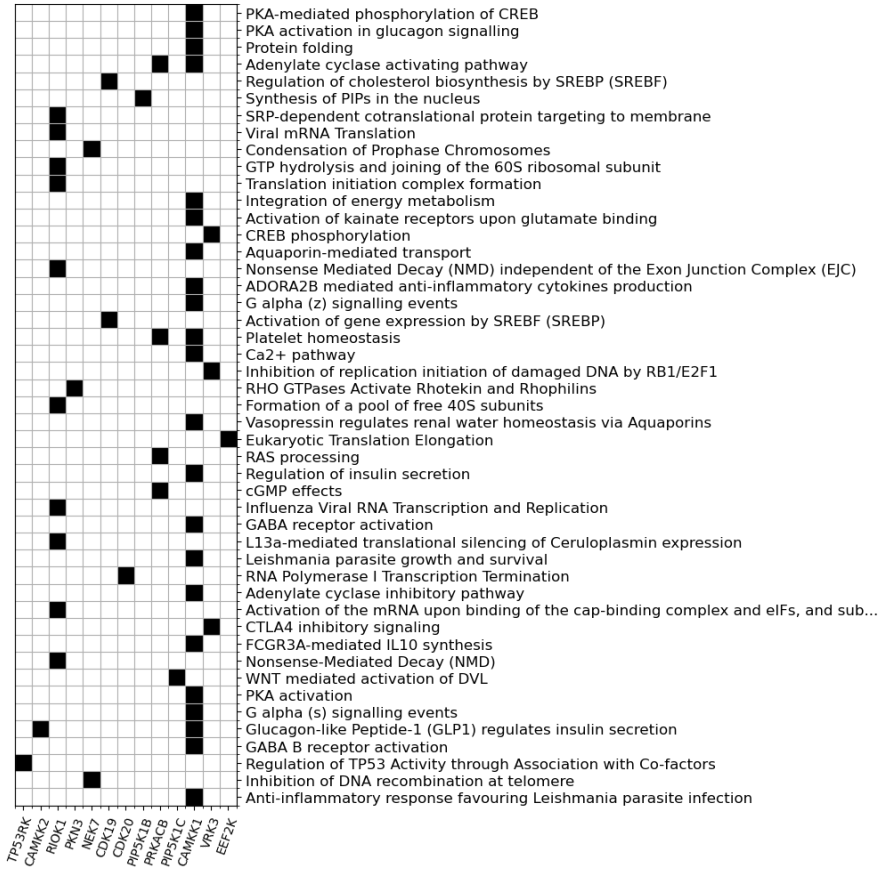

Figure 5. 7. Top 50 overlap predictions of RegPattern2Vec and node2vec methods on dark kinases.

5.4.3 Meta-paths contributing to the predictions.

Comparing RegPattern2vec with meta-path approaches show that although regular pattern does not explicitly use meta-path, it can take advantage of such paths implicitly. RegPattern2Vec discovers meta-paths along the way and benefits from their power to find similarity between nodes in the graph. Defining each of the meta-paths discovered by regular patterns is not trivial for a domain expert and the representation learning model determines their importance based on the node types and their proximity to each other within the generated path instances. Figure 8.5 shows the top 5 predictions of protein and pathway pairs, and the analysis the path instances in which they appeared together. The x-axis contains the frequent patterns inferred from those path instances and the y-axis is five prediction pairs. And the number of time that each pattern is discovered during guided walk using RegPattern are colored.

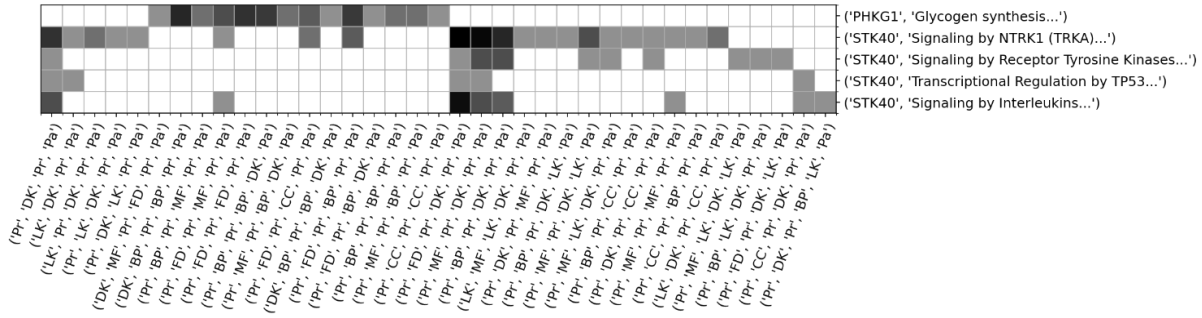

Figure 5. 8. Explored patterns contributing to the top 5 predictions by Regular Pattern's Constrained walks. Note Pr stands for protein, DK for dark kinase, LK for light kinase, MF for molecular function, FD for functional domain, BP for biological process, CC for cellular components.

5.4.4 Overlap predictions and IDG Protein-Protein Interaction

The enrichment analysis results based on IDG protein-protein interaction contains 533 associations from 16 unique dark kinases and 216 pathways. From those numbers, only 12 dark kinases and 183 pathways exist in our graph. If we extract enrichment result only for protein and pathways that exist in our graph, there will be only 472 associations. We compare them with overlap predicted associations by both methods and we find out 27 predictions in common, although RegPattern2Vec was able to predict 76 of them. The 27 common predictions are listed in table 5.3.

Table 5. 3. Overlap association commonly predicted by RegPattern2Vec and node2vec and the result of enrichment analysis on IDG PPI.

Protein	Pathway	RegPattern2Vec	node2vec	FDR
CDK12	RNA Polymerase II Pre-transcription Events	0.999999464	0.995687	0.0344
CDK12	Formation of RNA Pol II elongation complex	0.999999941	0.992527	0.0241
CDK12	RNA Polymerase II Transcription Elongation	0.999999557	0.991911	0.0321
CDK12	RNA polymerase II transcribes snRNA genes	0.99999982	0.991324	0.0279
CSNK1G2	G2/M Checkpoints	0.997483821	0.942078	0.00455
TLK2	Formation of Senescence-Associated Heterochromatin Foci (SAHF)	0.999998816	0.934615	0.00102
CSNK1G2	Activated PKN1 stimulates transcription of AR (androgen receptor) regulated genes KLK2 and KLK3	0.982463888	0.938096	0.00103
CSNK1G2	Signaling by WNT	0.999999723	0.910032	0.00165
CDK13	RNA Polymerase II Transcription Elongation	0.999575457	0.909773	0.0552
CSNK1G2	M Phase	0.998510706	0.886028	0.0425
CDK13	Formation of RNA Pol II elongation complex	0.999986057	0.840819	0.0368
CSNK1G2	Mitotic Prophase	0.993864995	0.822416	0.00184
CSNK1G2	Nonhomologous End-Joining (NHEJ)	0.990734367	0.822585	0.000915
CSNK1G2	TCF dependent signaling in response to WNT	0.99996334	0.78728	0.000838
CSNK1G2	Signaling by Nuclear Receptors	0.999999822	0.77718	0.00115
CAMKK1	Cilium Assembly	0.997762204	0.776394	0.00419
CSNK1G2	Cell Cycle Checkpoints	0.999919748	0.720566	0.0217
CSNK1G2	RHO GTPases activate PKNs	0.999970941	0.713065	0.000795
CSNK1G2	Processing of DNA double-strand break ends	0.999509566	0.702158	0.00112
CAMKK1	Organelle biogenesis and maintenance	0.98835753	0.67373	0.00922
CSNK1G2	E3 ubiquitin ligases ubiquitinate target proteins	0.99902807	0.653647	0.0189
CSNK1G2	Developmental Biology	0.992324873	0.643503	0.00163
CSNK1G2	Amyloid fiber formation	0.999952875	0.630661	0.00114
CSNK1G2	Cellular Senescence	0.929697325	0.64151	0.00571
CSNK1G1	Signaling by Nuclear Receptors	0.861378864	0.693192	0.0275
CSNK1G2	Depurination	0.922711131	0.588167	0.00128
CSNK1G2	Oxidative Stress Induced Senescence	0.603904458	0.804577	0.00143

5.5 Conclusion and future work:

In this work, we propose a new guided random walk approach for link prediction in KGs. Unlike metapath2vec [34], RegPattern2Vec [75] does not need prior knowledge of graph structure (i.e., predefined metapaths) for capturing contextual information in KGs. RegPattern2Vec is more accurate than metapath2vec and more efficient than approaches like node2vec.

We aim to extend this framework as a tool for graph databases, such as neo4j, where users and a domain expert can effortlessly perform link prediction on the live data and evaluate their predictions with the existing knowledge. Another goal is to investigate existing explainable AI approaches to justify the predicted links and discover better method when they can be applied to any domain. Using the paths (the sequences of nodes) that are generated in random walks is possibly the best way to tackle this problem. The instances of nodes usually have descriptive names consist of terms and domain specific phrases. An overrepresentation analysis of these terms for each of predictions may consist of semantics that needed for justification. Also, as the RegPattern2vec method uses regular expression patterns, the patterns with high frequency for each prediction can also be used to explain the vector representation of each node and therefore identifying the semantic behind the proposed links in the knowledge graph.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this dissertation, we presented a framework, called KGdiff, to discover the schema and schema-in-use of knowledge graphs and track its evolution over time where it collects meta-data information of KG by running many fixed (domain-independent) SPARQL queries against a SPARQL endpoint and stores the information for further analysis and comparison with future and prior versions. The report consists of statistics and descriptions of various constructs in a KG. It helps KG engineers avoid manual comparison of different versions of KG, which is time-consuming and expensive. Our results show that KGdiff is efficient even on sizable KGs and ontologies, and it can accurately identify changes both in the schema and the number of individuals on different versions. To accomplish this, we started by learning the process of ontology population, differentiate the validation and verification of ontologies and how to query in triple stores and what are the main construct of vocabularies used to build a KG. Then we learnt the different methods and API to connect the KG/Ontology, analyzing the data by discovering schema and schema in-use from the KGs.

Our second work presents a novel graph sampling approach called RegPattern2Vec, where a regular pattern guides a random walk to sample a large KG more efficiently and learn node embeddings for different tasks, especially link prediction, as demonstrated in our work. RegPattern2Vec has shown to be superior to related graph embedding approaches both in accuracy and efficiency. We have examined different

methods in different disciplines and tested the state-of-the-art methods on a real-world application. We thoroughly tested our method and verified our results through a comprehensive experiments and evaluations. And finally, we thoroughly studied the impact of our results in an important domain of biology.

As for future direction, we want to extend the KGdiff to be able to work with different data stores and graph databases and their query languages, such as Cypher in Neo4j. Although Cypher was inspired by SPARQL, a specific mapping from our generic SPARQL queries should be converted to Cypher. Another improvement would be to identify and report other OWL constructs and include them in a complete KGdiff report. Also, it would be beneficial to include a visualization component to provide more insight about the KG for the users.

For RegPattern2Vec, we will explore the possibility of learning the biases of random walks to favor indicated types of nodes (or relation types) that might better contribute to the link prediction accuracy. It has been shown that some patterns and types of nodes and/or relations are more relevant for discovery of possible unknown linkages in a graph. Having such a flexibility, we will be able to take advantage of the existing knowledge in the domain of interest to increase the accuracy and confidence of the predictions.

We have implemented the RegPattern2Vec as Python package, which can be easily installed on a computer system and used for link prediction experiments. The data in this case stored in some text files, such as CSV, and an extra step needs to be done to convert the KG to edge and node files with specific formats. Recently, graph databases, such Neo4j, have been rapidly increasing in popularity and we want to implement our method as a

plugin for Neo4j, where the RegPattern2Vec can be applied directly to the data in the graph database, avoiding the extra data preparation step.

Finally, we have implemented some level of parallelism in RegPattern2Vec taking advantage of multiprocessing servers with multiple cores, to speed up the process of model learning. As multi-clustering is available in Neo4j 3.4 [28] or higher, we plan to adapt our method to work on partitioned KGs. We plan to study the effects of different types of partitioning on time efficiency and accuracy of random walks on partitioned knowledge graphs.

[28] https://neo4j.com/blog/neo4j-graph-database-3-4-ga-release/

REFERENCES

[1] T. BERNERS-LEE, J. HENDLER, and O. LASSILA, “The Semantic Web,” *Sci. Am.*, vol. 284, no. 5, pp. 34–43, 2001.

[2] J. Barrasa, “Neosemantics.” .

[3] G. A. Miller, “WordNet: a lexical database for English,” *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995, doi: 10.1145/219717.219748.

[4] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, *Freebase: A Collaboratively Created Graph Database For Structuring Human Knowledge*. .

[5] S. Mudgal *et al.*, “Deep Learning for Entity Matching: A Design Space Exploration,” p. 16, 2018, doi: 10.1145/3183713.3196926.

[6] N. Taleb, B. Tighiouart, and S. Laiche, “A method based on OWL schema for detecting changes between Ontology’s versions,” *Intell. Decis. Technol.*, vol. 8, pp. 45–52, 2014, doi: 10.3233/IDT-130176.

[7] N. Noy, M. Klein, and M. A. Musen, “Tracking Changes During Ontology Evolution,” 2004, doi: 10.1007/978-3-540-30475-3_19.

[8] J. Eder and K. Wiggisser, “Detecting Changes in Ontologies via DAG Comparison,” 2006. Accessed: Sep. 18, 2019. [Online]. Available: http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-200/11.pdf.

[9] D. Hutchison and J. C. Mitchell, *A Versioning and Evolution Framework for RDF Knowledge Bases*, vol. 9, no. 3. 2006.

[10] D. Zeginis, Y. Tzitzikas, and V. Christophides, “On the Foundations of Computing Deltas Between RDF Models,” 2007. Accessed: Sep. 18, 2019. [Online]. Available: http://www.w3.org/2004/03/trix/.

[11] R. S. Gonçalves, B. Parsia, and U. Sattler, *Categorising Logical Differences Between OWL Ontologies*. 2011.

[12] M. Hartung, A. Groß, and E. Rahm, "COnto-Diff: Generation of complex evolution mappings for life science ontologies," *J. Biomed. Inform.*, vol. 46, no. 1, pp. 15–32, 2013, doi: 10.1016/j.jbi.2012.04.009.

[13] Y. Roussakis, I. Chrysakis, K. Stefanidis, G. Flouris, and Y. Stavrakas, "A Flexible Framework for Understanding the Dynamics of Evolving RDF Datasets: Extended Version." Accessed: Sep. 20, 2019. [Online]. Available: http://www.w3.org/RDF/.

[14] J. Malone *et al.*, "Modeling sample variables with an Experimental Factor Ontology," *Bioinformatics*, vol. 26, no. 8, pp. 1112–1118, 2010, doi: 10.1093/bioinformatics/btq099.

[15] F. Emmert-Streib, M. Dehmer, and Y. Shi, "Fifty years of graph matching, network alignment and network comparison," *Inf. Sci. (Ny).*, vol. 346–347, pp. 180–197, 2016, doi: 10.1016/j.ins.2016.01.074.

[16] G. Nayak, S. Dutta, D. Ajwani, P. Nicholson, and A. Sala, "Automated assessment of knowledge hierarchy evolution: comparing directed acyclic graphs," *Inf. Retr. J.*, vol. 22, no. 3–4, pp. 256–284, 2019, doi: 10.1007/s10791-018-9345-y.

[17] M. Pietranik and N. T. Nguyen, "Framework for ontology evolution based on a multi-attribute alignment method," *Proc. - 2015 IEEE 2nd Int. Conf. Cybern. CYBCONF 2015*, pp. 108–112, 2015, doi: 10.1109/CYBConf.2015.7175915.

[18] A. Castelltort and A. Laurent, "Representing history in graph-oriented NoSQL databases: A versioning system," *8th Int. Conf. Digit. Inf. Manag. ICDIM 2013*,

vol. 1, pp. 228–234, 2013, doi: 10.1109/ICDIM.2013.6694022.

[19] G. Gosal, K. J. Kochut, and N. Kannan, “Prokino: An ontology for integrative analysis of protein kinases in cancer,” *PLoS One*, vol. 6, no. 12, pp. 1–9, 2011, doi: 10.1371/journal.pone.0028782.

[20] D. I. Mcskimming *et al.*, “ProKinO: A unified resource for mining the cancer kinome,” *Hum. Mutat.*, vol. 36, no. 2, pp. 175–186, 2015, doi: 10.1002/humu.22726.

[21] S. Bamford *et al.*, “The COSMIC (Catalogue of Somatic Mutations in Cancer) database and website,” *Br. J. Cancer*, vol. 91, no. 2, pp. 355–358, 2004, doi: 10.1038/sj.bjc.6601894.

[22] A. Bateman, “UniProt: A worldwide hub of protein knowledge,” *Nucleic Acids Res.*, vol. 47, no. D1, pp. D506–D515, 2019, doi: 10.1093/nar/gky1049.

[23] D. Croft *et al.*, “The Reactome pathway knowledgebase,” *Nucleic Acids Res.*, vol. 42, no. D1, pp. 472–477, 2014, doi: 10.1093/nar/gkt1102.

[24] A. Kwon *et al.*, “Tracing the origin and evolution of pseudokinases across the tree of life,” *Sci. Signal.*, vol. 12, no. 578, 2019, doi: 10.1126/scisignal.aav3810.

[25] A. Bandrowski *et al.*, “The Ontology for Biomedical Investigations,” *PLoS One*, vol. 11, no. 4, pp. 1–19, 2016, doi: 10.1371/journal.pone.0154556.

[26] I. Atastina, B. Sitohang, G. A. P. Saptawati, and V. S. Moertini, “A Review of Big Graph Mining Research,” 2018, Accessed: Feb. 20, 2018. [Online]. Available: http://iopscience.iop.org/article/10.1088/1757-899X/180/1/012065/pdf.

[27] A. R. Elias, N. Golubovic, C. Krintz, and R. Wolski, “Where’s The Bear?,” *2017 IEEE/ACM Second Int. Conf. Internet-of-Things Des. Implement.*, pp. 247–258,

2017, doi: 10.1145/3054977.3054986.

[28] M. Toutiaee, A. Keshavarzi, A. Farahani, and J. A. Miller, “Video Contents Understanding using Deep Neural Networks.”

[29] I. Spasic, S. Ananiadou, J. McNaught, and A. Kumar, “Text mining and ontologies in biomedicine: Making sense of raw text,” *Brief. Bioinform.*, vol. 6, no. 3, pp. 239–251, 2005, doi: 10.1093/bib/6.3.239.

[30] G. Wan, B. Du, S. Pan, and G. Haffari, “Reinforcement Learning based Meta-path Discovery in Large-scale Heterogeneous Information Networks,” *Aaai*, 2020, Accessed: Jun. 12, 2020. [Online]. Available: https://github.com/mxz12119/MPDRL.

[31] C. Shi, Y. Li, J. Zhang, Y. Sun, and P. S. Yu, “A Survey of Heterogeneous Information Network Analysis,” *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 1, pp. 17–37, 2017, doi: 10.1109/TKDE.2016.2598561.

[32] H. Cai, V. W. Zheng, and K. C. C. Chang, “A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications,” *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 9, pp. 1616–1637, 2018, doi: 10.1109/TKDE.2018.2807452.

[33] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality.” Accessed: Oct. 29, 2018. [Online]. Available: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[34] Y. Dong, N. V Chawla, and A. Swami, “metapath2vec: Scalable Representation Learning for Heterogeneous Networks,” 2017, doi: 10.1145/3097983.3098036.

[35] M. Belkin and P. Niyogi, “Laplacian Eigenmaps and Spectral Techniques for

Embedding and Clustering," 2002.

[36] P. Ristoski and H. Paulheim, "RDF2Vec: RDF graph embeddings for data mining," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016, vol. 9981 LNCS, pp. 498–514, doi: 10.1007/978-3-319-46523-4_30.

[37] H. Wang *et al.*, "GraphGAN: Graph Representation Learning with Generative Adversarial Nets," pp. 2508–2515, 2017, [Online]. Available: http://arxiv.org/abs/1711.08267.

[38] L. F. R. Ribeiro, P. H. P. Saverese, and D. R. Figueiredo, "struc2vec: Learning Node Representations from Structural Identity," 2017, doi: 10.1145/3097983.3098061.

[39] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning Entity and Relation Embeddings for Knowledge Graph Completion." Accessed: Oct. 02, 2019. [Online]. Available: www.aaai.org.

[40] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling Relational Data with Graph Convolutional Networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 10843 LNCS, pp. 593–607, doi: 10.1007/978-3-319-93417-4_38.

[41] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, vol. 13-17-Augu, pp. 1225–1234, 2016, doi: 10.1145/2939672.2939753.

[42] M. Ley, "Dblp computer science bibliography," 2005. .

[43] C. Meng, R. Cheng, S. Maniu, P. Senellart, and W. Zhang, “Discovering meta-paths in large heterogeneous information networks,” in *WWW 2015 - Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 754–764, doi: 10.1145/2736277.2741123.

[44] A. Keshavarzi and K. J. Kochut, “KGdiff: Tracking the Evolution of Knowledge Graphs,” *Proc. - 2020 IEEE 21st Int. Conf. Inf. Reuse Integr. Data Sci. IRI 2020*, pp. 279–286, 2020, doi: 10.1109/IRI49571.2020.00047.

[45] Z. Huang, Y. Zheng, R. Cheng, Y. Sun, N. Mamoulis, and X. Li, “Meta structure: Computing relevance in large heterogeneous information networks,” in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, vol. 13-17-Augu, pp. 1595–1604, doi: 10.1145/2939672.2939815.

[46] D. Zhang, J. Yin, X. Zhu, and C. Zhang, “MetaGraph2Vec: Complex Semantic Path Augmented Heterogeneous Network Embedding,” 2018.

[47] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, “Convolutional 2D Knowledge Graph Embeddings,” 2017, Accessed: Nov. 23, 2018. [Online]. Available: www.aaai.org.

[48] W. L. Hamilton, P. Bajaj, M. Zitnik, D. Jurafsky, and J. Leskovec, “Embedding Logical Queries on Knowledge Graphs.” Accessed: Aug. 20, 2019. [Online]. Available: https://papers.nips.cc/paper/7473-embedding-logical-queries-on-knowledge-graphs.pdf.

[49] M. Spiser, *Introduction to the Theory of Computation*. Cengage learning., 2012.

[50] R. Cyganiak, D. Wood, and M. Lanthaler, “RDF 1.1 Concepts and Abstract

Syntax," *W3C Recommendation*, 2014. https://www.w3.org/TR/rdf11-concepts/ (accessed Dec. 31, 2020).

[51] W3C, "RDF Schema 1.1," 2014. Accessed: Dec. 31, 2020. [Online]. Available: https://www.w3.org/TR/rdf-schema/.

[52] X. Cao, Y. Zheng, C. Shi, J. Li, and · Bin Wu, "Meta-path-based link prediction in schema-rich heterogeneous information network," *Int. J. Data Sci. Anal.*, vol. 3, pp. 285–296, 2017, doi: 10.1007/s41060-017-0046-1.

[53] J. E. Friedl, *Mastering Regular Expressions*. O'Reilly Media, Inc., 2006.

[54] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago," in *Proceedings of the 16th international conference on World Wide Web - WWW '07*, 2007, p. 697, doi: 10.1145/1242572.1242667.

[55] J. Lehmann *et al.*, "DBpedia-A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia LinkingLOD: interlinking knowledge bases View project DL-Learner View project DBpedia-A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia," *Semant. Web*, vol. 1, pp. 1–5, 2012, doi: 10.3233/SW-140134.

[56] T. Mitchell *et al.*, "Never-ending learning," *Commun. ACM*, vol. 61, no. 5, pp. 103–115, 2018, doi: 10.1145/3191513.

[57] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," *Proc. VLDB Endow.*, vol. 4, no. 11, pp. 992–1003, 2011, doi: 10.14778/3402707.3402736.

[58] C. Yang, M. Liu, F. He, X. Zhang, J. Peng, and J. Han, "Similarity modeling on heterogeneous networks via auto-matic path discovery," in *Lecture Notes in*

Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2018, vol. 11052 LNAI, pp. 37–54, doi: 10.1007/978-3-030-10928-8_3.

[59] H. Zhao, Q. Yao, J. Li, Y. Song, and D. L. Lee, "Meta-graph based recommendation fusion over heterogeneous information networks," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, vol. Part F1296, pp. 635–644, doi: 10.1145/3097983.3098063.

[60] Y. Sebastian, E. G. Siew, and S. O. Orimaye, "Learning the heterogeneous bibliographic information network for literature-based discovery," *Knowledge-Based Syst.*, vol. 115, pp. 66–79, 2017, doi: 10.1016/j.knosys.2016.10.015.

[61] J. Jiang, L.-P. Liu, and S. Hassoun, "Learning graph representations of bio-chemical networks and its application to enzymatic link prediction," 2020. Accessed: Aug. 09, 2020. [Online]. Available: https://github.com/HassounLab/ELP.

[62] D. S. Himmelstein and S. E. Baranzini, "Heterogeneous Network Edge Prediction: A Data Integration Approach to Prioritize Disease-Associated Genes," *PLoS Comput. Biol.*, vol. 11, no. 7, 2015, doi: 10.1371/journal.pcbi.1004259.

[63] L.-C. Huang *et al.*, "KinOrtho: a method for mapping human kinase orthologs across the tree of life and illuminating understudied kinases," doi: 10.1101/2021.03.05.434161.

[64] D. Szklarczyk *et al.*, "STRING v11: protein-protein association networks with increased coverage, supporting functional discovery in genome-wide experimental

datasets," *Nucleic Acids Res.*, vol. 47, pp. 607–613, 2018, doi: 10.1093/nar/gky1131.

[65] B. Jassal *et al.*, "The reactome pathway knowledgebase," *Nucleic Acids Res.*, vol. 48, no. D1, pp. D498–D503, Jan. 2020, doi: 10.1093/nar/gkz1031.

[66] A. Bateman *et al.*, "UniProt: The universal protein knowledgebase in 2021," *Nucleic Acids Res.*, vol. 49, no. D1, pp. D480–D489, Jan. 2021, doi: 10.1093/nar/gkaa1100.

[67] J. Mistry *et al.*, "Pfam: The protein families database in 2021," *Nucleic Acids Res.*, vol. 49, 2021, doi: 10.1093/nar/gkaa913.

[68] H. Huang *et al.*, "IPTMnet: An integrated resource for protein post-translational modification network discovery," *Nucleic Acids Res.*, vol. 46, no. D1, pp. D542–D550, 2018, doi: 10.1093/nar/gkx1104.

[69] A. P. Davis *et al.*, "Comparative Toxicogenomics Database (CTD): update 2021," *Nucleic Acids Res.*, no. 1, 2020, doi: 10.1093/nar/gkaa891.

[70] C. Meng, R. Cheng, S. Maniu, P. Senellart, and W. Zhang, "Discovering Meta-Paths in Large Heterogeneous Information Networks," doi: 10.1145/2736277.2741123.

[71] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2016, vol. 13-17-Augu, pp. 855–864, doi: 10.1145/2939672.2939754.

[72] P. Minervini, N. Fanizzi, C. D'amato, and F. Esposito, "Scalable Learning of Entity and Predicate Embeddings for Knowledge Graph Completion." Accessed:

Feb. 17, 2019. [Online]. Available: https://developers.google.com/freebase/data.

[73] A. Grover and J. Leskovec, “node2vec: Scalable Feature Learning for Networks Aditya,” 2016, pp. 855–864, doi: 10.1145/2939672.2939754.

[74] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, “Learning Entity and Relation Embeddings for Knowledge Graph Completion,” in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence Learning*, 2015, pp. 2181–2187, Accessed: Apr. 08, 2019. [Online]. Available: www.aaai.org.

[75] A. Keshavarzi, N. Kannan, and K. Kochut, “RegPattern2Vec: Link Prediction in Knowledge Graphs,” *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, 2021. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9422604 (accessed May 18, 2021).