Learning Hierarchies from Knowledge Graphs

by

Marcin Pietrasik

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Knowledge graphs are data storage structures that rely on principles from graph theory to represent information. Specifically, facts are stored as triples which bring together two entities via a predicate. In a graphical context, these entities are analogous to nodes, and the relations between them are analogous to edges. In recent years, using graph structures to model and store data has been garnering an increasing amount of attention among practitioners in sectors ranging from academia to government to industry. Indeed by some measures, graph database management systems are the fastest growing database type over the past decade and large scale public knowledge bases counting billions of triples have become widely available. The open access to such amounts of graph data has spurred on its use in research related to the Semantic Web, artificial intelligence, and computer science broadly. One field of research which has received less attention is that of learning hierarchies from knowledge graphs.

Learning hierarchies from knowledge graphs refers to a broad concept that encompasses distinct tasks related by their induction of hierarchical relations between knowledge graph entities. Perhaps the simplest reason for learning hierarchical structures is that they organize data in a way that is highly intuitive and interpretable to humans. Indeed, the most widely used knowledge bases are organized by hierarchical structures, namely trees and directed acyclic graphs. That is to say, knowledge graphs are hierarchical at their core. With this in mind, this doctoral work proposes novel methods and models for learning hierarchies from knowledge graphs using artificial intelligence. In doing so, it seeks to advance both hierarchy induction as well as the application of hierarchies to common tasks and problems in the knowledge graph community.

The first problem confronted is that of class taxonomy induction. In this regard, a novel method using a greedy algorithm based on class frequencies and co-occurrences is proposed. It's shown empirically that this method is capable of inducing well structured taxonomies and that it outperforms existing class taxonomy induction methods. The second problem is that of using hierarchies to improve knowledge graph embeddings. Embeddings are one of the most widely investigated knowledge graph representations and their utility reaches far to downstream tasks such as link prediction and entity classification. With this in mind, a meta-strategy involving hierarchically coarsening a knowledge graph before embedding is proposed. This allows the embedding process to be performed on a smaller graph, thereby reducing computational complexity. It's shown that such a strategy is capable of attaining faster and oftentimes higher quality knowledge graph embeddings. The third problem investigated is that of learning a hierarchical clustering of a knowledge graph's entities. To this end, a class of probabilistic models called stochastic blockmodels are leveraged. First, it is shown that blockmodels are capable of modelling the intricacies of knowledge graphs by proposing a novel model for knowledge graph generation. This model fuses blockmodelling together with neural networks in what is a first in the context of knowledge graphs. Empirical results demonstrate that such an approach yields results comparable to state-of-the-art methods on real world datasets. Afterwards, a fully probabilistic model is proposed for hierarchical clustering of a knowledge graph's entities. The model is presented in a non-parametric and fully probabilistic framework, allowing for flexibility in learning the structure of the hierarchy. Results indicate that such an approach is capable of learning a coherent cluster hierarchy as per quantitative and qualitative evaluation.

Preface

This thesis presents original research work conducted by Marcin Pietrasik during his time as a Master's and Doctoral student in the Department of Electrical and Computer Engineering at the University of Alberta working under the supervision of Prof. Marek Reformat.

The thesis contains adapted versions of the following works which were published in the following conference proceedings:

- **Pietrasik, Marcin**, and Marek Reformat. "Neural Blockmodeling for Multilayer Networks." 2021 International Joint Conference on Neural Networks (IJCNN). IEEE, 2021.
- **Pietrasik, Marcin**, and Marek Reformat. "A Simple Method for Inducing Class Taxonomies in Knowledge Graphs." In European Semantic Web Conference, pp. 53-68. Springer, Cham, 2020.

Furthermore, the thesis contains segments of the following work published as a preprint on arXiv:

- **Pietrasik, Marcin**, and Marek Reformat. "Path Based Hierarchical Clustering on Knowledge Graphs." Preprint. 2021.

Finally, the thesis draws heavily from the following completed works, intended for publication and to be submitted before the final defence:

- **Pietrasik, Marcin**, and Marek Reformat. "Hierarchical Blockmodelling for Knowledge Graphs."

- **Pietrasik, Marcin**, and Marek Reformat. "Probablistic Coarsening for Knowledge Graphs Embeddings."

The doctoral work also included significant contributions to the following published work not included in the thesis:

- Zhang, Yujia, **Marcin Pietrasik**, Wenjie Xu, and Marek Reformat. "Hierarchical Topic Modelling for Knowledge Graphs." In European Semantic Web Conference, pp. 270-286. Springer, Cham, 2022.

- Singh, Abhineet, **Marcin Pietrasik**, Gabriell Natha, Nehla Ghouaiel, Ken Brizel, and Nilanjan Ray. "Animal Detection in Man-made Environments." In The IEEE Winter Conference on Applications of Computer Vision, pp. 1438-1449. 2020.

- Yu, Zheng, Xuhui Fan, **Marcin Pietrasik**, and Marek Z. Reformat. "Fragmentation Coagulation Based Mixed Membership Stochastic Blockmodel." In AAAI, pp. 6704-6711. 2020.

- Yu*, Zheng, **Marcin Pietrasik***, and Marek Reformat. "Deep Dynamic Mixed Membership Stochastic Blockmodel." In 2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI), pp. 141-148. IEEE, 2019.

Where the asterisk (*) denotes equal contribution.

In all the aforementioned works where Marcin Pietrasik is listed as the first author, he was solely responsible for all aspects of completing the research. These aspects include and are not limited to: background research; problem formulation and its proposed solution; implementation; evaluation; and manuscript preparation. Due to the fact that portions of the these works appear in segments dispersed throughout the thesis' chapters, we indicate which works constitute each chapter in each corresponding chapter preamble.

Acknowledgements

First of all, I would like to extend my sincerest gratitude to my supervisor, Prof. Marek Reformat, for his guidance throughout my degree. I have developed tremendously as an academic in the past 6 years and am grateful that I could always count on his encouragement and insight during this time. I would also like to thank the remaining members of my supervisory committee, Prof. Petr Musilek and Prof. Witold Pedrycz, for reviewing my work and providing feedback.

Furthermore, I would like to thank my internship providers: ACAMP, Granfiy, FIND AI, and TU Dresden. During these internships I learned the invaluable skill of applying my knowledge outside the world of academia to tackle real world problems. Perhaps it is this exact skill which will prove most important as I continue in my professional journey. Specifically, I would like to thank (in order of acquaintance): Ken Brizel, Dr. Nehla Ghouaiel, Prof. Denilson Barbosa, Dr. Marcin Mizianty, Alireza Haghnegahdar, and Prof. Markus Krötzsch. Without their risk in taking me on, these internships would not have been possible.

Finally, I would like to thank my friends and family for their unconditional love and support. In particular, I would like to thank my parents who have always been there for me. I would also like to thank my little brother for not believing in me and pushing me to prove him wrong.

Table of Contents

List of Tables

List of Figures

Abbreviations

ANN Artificial Neural Network.

ARI Adjusted Rand Index.

AUC Area Under the Receiver Operating Characteristic Curve.

CRP Chinese Restaurant Process.

DDBM Deep Dynamic Mixed Membership Stochastic Blockmodel.

dMMSB Dynamic Mixed Membership Stochastic Blockmodel.

fcMMSB Fragmentation Coagulation Mixed Membership Stochastic Blockmodel.

IRM Infinite Relational Model.

LDA Latent Dirichlet Allocation.

MCMC Markov Chain Monte Carlo.

MI Mutual Information.

MLP Multilayer Perceptron.

MMSB Mixed Membership Stochastic Blockmodel.

MNE Multiplex Network Embedding.

nCRP Nested Chinese Restaurant Process.

NLP Natural Language Processing.

NMI Normalized Mutual Information.

OWL Web Ontology Language.

PDF Probability Density Function.

PMF Probability Mass Function.

PMNE Principled Multilayer Network Embedding.

RDF Resource Description Framework.

URI Universal Resource Identifier.

Chapter 1

Introduction

This thesis outlines the research of Marcin Pietrasik's doctoral work, namely the induction of hierarchical structures from knowledge graphs. It is structured as a summary of five papers captured in four chapters. In addition to this, it provides the motivation for the work as well as its objectives in contributing to the scientific community. Furthermore, it briefly outlines the relevant background information and related works, placing it at the intersection of knowledge graphs and artificial intelligence. Finally, it presents potential directions for future work.

This chapter contains excerpts from the candidacy exam as well as the paper *A Simple Method for Inducing Class Taxonomies in Knowledge Graphs*.

1.1 Motivation

Knowledge graphs are data storage structures that rely on principles from graph theory to represent information. Specifically, facts are stored as triples which bring together two entities via a predicate. In a graphical context, these entities are analogous to nodes, and the relations between them are analogous to edges. In recent years, knowledge graphs have garnered widespread attention as a medium for storing data on the web. Public knowledge graphs such as DBpedia [1], YAGO [2], and WikiData [3] are all underpinned by large-scale knowledge graphs containing upwards of one billion triples each. The last of these, for instance, contains just over 100 million entities as of 2022, a near seven fold increase over its count in 2014. The open access to such amounts of data has spurred on its uses in personal, academic, and commercial domains and knowledge graphs are ubiquitous in the research fields of the Semantic Web, artificial intelligence, and computer science broadly. Furthermore, private companies are known to use proprietary knowledge graphs as a component of their data stores. Google, for instance, uses a knowledge graph derived from Freebase [4] to enhance their search engine results by providing infoboxes which summarize facts about a user's query [5].

In this thesis, the titular *Learning Hierarchies from Knowledge Graphs* refers to a broad concept that encompasses distinct tasks related by their induction of hierarchical relations between knowledge graph entities. The clearest example of a knowledge graph hierarchy is the class taxonomy which organizes a knowledge graph's classes through superclass-subclass relations. The task of inducing such a taxonomy merely amounts to learning how the classes are organized hierarchically in the knowledge graph. Similarly, hierarchical clustering of knowledge graphs allows not only to discover which entities are semantically similar as per the clustering but also how entities relate to one another hierarchically. Perhaps less obviously, knowledge graph coarsening may be viewed as a form of hierarchy learning. This task involves collapsing knowledge graph entities with one another based on some – generally structural – similarity. By iteratively coarsening a knowledge graph, a chain of progressively smaller knowledge graphs is built and a hierarchy emerges. These three tasks are further elucidated upon in this thesis and novel solutions for them are proposed.

The motivating factors behind learning knowledge graph hierarchies are various. Perhaps the simplest is that hierarchical structures organize data in a way that is highly intuitive and interpretable to humans. For instance, a hierarchical clustering of knowledge graph entities makes it apparent which entities constitute the broadest concepts in the knowledge graph and how they relate to their descendants. Similarly, a taxonomy of classes reveals implicit relations between entities through its transitive properties. Put plainly, hierarchies induced from knowledge graphs are useful because they are easy to understand. Indeed, the most widely used knowledge bases such as DBpedia, YAGO, and WikiData are organized by hierarchical structures, namely trees and directed acyclic graphs. That is to say, knowledge graphs are hierarchical at their core. Furthermore, hierarchies are used as components of larger systems to solve common tasks related to knowledge graphs. For instance, hierarchies are used in learning knowledge graph embeddings, both explicitly as an input feature of the model [6] and implicitly as a byproduct of the embedding process [7]. As embedding is one of the most common problems in the knowledge graph community, learning accurate hierarchies is valuable. In summary, the importance of hierarchical learning to knowledge graphs in conjunction with the rise of knowledge graphs as a viable data storage structure provides the motivation for this work.

1.2 Objectives

The objectives of this doctoral work are to develop novel methods and models for learning hierarchies from knowledge graphs using artificial intelligence. Although broad in scope and consisting of four independent projects, the thesis seeks to advance both hierarchy induction as well as application of hierarchies to common tasks and problems in the knowledge graph community. Due to the varied nature of the aforementioned projects, their objectives are summarized independently:

- The first objective is to develop a novel method for class taxonomy induction and hierarchical clustering from knowledge graphs. Developing such a method allows for the automation of class taxonomy construction which, when done manually, is time consuming and requires curators knowledgeable in the area. The dearth of approaches pertaining to this task motivates us to propose a method which outperforms existing methods.

- Afterwards, we will investigate the use of hierarchies to improve knowledge graph embeddings. Specifically, we will propose using hierarchical coarsening as a meta-strategy for embedding with the objective of obtaining higher quality embeddings at reduced training cost. Due to the meta-strategy's independence to the embedding method used, we will perform a pairwise comparison to determine the effectiveness of such an approach

- Furthermore, we will introduce a novel technique for modelling knowledge graphs based on the marriage of blockmodelling and embeddings methods. To our best knowledge, this will be the first fusion of its kind in the context of knowledge graph representation. The objective with this work is to establish the utility of using a blockmodel structure for modelling knowledge graphs. This structure can then be utilized for learning a hierarchical clustering of entities in a subsequent project.

- Finally, we will present a novel stochastic blockmodel for learning a hierarchical clustering of knowledge graph entities. The objective of developing this model is to show that stochastic blockmodels can be used for learning hierarchies from knowledge graphs as such models have not been used for this task thus far. Utilizing a model which is fully non-parametric, allows for greater flexibility in learning the structure of the hierarchy. By developing this model, the foundations were set for further application of stochastic blockmodels to the domain of knowledge graphs.

The aforementioned points are unified in that the learning of hierarchies, whether explicit or implicit, constitutes a significant part of each work. Each of the above objectives relates to a chapter in the thesis, as outlined in what follows.

1.3 Outline

After this introductory chapter, the thesis proceeds with a brief overview of the information and notation necessary for its understanding in Chapter 2. In particular, we formally define knowledge graphs as well as the types of methods we use in our work, namely stochastic blockmodels and artificial neural networks (ANNs). The subsequent four chapters summarize four independent works covering various aspects of hierarchy learning from knowledge graphs. Specifically, Chapter 3 presents the class taxonomy and hierarchy induction methods from the papers *A Simple Method for Inducing Class Taxonomies in Knowledge Graphs* and *Path Based Hierarchical Clustering on Knowledge Graphs*, respectively. The coarsening based meta-strategy for knowledge graph embeddings adapted from *Probabilistic Coarsening for Knowledge Graph Embeddings* is outlined in Chapter 4. Chapter 5 comprises of an extended version of the paper *Neural Blockmodeling for Multilayer Networks* which discusses the aforementioned marriage of stochastic blockmodels with neural networks. The stochastic blockmodel for hierarchical clustering of knowledge graphs, is introduced in Chapter 6 which is in large part taken from the paper *Hierarchical Blockmodelling for Knowledge Graphs.* Finally, the thesis is summarized and possible avenues for future work are discussed in Chapter 7.

Chapter 2

Background

This chapter provides a brief summary to the background information necessary for understanding the thesis. Due to the scope and complexity of the concepts covered in this work, what follows is merely an introduction to each concept and readers are encouraged to follow the citations in each section for a thorough discussion.

This chapter contains sections adapted from the candidacy exam as well as the paper *Hierarchical Blockmodelling for Knowledge Graphs.*

2.1 Knowledge Graphs

We refer to Hogan et al. [8] for their definition of knowledge graphs as *"a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent potentially different relations between these entities."* Concretely, information is stored as a collection of triples wherein each triple relates a subject entity, e_i, to an object entity, e_j, via a predicate, r_r. Formally, we define a knowledge graph, $\mathcal{G}$, as a set such that $\mathcal{G} = \{\langle e_i, r_r, e_j \rangle \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}\}$ where $\langle e_i, r_r, e_j \rangle$ is a triple, $\mathcal{E}$ is the set of entities in $\mathcal{G}$, and $\mathcal{R}$ is the set of predicates in $\mathcal{G}$. When put together, the triples form a directed graph with vertices corresponding to entities and edges corresponding to predicates. Each triple in a knowledge graph describes one piece of information, or a fact. For instance, ⟨`dbr:Henry_Ford, dbo:Occupation, dbr:Engineer`⟩ relates the subject `dbr:Henry_Ford` to the object

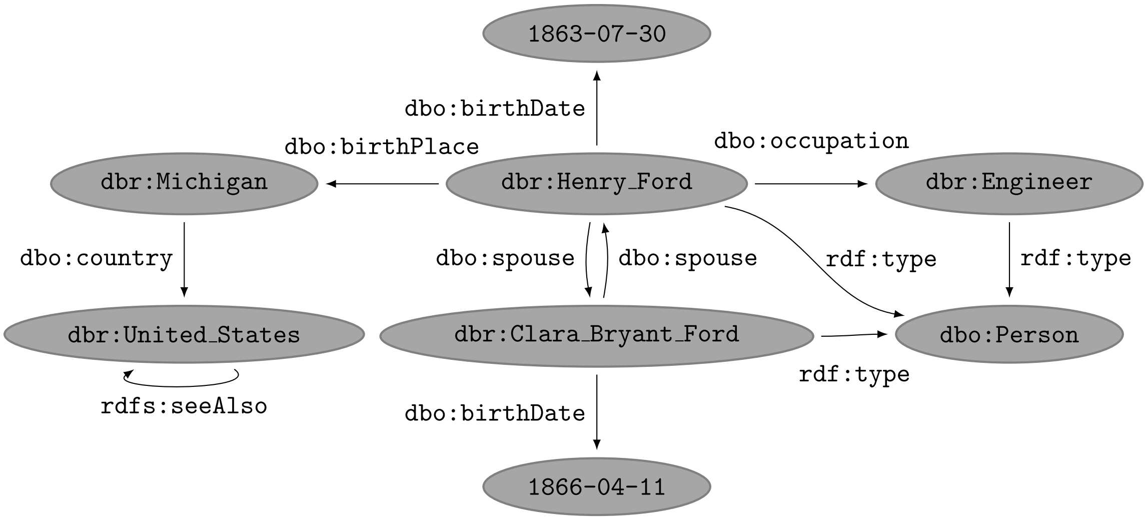


Figure 2.1: Excerpt of DBpedia triples describing Henry Ford and his wife Clara.

`dbr:Engineer` through the predicate `dbo:Occupation` and states that, in plain English, Henry Ford's occupation is an engineer. When put together, triples form a graph as shown in Figure 2.1 which visualizes a subset of DBpedia triples describing Henry Ford and his wife Clara. Notice that knowledge graphs allow for cycles and self-relations as shown through the two `dbo:spouse` and `rdfs:seeAlso` relations, respectively. This is further made clear when analyzing a knowledge graph's binary adjacency tensor which may be symmetric and containing non-zero values in its main diagonal. To this point, knowledge graphs are oftentimes represented in their tensor form as it allows for easier numerical operation and thus opens the door to various tools and methods in artificial intelligence. A binary adjacency tensor is obtained from a knowledge graph by ordering its entities and predicates along an $|\mathcal{E}| \times |\mathcal{E}| \times |\mathcal{R}|$ tensor, $\mathbf{G}$, that takes on values $g_{ijr} = 1$ if there exists a triple in $\mathcal{G}$ from entity e_i to entity e_j on predicate r_r and $g_{ijr} = 0$ otherwise. A comprehensive introduction to knowledge graphs is provided by Gutierrez and Sequeda [9].

2.1.1 Resource Description Framework

The prefixes `dbr`, `dbo`, `rdf`, and `rdfs` in Figure 2.1 are artefacts of the notation used as part of the Resource Description Framework (RDF)[1]. RDF refers to a set of standards introduced by the World Wide Web Consortium [2] for representing and exchanging data on the web. It uses the triple structure as its foundation and includes notations and formats for serializing triples. For instance, in the N-Triples format, the three leftmost triples in Figure 2.1 are written as:

```
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema> .
dbr:Michigan dbo:country dbr:United_States .
dbr:United_States rdfs:seeAlso dbr:United_States .
dbr:Henry_Ford dbo:birthPlace dbr:Michigan .
```

Where the first three lines define Universal Resource Identifiers (URIs) as prefixes before using them to define the triples themselves. Using such prefixes decreases the size of storing triples and enhances readability. This type of format may be queried using languages such as SPARQL [10] and Cypher [11].

2.1.2 Ontologies

Ontologies are often used in conjunction with knowledge graphs to provide an axiomatic foundation on which knowledge graphs are built. In this view, an ontology may be seen as a vocabulary and a rule book that provides semantics to a knowledge graph and governs how the information contained within it is represented and how it can be reasoned with. Perhaps the most salient feature of ontologies is their definition of knowledge graph classes as well as rules for how they are organized and interact with one another. In adding a layer which conceptually sits on top of a knowledge

[1] https://www.w3.org/RDF/

[2] https://www.w3.org/

graph, ontologies both enrich the knowledge graph by allowing it to be reasoned with in more complex ways and constrain it by introducing boundaries for what is and isn't possible. Consider, for instance, the classes `Parent` and `Child` and the predicate `hasChild`. An ontology can provide structure to the predicate by defining its domain and range. Specifically, it can state that the subject related by the predicate `hasChild` has to be of class `Parent` and the object of class `Child`. In the Web Ontology Language (OWL) – a widely used collection of languages for defining ontologies – this constraint may look as follows:

```
<owl:ObjectProperty rdf:ID="hasChild">
 <rdfs:domain rdf:resource="Parent"/>
 <rdfs:range rdf:resource="Child"/>
</owl:ObjectProperty>
```

In addition to the aforementioned constraints on domain and range, ontologies allow for defining subsumption, transitivity, symmetricity, cardinality, equivalence, set operations, and enumeration amongst many more. Of particular importance to this thesis is the use of subsumption rules, which define superclass-subclass relationships between classes. This allows for building of class taxonomies to organize concepts hierarchically. Consider, for instance, the following definitions:

```
<owl:Class rdf:ID="Dog">
 <rdfs:subClassOf rdf:resource="Mammal"/></owl:Class>
<owl:Class rdf:ID="Cat">
 <rdfs:subClassOf rdf:resource="Mammal"/></owl:Class>
<owl:Class rdf:ID="Mammal">
 <rdfs:subClassOf rdf:resource="Animal"/></owl:Class>
<owl:Class rdf:ID="Turtle">
 <rdfs:subClassOf rdf:resource="Reptile"/></owl:Class>
<owl:Class rdf:ID="Reptile">
 <rdfs:subClassOf rdf:resource="Animal"/></owl:Class>
```

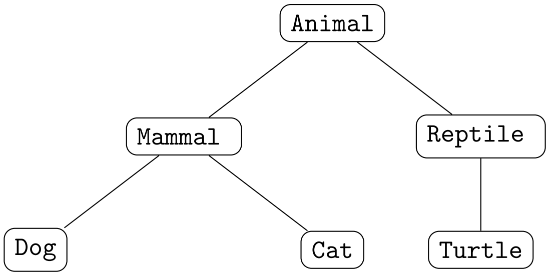


Figure 2.2: Toy example of class taxonomy describing the relations between different animal classes.

These rules form a taxonomy between five animal classes as shown in Figure 2.2. Building such hierarchies enriches a knowledge graph as it allows for reasoning that instances of the `Dog` class are also instances of `Mammal` and `Animal`. Furthermore, it provides a conceptual basis for how classes are related to one another. `Dog` and `Cat`, for example, are more closely related conceptually than `Dog` and `Turtle`.

2.2 Embeddings

In mathematics, embeddings are defined as an injective mapping between two objects which preserves properties in the domain. Formally, an embedding function, f, maps object x to its embedding y as $f : x \mapsto y$. A more granular definition of property preservation as well as the constraints and conditions of the mapping vary depending on the types of objects being embedded and the branch of mathematics being studied.

In this thesis, and in artificial intelligence more generally, embeddings are used in a narrower context. Specifically, embeddings are continuous vector representations of entities, mapped from their original, or ambient, space. The vector space that is formed by the set of embeddings is called the embedding space and is usually of a lower dimension than the ambient space. Generally, embeddings rely on the intuition that entities which are embedded close to one another in the embedding space share similar semantics or properties which are intended to be preserved in the embedding process. Such representations are desired as they are both intuitive and highly operable by machines. For instance, embeddings allow for the calculation of distance between any two entities, revealing potentially useful information. Furthermore, if the ambient space is discrete, the embedding process provides a representation which opens the door to the range of tools and methods in artificial intelligence which operate on continuous inputs.

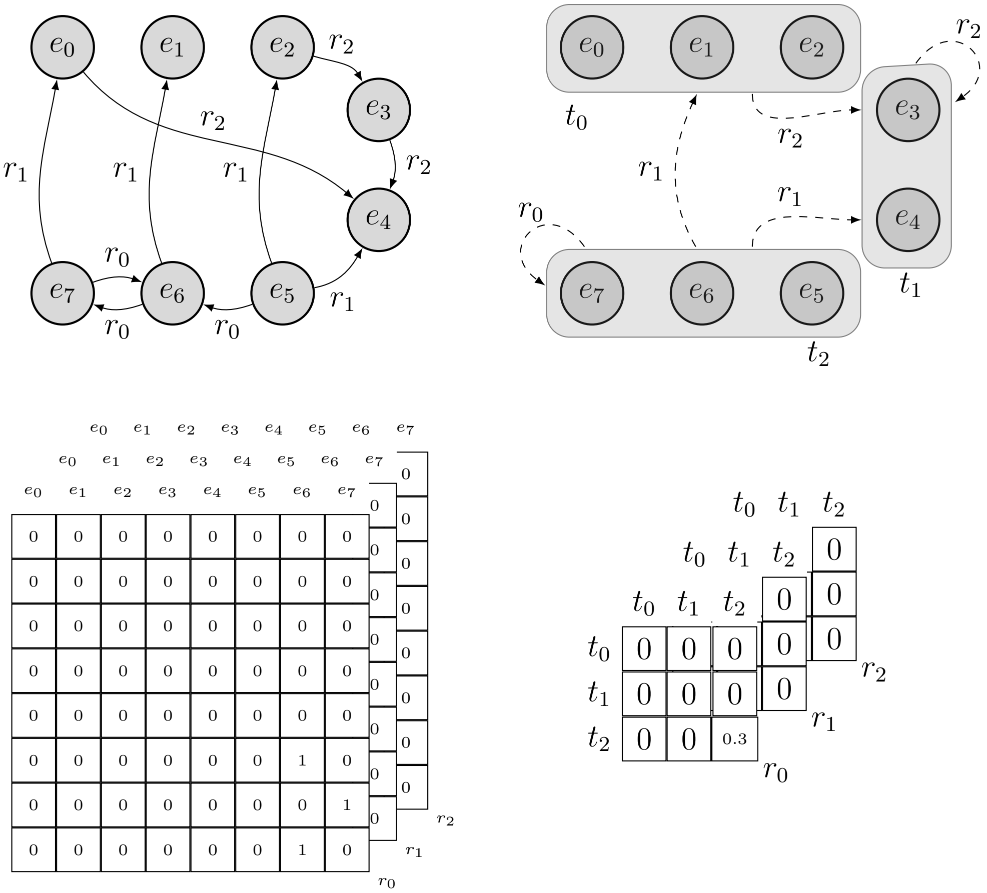
Graph embeddings are continuous vector representations of a graph's entities and, in some cases, relations. Thus, the task of generating graph embeddings involves finding a function, f, which maps each entity to the embedding space. Defined formally, $f : \mathcal{E} \mapsto \mathbb{R}^{|\mathcal{E}| \times d}$ where d is the dimensionality of the embedding space such that $d << |\mathcal{E}|$. The obtained embeddings are subsequently used to solve downstream tasks such as link prediction and entity classification.

2.3 Probabilistic Graphical Models

Probabilistic graphical models are – as their name suggests – probabilistic models in which the conditional dependencies between their variables are structured as a graph. The range of models and approach which fall under this category is too broad to be adequately introduce in this thesis. We thus focus on just the concepts which are utilized in subsequent chapters.

2.3.1 Stochastic Blockmodels

Stochastic blockmodels are a class of probabilistic graphical models used for generating random graphs with roots in the fields of social science and mathematics. First proposed in 1983 by Holland et al. [12] for modelling social networks, they have expanded their utility to fields such as biochemistry [13], education [14], and artificial intelligence [15–17] among others. In simplest terms, stochastic blockmodels are a type of Bayesian non-parametric graph partition model in that their approach relies on grouping graph entities together via partitions – often referred to as blocks – which share similar structural properties. The generative process by which this partitioning occurs is realized by sampling from a set of probability distributions, giving rise



$$\begin{array}{c|cccccccc}
 & e_0 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\
\hline
e_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
e_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
e_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
e_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
e_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
e_5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
e_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
e_7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}$$

$$\begin{array}{c|ccc}
 & t_0 & t_1 & t_2 \\
\hline
t_0 & 0 & 0 & 0 \\
t_1 & 0 & 0 & 0 \\
t_2 & 0 & 0 & 0.3
\end{array}$$

Figure 2.3: Toy example of a knowledge graph and how it may be modelled by a stochastic blockmodel. Starting from top left quadrant and proceeding clockwise: graphical representation of a knowledge graph with entities e_0 through e_7 and predicates r_0 through r_2; graphical representation of the aforementioned knowledge graph as modelled by a stochastic blockmodel with communities t_0 through t_2; potential community relations tensor induced by stochastic blockmodel; adjacency tensor of knowledge graph above it.

to the stochasticity of stochastic blockmodels. The learning process is then to infer the parameters of these distributions using a Bayesian inference scheme. Figure 2.3 provides a toy example of a knowledge graph along with how it may be modelled by a stochastic blockmodel.

Because stochastic blockmodels constitute a heterogeneous class of methods, describing them by referring to a concrete instance is bound to include definitions which do not apply to all members of the class. With this in mind, our introduction to stochastic blockmodels draws on their key characteristics to motivate a toy stochastic blockmodel for generating a knowledge graph. All stochastic blockmodels are defined by a set of probability distributions which are sampled from to generate the adjacency tensor of the knowledge graph, $\mathbf{G}$. In order to perform this generation, the knowledge graph's entities must first be assigned to one of the model's communities. This is done by sampling the model's variables responsible for this assignment. Let $\mathbf{A}$ be a tensor representing these variables with a corresponding hyperparameter of α responsible for parameterizing their prior distribution. In stochastic blockmodels, the probability of an interaction between two entities is modelled as the degree of interaction between their respective communities. It is necessary, therefore, to capture these community interactions by sampling their corresponding model variables. Let $\mathbf{B}$ be a tensor representing this subset of variables with a prior hyperparameter of β. The joint distribution of this model is obtained by applying the chain rule of probability as follows:

$$\mathbb{P}(\mathbf{G}, \mathbf{A}, \mathbf{B} \mid \alpha, \beta) = \prod_{g_{ijr} \in \mathbf{G}} \mathbb{P}(g_{ijr} | \mathbf{A}_{ijr}, \mathbf{B}_{ijr}, \alpha, \beta) \mathbb{P}(\mathbf{B}_{ijr} \mid \mathbf{A}_{ijr}, \beta) \mathbb{P}(\mathbf{A}_{ijr} \mid \alpha) \tag{2.1}$$

Where $\mathbf{A}_{ijr}$ and $\mathbf{B}_{ijr}$ indicate the latent variables in $\mathbf{A}$ and $\mathbf{B}$ associated with sampling g_{ijr} and α and β are their respective hyperparameters. Notice that the probability of drawing a value in the knowledge graph's adjacency tensor, $\mathbb{P}(g_{ijr} | \mathbf{A}_{ijr}, \mathbf{B}_{ijr}, \alpha, \beta)$, is conditioned on $\mathbf{A}_{ijr}$ and $\mathbf{B}_{ijr}$. Thus, in order to generate the knowledge graph, it's necessary to first infer the values of $\mathbf{A}_{ijr}$ and $\mathbf{B}_{ijr}$. This inference process is analogous to the training phase of other machine and deep learning models. In most cases, the solution is intractable for exact inference and must be approximated using an inference scheme. Perhaps the simplest inference scheme used in stochastic blockmodelling is Gibbs sampling, a Markov chain Monte Carlo method which can be used for sampling from a joint distribution. Gibbs sampling approximates this distribution by iteratively sampling from its variables' full conditional distributions. This iterative sampling creates a Markov chain of samples wherein its stationary distribution approximates the joint distribution of the model. Continuing the example above, to infer the blockmodel's parameters for g_{ijr}, namely $\mathbf{A}_{ijr}$ and $\mathbf{B}_{ijr}$, inference is performed on their conditional distributions $\mathbb{P}(\mathbf{A}_{ijr} \mid \mathbf{G}, \mathbf{B}, \alpha)$ and $\mathbb{P}(\mathbf{B}_{ijr} \mid \mathbf{G}, \mathbf{A}, \beta)$, respectively. We apply Bayes' theorem to obtain these distributions. Recall that by this theorem the posterior distribution is proportional to the product of the likelihood and the prior. We can therefore express the conditionals of $\mathbf{A}_{ijr}$ and $\mathbf{B}_{ijr}$ as follows:

$$\mathbb{P}(\mathbf{A}_{ijr} \mid \mathbf{G}, \mathbf{B}, \alpha) \propto \mathbb{P}(\mathbf{G} \mid \mathbf{A}, \mathbf{B})\mathbb{P}(\mathbf{A}_{ijr} \mid \alpha) \tag{2.2}$$

$$\mathbb{P}(\mathbf{B}_{ijr} \mid \mathbf{G}, \mathbf{A}, \beta) \propto \mathbb{P}(\mathbf{G} \mid \mathbf{B}, \mathbf{A})\mathbb{P}(\mathbf{B}_{ijr} \mid \beta) \tag{2.3}$$

Where $\mathbb{P}(\mathbf{G} \mid \mathbf{A}, \mathbf{B})$ and $\mathbb{P}(\mathbf{G} \mid \mathbf{B}, \mathbf{A})$ are the likelihoods, and $\mathbb{P}(\mathbf{A}_{ijr} \mid \alpha)$ and $\mathbb{P}(\mathbf{B}_{ijr} \mid \beta)$ are the priors of $\mathbf{A}_{ijr}$ and $\mathbf{B}_{ijr}$, respectively. The likelihood may be understood as the chance observing the data given the model parameters. In Equations 2.2 and 2.3, it is the likelihood of drawing $\mathbf{G}$ from our model with parameters $\mathbf{A}$ and $\mathbf{B}$. The prior represents the assumptions about a variable before any data is taken into account. They are oftentimes chosen in order to leverage a conjugacy with their dependant variables. Priors are parameterized by hyperparameters which must be specified a priori. The choice of these hyperparameters influences the density of the prior and can thus change the output of the model. Gibbs sampling draws from the variables' full conditional distributions iteratively for a predetermined number of iterations, $iters$. To highlight this, the superscript $iter$ is added to denote the value of a variable at the corresponding iteration. The Gibbs sampling process may be summarized as follows:

1. Initialize $\mathbf{A}^0_{ijr}$ and $\mathbf{B}^0_{ijr}$ for each $g_{ijr} \in \mathbf{G}$
2. For iteration $iter$ in $1, 2, ..., iters$
   (a) Draw $\mathbf{A}^{iter}_{ijr} \sim \mathbb{P}(\mathbf{A}^{iter}_{ijr} \mid \mathbf{G}, \mathbf{B}^{iter-1}, \alpha)$ for each $g_{ijr} \in \mathbf{G}$ using Equation 2.2
   (b) Draw $\mathbf{B}^{iter}_{ijr} \sim \mathbb{P}(\mathbf{B}^{iter}_{ijr} \mid \mathbf{G}, \mathbf{A}^{iter-1}, \alpha)$ for each $g_{ijr} \in \mathbf{G}$ using Equation 2.3

In the first step, variables can be initialized by sampling from their prior distributions or specified explicitly if a priori evidence to suggest their true values exists. The second step depicts the iterative sampling of model variables from their full conditionals. We note that samples obtained early in this process may be drawn from a distribution distant to that of the desired stationary distribution. As such it is necessary to discard the samples obtained before this distribution has been reached. This process is commonly referred to as burning in the Gibbs sampler and the number of discarded samples as the burn in samples. Furthermore, as successive samples in this process are autocorrelated, there may be a lag period applied in obtaining results such that samples in during the lag period are also discarded. Thus, if our toy example performs 1000 iterations with a burn in of 900 and a lag of 10, only 9 samples will be obtained as the output of the Gibbs sampler. These 9 samples are then aggregated over to account for the stochasticity in sampling from the posterior and arrive at a final result. The process by which these samples are aggregated are model specific and may be as simple as merely taking the sampled mode. An introduction to Gibbs sampling and related sampling schemes is covered by Mackay [18] and a thorough discussion of stochastic blockmodels along with their concrete examples is provided by Abbe [19].

2.3.2 The Chinese Restaurant Process

The Chinese restaurant process (CRP) [20] is a discrete stochastic process that yields a probability distribution in accordance with the preferential attachment principle. In this view, it is both a Dirichlet process [21] as it generates a probability distribution and a preferential attachment process [22] as the distribution is generated such that probabilities are proportional to past draws. The process is explained through a metaphor of sitting patrons at a Chinese restaurant. Consider this restaurant as containing an infinite number of tables with each table having the capacity to seat an infinite number of patrons. Patrons are seated sequentially, such that the first patron is seated at the first table and every subsequent patron may be seated at an occupied table or the first unoccupied table. The probability of being seated at an occupied

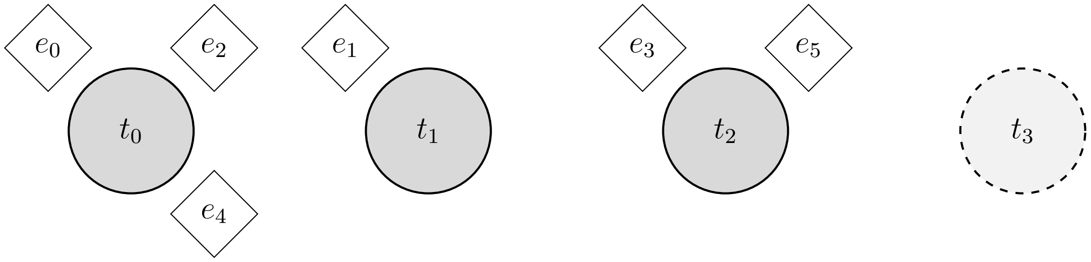


Figure 2.4: Toy example of the CRP after sitting patrons e_0 through e_5. Tables t_0 through t_2 are occupied and table t_3 is the next unoccupied table. We illustrate Equation 2.4 by calculating the probabilities of sitting patron p_6 at tables t_0 and t_3: $\mathbb{P}(e_6 = t_0) = \frac{3}{6+\gamma}$ and $\mathbb{P}(e_6 = t_2) = \frac{\gamma}{6+\gamma}$.

table is proportional to the number of patrons already seated at it. This process is illustrated through the toy example in Figure 2.4 which shows a potential state of the CRP after sitting six patrons along with the sample probabilities of sitting the seventh. Formally, the probability of seating patron e_i at a table t_m in a restaurant where $\mathcal{T}_i$ is the set of occupied tables when patron e_i arrives is:

$$\mathbb{P}(e_i = t_m \mid e_0, e_1, ..., e_{i-1}, \gamma) = \begin{cases} \dfrac{\#_i^m}{i+\gamma} & t_m \in \mathcal{T}_i \\ \dfrac{\gamma}{i+\gamma} & t_m \notin \mathcal{T}_i \end{cases} \tag{2.4}$$

Here, $\#_i^m$ is the number of patrons seated at table t_m when patron e_i arrives and $\gamma > 0$ is a hyperparameter of the CRP responsible for controlling the probability that an incoming patron is seated at an unoccupied table such that increasing γ increases this probability. Thus, increasing γ values will yield results with an increasing number of occupied tables. Specifically, the expected number of occupied tables grows logarithmically with respect to the number of seated patrons:

$$\mathbb{E}\left[\sum_{t_m \in \mathcal{T}_i} \mathbb{I}(\#_i^m > 0) \mid \gamma\right] = \mathcal{O}(\gamma \log i) \tag{2.5}$$

Where $\mathbb{I}$ is the indicator function which returns 1 if the condition is met and 0 otherwise. Big-O notation is leveraged with $\mathcal{O}$ to indicate the asymptotic upper bound of the expectation. This principle become relevant when controlling for the branching factor of the induced tree as we will see later on. The realization of the

CRP yields a partition of patrons over the infinitely many tables in the restaurant. If we consider each table to be a community, we can leverage this process to obtain a probability distribution over an infinite number of communities. Indeed this is the main utility of the CRP, namely to serve as a conjugate prior to infinite non-parametric discrete distributions. While this approach allows for the modelling of flat communities, it does not account for hierarchical relations between them. To remedy this, the CRP must be extended to its nested variant.

2.3.3 The Nested Chinese Restaurant Process

The nested Chinese restaurant process (nCRP) [23, 24] is an extension of the CRP formulated to account for hierarchical relations between the generated communities. The realization of this process is an infinitely deep and infinitely branching tree of communities defined by a set of paths, $\mathcal{P}$, taken from the root community to a leaf community. In principle, the tree is unbounded in depth, however we limit our discussion to a nCRP bounded to a depth of L. As in the case of the CRP, the allocation of paths along the tree is consistent with the preferential attachment principle. The tree is generated stochastically by sampling a path at each level in the tree via the CRP such that drawing a table is analogous to taking a path at that level. To extend the metaphor of seating patrons at a Chinese restaurant, consider the scenario of an infinite number of restaurants with an infinite number of infinite seat tables. When patrons are seated at these restaurants they are not served food but rather a table specific reference to another restaurant to which they must go. One of these restaurants is designated a root restaurant with no reference and all other restaurants are referenced exactly once. The seating of patrons at these restaurants is performed as in the CRP. We can see how realizing this process yields a tree by examining the paths taken by patrons. They first arrive at the root restaurant before being sent off to one of the root restaurant's descendant restaurants. At this restaurant the patron is sent off to another descendant restaurant and this process is repeated until

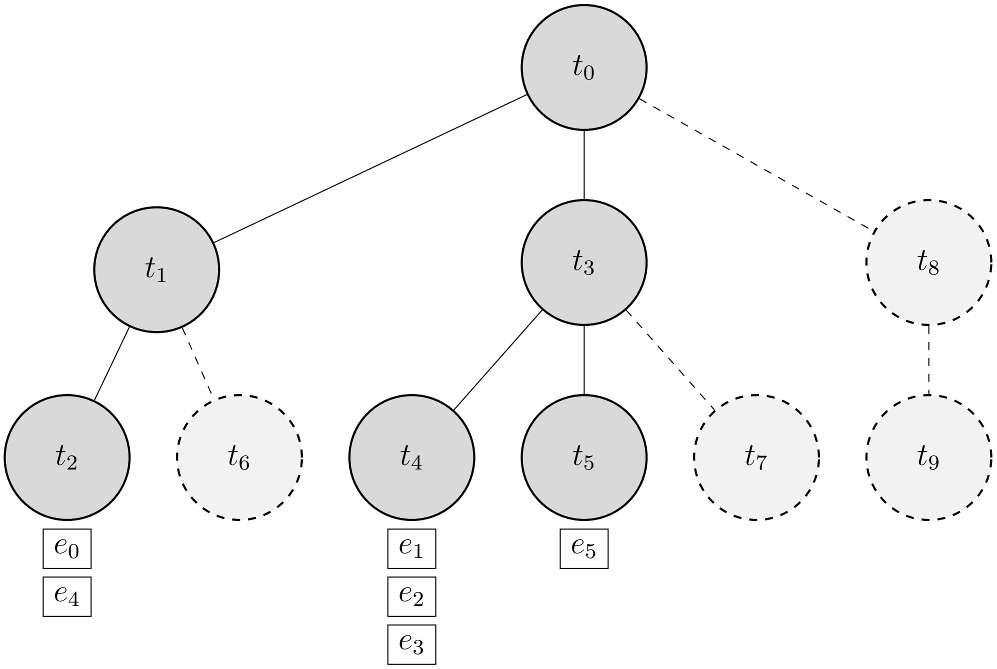


Figure 2.5: Toy example of a nCRP truncated to a depth of $L = 2$ after assigning patrons e_0 through e_5. Solid lines indicate paths which have been taken by patrons and thus exist in the tree whereas dashed lines indicate indicate potential paths. We illustrate Equation 2.4 by calculating the probability of a patron taking a path through communities t_2 and t_9: $\mathbb{P}(e_6 = t_2) = (\frac{2}{2+\gamma})(\frac{2}{6+\gamma})$ and $\mathbb{P}(e_6 = t_9) = \frac{\gamma}{6+\gamma}$.

L restaurants have been visited in the bounded case. The paths taken by patrons generate the tree as illustrated in the toy example in Figure 2.5. As before, we extend this analogy of patrons and tables to entities and communities, respectively. Thus, when drawing path $\mathbf{p}_i$ for entity e_i, the process starts by initializing the path at the top level to the root community, namely $p_i^0 = t_0$ where the superscript in p_i^0 indexes into the path vector to obtain the community at the corresponding level and t_0 is the root community. The process then continues by drawing a descendent community according to the CRP. Recall that this draw results in a community which either has or hasn't been visited before by a previous entity. The latter case corresponds to branching out a new path in the tree at the descendant level. This process is repeated L times at which point the specified depth has been reached. We can formalize this process by extending the previously defined notation. Specifically, let $\mathcal{T}_i$ be the set of communities in the tree before entity e_i has its path sampled and $\mathcal{C}_i^q$ be the set of children communities for community t_q at this time as well. The sampling process is then expressed as follows: when entity e_i arrives at community t_q on the $(l-1)^{\text{th}}$ level in the tree , the probability of selecting an existing community, $p_i^l \in \mathcal{C}_i^q$ or creating a new community, $p_i^l \notin \mathcal{T}_i$, is:

$$\mathbb{P}(p_i^l = t_c \mid \mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_{i-1}, \mathbf{p}_i^{1:\,l-1}, \gamma) = \begin{cases} \dfrac{\#_i^{t_c}}{\#_i^{t_q} + \gamma} & t_c \in \mathcal{C}_i^c \\ \dfrac{\gamma}{\#_i^{t_q} + \gamma} & t_c \notin \mathcal{T}_i \end{cases} \tag{2.6}$$

Where $\#_i^{t_q}$ and $\#_i^{t_c}$ is the number of entities that have passed through communities t_q and t_c before entity e_i started its path. The superscript in $\mathbf{p}_i^{1:l-1}$ indicates that the probability distribution for sampling p_i^l is conditioned on the path taken by entity e_i up until level l. The hyperparameter γ serves a similar function as in the CRP, namely controlling the branching factor of the tree such that higher γ values yield trees with more branches. The use of the CRP in the path decision process ensures that the probability mass will be pulled towards drawing paths which have been more frequently drawn before. The resulting distribution allows us to use the nCRP as a non-parametric prior over a tree structure in our model. In drawing paths, we not only generate a hierarchy but also define a subset of communities to which an entity can belong to, namely those along the path.

2.3.4 The Stick Breaking Process

The stick breaking process [25] is – like the CRP and nCRP – a Dirichlet process that draws its name from a metaphor which describes it. The metaphor starts by breaking a stick of unit length into two fragments at a point in the interval from 0 to 1 as drawn from the Beta distribution. One of the two fragments is preserved and the other fragment is broken again, analogously to the initial stick. This process is repeated an infinite number of times to yield an infinite number of fragments whose combined length is that of the initial stick. These fragments may be viewed as a probability distribution over the infinite sequence of discrete time-steps used to generate them.

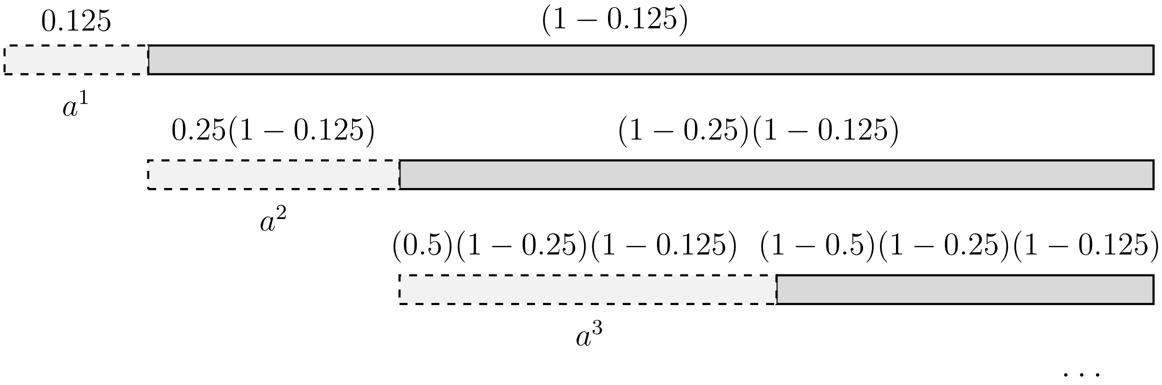


Figure 2.6: Toy example of the stick breaking process with values $v^1 = 0.125$ $v^2 = 0.25$ $v^3 = 0.5$. Starting at the top of the figure, a unit length stick is broken at v^1. The remainder is then iteratively broken proportionally to draws from the Beta distribution.

In other words, the stick breaking process is an infinite extension of the Dirichlet distribution insofar as while the Dirichlet distribution yields a probability distribution over L categories, the stick breaking process yields a probability distribution over an *infinite* number of categories. Formally, let the draw from the Beta distribution at the l^{th} iteration of the stick breaking process be denoted as $v^l \sim \text{Beta}(\mu\sigma, (1-\mu)\sigma)$. Thus, the lengths of the first fragment, denoted a^1, and its remainder are v^1 and $1 - v^1$, respectively. To obtain the length of the second fragment, a^2, draw v^1 and break off that fragment from what remains of the stick, namely $a^2 = v^1(1 - v^1)$. We define this process for an arbitrary l^{th} time-step as follows:

$$a^l = v^l \prod_{k=1}^{l-1} (1 - v^k) \tag{2.7}$$

A concrete example involving the application of this rule is illustrated in Figure 2.6 which demonstrates the first three breaks of the stick along with the respective values of the broken fragments and their remainders. The realized stick fragments form a probability distribution in that $\sum_{l=1}^{\infty} a^l = 1$. We can thus define the probability mass function of the stick breaking process, denoted Stick(μ, σ), as follows:

$$\text{Stick}(\mu, \sigma) = \sum_{l=1}^{\infty} a^l$$

$$= \sum_{l=1}^{\infty} v^l \prod_{k=1}^{l-1} (1 - v^k) \tag{2.8}$$

The stick breaking process is a generalization of the Griffiths-Engen-McCloskey distribution [24, 26] which may be seen as a special case where $\mu\sigma = 1$. The hyperparameters, $1 > \mu > 0$ and $\sigma > 0$, control the mean and variance of the distribution, respectively. Specifically, increasing μ values will pull the mean towards fragments broken later in the process and increasing σ values will increase the variance of the distribution.

2.4 Artificial Neural Networks

Artificial neural networks comprise a class of models whose underlying inspiration is the intelligence of living organisms, specifically the biological neuron of animals. Neurons are a type of cell found in the nervous system and are composed of three main parts: the soma, the axon, and dendrites. They operate by sending and receiving chemical and electrical signals to and from cells around them via tiny gaps called synapses. Specifically, when a neuron receives a chemical signal via the stimulation of its dendrites, it transmits it along as an electrical signal through its axon to stimulate surrounding cells. When these signals are transmitted to other neurons, they form what is called a neural circuit. Neural circuits form the basis of intelligent life on earth and are responsible for a broad range of tasks from simple movements to higher order thinking. Currently, understanding of how neurons give rise to such complex processes is still in its infancy. Part of this is due to the high number of neurons interacting with one another and the combinatoric nature of these interactions. To highlight this, the human brain, for instance, has approximately 100 billion neurons [27] and 60 trillions neuronal connections [28]. These hindrances to biological understanding have not prevented the adoption of neuronal concepts to the field of artificial intelligence. Indeed, one of the most successful areas of artificial intelligence research is related to the study of ANNs. These networks mimic the structure neural circuits through

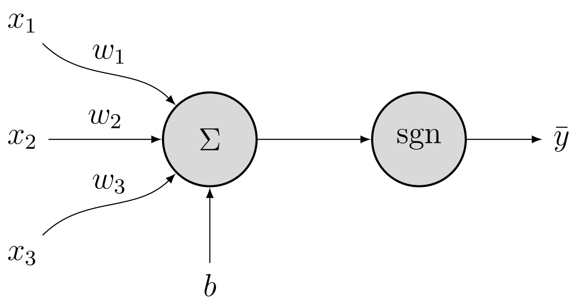


Figure 2.7: Visualization of a three dimensional input Perceptron.

the interaction of artificial neurons. The canonical example of this is the Multilayer Perceptron (MLP) which catapulted the field of neural networks in the 1980s. Since then the MLP has evolved into the most commonly used neural networks today such as convolutional neural networks [29], recurrent neural networks [30], and autoencoders [31, 32]. Compared to other [33] classification techniques, neural networks have several advantages, including: self-adaptation [34], universal function approximation [34, 35], and flexibility [34]. Going as far back as the mid 1990s, neural networks were being applied to solve practical problems in commerce, science, industry, and medicine [36]. Furthermore, they've won contests and set benchmarks in speech recognition [37], character recognition [38], and language identification [39] among other tasks. This gives reason, outside of a strictly theoretical nature, to pursue the study and understanding of neural networks. A recent review of neural networks is provided by Samek et al. [40].

2.4.1 Rosenblatt's Perceptron

The MLP is best understood by first examining Rosenblatt's Perceptron. Introduced in 1958 by Frank Rosenblatt [41], the Perceptron takes one or more inputs and associates a weight with each input. The product of these inputs and their corresponding weights is summed along with a bias term. The result is called the induced local field and is then passed through the sign function [42], denoted sgn, giving a result of -1 if the induced local field is negative and 1 if the induced local field is positive. These values correspond to two class labels to which the input is classified. Figure 2.7 shows — from left to right -– the flow of a three input Perceptron. The Perceptron's output, $\bar{y}$, can be expressed mathematically. Given that $\mathbf{x}$ is vector of inputs, $\mathbf{w}$ is a vector of its corresponding weights, b is a scalar bias term, and sgn is the sign function, $\bar{y}$ is as follows:

$$\bar{y} = \mathrm{sgn}(\mathbf{x}^T\mathbf{w} + b) \tag{2.9}$$

Finding weights which result in the correct classification of the input is done iteratively, adjusting the weights after every misclassification of input $\mathbf{x}$. After a time step, denoted n, the update of the weights for the next time step, $n+1$, is computed as follows:

$$\mathbf{w}^{n+1} = \mathbf{w}^n + \eta(y - \bar{y})\mathbf{x}^n \tag{2.10}$$

Where the bias, b, is treated as a weight with a corresponding fixed input of 1, y is the desired classification of input $\mathbf{x}^n$ such that $y \in \{-1, 1\}$ and η is the learning parameter such that $\eta > 0$. This update rule has been proven to converge when the data it's classifying is linearly separable [43]. For data which is not linearly separable, such as the XOR function, the update rule fails to guarantee weights which correctly classify the inputs [44]. It is this problem which is solved by the MLP.

2.4.2 The Multilayer Perceptron

The MLP uses a form of artificial neuron that is similar to that of the Perceptron. The induced local field is the summation of the product of inputs and their weights plus a bias term. The result of which is passed through an activation function which must be differentiable for all real numbers [42]. This requirement precludes the use of the signum function used in Rosenblatt's perceptron. Many [45] functions can be used in its place and such as the logistic sigmoid and the hyperbolic tangent. We use the notation ϕ to denote an arbitrary, differentiable activation function.

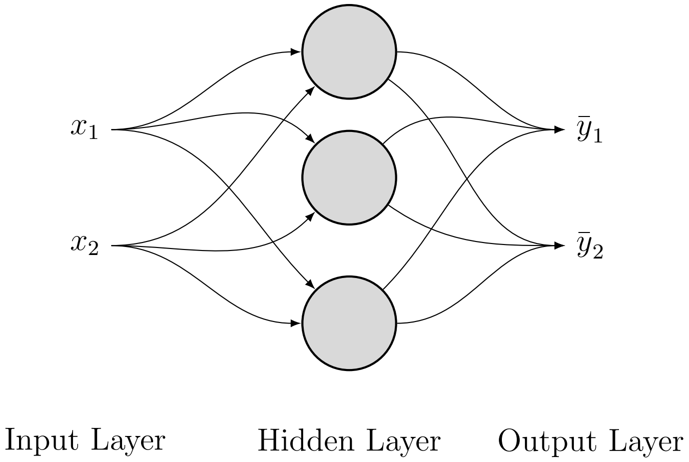


Figure 2.8: Visualization of a two dimensional input, two dimensional output MLP with one hidden layer of three artificial neurons.

In the MLP, the neuron is used many times throughout the network and is grouped according to when it is activated after the network receives an input. This grouping is called a layer and is comprised of one or more nodes. A MLP network must have at least three layers: one input layer, one output layer, and one or more hidden layers. The input layer processes the features of what is to be classified. Input layer nodes don't perform any calculation but rather send the input signals to the hidden layer. Each node in a hidden layer is an artificial neuron whose output connects to another hidden layer or the output layer. Like the hidden layer, the output layer nodes are artificial neurons, the output of which is a final output of the network. Figure 2.8 shows a network with one hidden layer having two input nodes, three hidden nodes, and two output nodes. The signals travel from left to right in what is called the forward step.

In a fully connected MLP network, each signal will pass through every node in the hidden and output layers. This means that the extent to which each hidden layer node impacts the final output cannot be calculated as in Rosenblatt's Perceptron. The problem of updating the weights is solved by backpropagation.

2.4.3 Backpropagation

Backpropagation is a technique used to find out to what extent connections between neurons are responsible for the output of a neural network and how to change their weights to move towards a more desirable network. It occurs after a forward pass of the neural network starting at the output layer and working its way backward. What follows is a simplified explanation of the backpropagation algorithm using gradient descent as described by Haykin [42] and Rumelhart et al. [31].

Consider a neuron, a, in the hidden layer and a neuron, b, in the subsequent layer. The connection weight between the neurons at time step n is written as w_{ab}^n. If E^n is a measure of the error of the entire network, the negative of its partial derivative with respect to w_{ab}^n indicates the direction in weight space that the weight adjustment should be made. This can be calculated with repeated application of the chain rule, the result of which is:

$$\frac{\partial E^n}{\partial w_{ab}^n} = -e_b^n \cdot \phi_b'(v_b^n) \cdot \bar{y}_a^n \tag{2.11}$$

Where e_b^n is the error signal produced by neuron b, $\phi_b'(v_b^n)$ is the derivative of the activation function given the induced local field of neuron b at time step n, and $\bar{y}_a^n$ is the output of neuron a. The local gradient, δ_b^n is defined as $\delta_b^n = -e_b^n \cdot \phi_b'(v_b^n)$ giving rise to the following:

$$\frac{\partial E^n}{\partial w_{ab}^n} = \delta_b^n \cdot \bar{y}_a^n \tag{2.12}$$

Given this information, w_{ab}^n can be updated via gradient descent. In this process, w_{ab} moves down the gradient at each time step by an amount determined by the learning rate, η, giving the weight correction equation:

$$\Delta w_{ab}^n = \eta \cdot \delta_b^n \cdot \bar{y}_a^n \tag{2.13}$$

For output neurons, applying this equation is simple because all the values are available from the forward pass or can be easily calculated. Such is the case for e^n which is calculated the same way as in Rosenblatt's perceptron, that is the difference between the desired output y and the neuron output $\bar{y}^n$, namely $e^n = y - \bar{y}^n$. In the case of hidden nodes, calculating e^n requires a different formula because the desired output is not known. The steps required to solve this problem are omitted from this explanation but can be found in Haykin [42]. The resulting local gradient for hidden neurons is as follows:

$$\delta_a^n = \phi_a'(v_a^n) \sum_{b \in \mathcal{B}} \delta_b^n \cdot w_{ab}^n \tag{2.14}$$

Where $\mathcal{B}$ is the set of neurons that neuron a is connected to in the subsequent layer. These formulas used in conjunction with one another, starting from the output layer working backwards allow the updating of weights by the weight correction equation, Δw_{ab}^n.

An extension to the backpropagation with gradient descent algorithm is to add the concept of momentum. In this scheme, the weight correction is not only affected by the local gradient, but also by the change in weight at the previous time step, Δw_{ab}^{n-1}. Intuitively, momentum adds the effect of inertia as w_{ab} moves on the error surface. This helps speed up learning especially when the weight is stuck in plateaus where the gradient is low. Momentum is controlled by the momentum constant, α, and is expressed as follows:

$$\Delta w_{ab}^n = -\eta \cdot \delta_b^n \cdot \bar{y}_a^n + \alpha \cdot \Delta w_{ab}^{n-1} \tag{2.15}$$

Such that higher values of α result in more inertia from the previous weight change being considered.

Chapter 3

A Simple Method for Inducing Class Taxonomies in Knowledge Graphs

This section summarizes the work presented in the papers: *A Simple Method for Inducing Class Taxonomies in Knowledge Graphs* [46] which was published in the proceedings of the 17$^{\text{th}}$ European Semantic Web Conference; and *Path Based Hierarchical Clustering on Knowledge Graphs* [47] which exists as a preprint on arXiv.

3.1 Introduction

One of the core components of an ontology is the class taxonomy: a set of subsumption axioms between the type classes that may exists in the knowledge graph. When put together, the subsumption axioms form a hierarchy of classes where general concepts appear at the top and their subconcepts appear as their descendants. A challenge that arises when working with large knowledge graphs is that of class taxonomy construction. Manual construction is time consuming and requires curators knowledgeable in the area. DBpedia, for instance, relies on its community to curate its class taxonomy. Similarly, YAGO relies on a combination of information from Wikipedia and WordNet, both of which are manually selected and organized. On the other hand, automated methods are not able to induce class taxonomies of the quality necessary to reliably apply to complex knowledge graphs. Furthermore, they oftentimes rely on external information which may itself be manually curated or may only be applicable to knowledge graphs in a particular domain. With this in mind, the impetus for automatically inducing class taxonomies of high quality from large-scale knowledge graphs becomes apparent.

In this chapter, we propose a scalable method for inducing class taxonomies from knowledge graphs without relying on information external to the knowledge graph's triples. Our approach applies methods used to solve the problem of tag hierarchy induction, which involves inducing a hierarchy of tags from a collection of documents, and identifying the tags that annotate them. Although extensively studied in the field of natural language processing, these methods have yet to be applied to knowledge graphs to the best of our knowledge. In order to use these methods, we reshape the knowledge graph's triple structure to a tuple structure, exploiting the graph's single dimensionality in assigning entities to type classes. Using the new structure, we construct a novel approach to inducing class taxonomies which outperforms existing tag hierarchy induction methods both in terms scalability and quality of induced taxonomies. Finally, we show that an induced class taxonomy may be used as the foundation for performing hierarchical clustering on the knowledge graph's subjects. The idea behind this is that each class in the taxonomy may serve as a hierarchical cluster, reducing the clustering procedure to merely assigning each entity to one class in the taxonomy. Empirical evaluation demonstrates that this process constructs coherent hierarchical clusters.

3.2 Related Work

We divide our discussion of related work into three subsections: class taxonomy induction methods, tag hierarchy induction methods, and hierarchical clustering methods for knowledge graphs. The first two methods are used to construct a hierarchy of concepts, however they differ in the type of data they are applied to. Class taxonomy induction methods are used on knowledge graphs and thus operate on data represented as triples. Tag hierarchy induction methods operate on documents and the tags that annotate them. In practice, these documents are often blog posts, images, and videos annotated by users on social networking websites. We can view our proposed method as a combination of the aforementioned categories as it takes the input structure of documents and tags but is applied to knowledge graphs to induce a class taxonomy. Hierarchical clustering methods seek to learn clusters of knowledge graph entities based on shared semantics and organize them hierarchically such that descendant clusters contain more specific instances of their corresponding ancestors.

3.2.1 Methods for Class Taxonomy Induction

Völker and Niepert [48] introduce Statistical Schema Induction which uses association rule mining on a knowledge graph's transaction table to generate ontology axioms. Each row in the transaction table corresponds to a subject in the graph along with the classes it belongs to. Implication patterns which are consistent with the table are mined from this table to create candidate ontology axioms. The candidate axioms are then sorted in terms of descending certainty values and added greedily to the ontology only if they are logically coherent with axioms added before them.

Nickel et al. [49] propose a method using hierarchical clustering on a decomposed representation of the knowledge graph. Specifically, they extend RESCAL [50], a method for factorizing a three-way tensor, to better handle sparse large-scale data and apply OPTICS [51], a density based hierarchical clustering algorithm.

Ristoski et al. [52] rely on entity and text embeddings in their proposed method, TIEmb. The intuition behind this approach is that entities of a subclass will be embedded within their parent class's embeddings. Thus if you calculate the centroid for each class's embeddings, you can infer its subclasses as those whose centroid falls within a certain radius. For instance, the class centroids of *Mammals* and *Reptiles* will fall inside the radius of *Animals* although the converse is not true since *Mammals* and *Reptiles* are more specific classes and are expected to have a smaller radius.

3.2.2 Methods for Tag Hierarchy Induction

Heymann and Garcia-Molina [53] propose a frequency based approach using cosine similarity to calculate tag generality. In their approach, tags are assigned vectors based on the amount of times they annotate each document. The pairwise cosine similarity between tag vectors is used to build a tag similarity graph. The closeness centrality of tags in this graph is used as the generality of tags. To build the hierarchy, tags are greedily added – in order of decreasing generality – as children to the tag in the hierarchy that has the highest degree of similarity. This approach was extended by Benz et al. [54] to better handle synonyms and homonyms in the dataset.

Schmitz [55] unveils a method extending on the work done by Sanderson and Croft [56] which uses subsumption rules to identify the relations between parents and children in the hierarchy. The subsumption rules are calculated by tag co-occurrence and filtered to control for "idiosyncratic vocabulary". These rules form a directed graph which is then pruned to create a tree. Solskinnsbakk and Gulla [57] use the Aprioir algorithm [58] to mine a set of association rules from the tags. Each of these rules has the relationship of premise and consequence which the authors treat as that of class and subclass. This is used to construct a tree which is then verified based on the semantics of each tag.

The application of Latent Dirichlet Allocation (LDA) [59] to generate topics comprised of tags is proposed in Tang et al. [60]. Generality can then be calculated following the reasoning that tags with high frequencies across many topics are more general than ones that have a high frequencies in a single topic. Relations between tags are induced based on four divergence measures calculated on the LDA results. Agglomerative Hierarchical Clustering for Taxonomy Construction [61] avoids explicitly computing tag generality by employing agglomerative clustering and selecting cluster medoids to be promoted upwards in the hierarchy. Cluster medoids are chosen based on a similarity metric calculated as the divergence between a tag's topic distributions as learned by LDA.

Wang et al. [62] introduce a taxonomy generation method based on repeated application of k-medoids clustering. As the distance metric necessary for k-medoids clustering, they propose a similarity score based on the weighted sum of document and textual similarities. Levels in the hierarchy are created by repeated application of k-medoids clustering such that for each cluster, the cluster medoid becomes the parent of all other tags in the cluster.

A supervised learning approach is used in Dong et al. [63] where binary classifiers are trained to predict a "broader-narrower" relation between tags. LDA is used to generate topic distributions for tags which act as a basis for three sets of features used to train the classifier. This approach does not guarantee that the relations between tags will form a rooted tree.

3.2.3 Methods for Hierarchical Clustering

In an early method, Roy et al. [64] sample a graph from a generative model in a fashion reminiscent of blockmodelling. The model is learned by performing inference on its parameters via the Metropolis-Hastings algorithm. A consequence of this process is the generation of a tree describing entity similarity. Nickel et al. [49] perform hierarchical clustering on latent representations learned by the aforementioned RESCAL method. Using these latent representations has the advantage of being agnostic to the underlying hierarchical clustering method used, allowing for flexibility to adapt to different data.

In an approach which bears similarity to our own, Chen and Reformat [65] describe each subject in a knowledge graph by its predicate-object pairs. These pairs are then used to calculate a similarity matrix between subjects on which agglomerative hierarchical clustering is performed using the extended Ward's minimum variance [66] as its measure. Mohamed [67] takes a similar approach wherein subjects which are described by the same predicate-object pairs are assigned to the same groups. The similarity between these groups is then calculated to construct a hierarchy. An embedding based approach was used in Martel and Zouaq [68] wherein embeddings are first learned on a knowledge graph before being clustered using hierarchical agglomerative clustering and assigned types. This type of clustering was used in the field of cybersecurity in Ding et al. [69] wherein a bag-of-words representation of a knowledge graph served as input.

3.3 Problem Description

Given a triple in the form $\langle e_i, r_r, e_j \rangle$, we can think of predicate-object pairs, $\langle r_r, e_j \rangle$, as tags that describe the subject entity, e_i. In this view, each entity that takes on the role of subject is annotated by tags, $t_j \in \mathcal{A}_i$, where $\mathcal{A}_i$ is the set of tags that annotate e_i. We call these entities documents, $d_i \in \mathcal{D}$, such that the set of all documents is a subset of all entities, $\mathcal{D} \subseteq \mathcal{E}$. Tags are defined as predicate-object pairs, $t := \langle r_r, e_j \rangle$, and belong to the set of all tags, the vocabulary, denoted as $\mathcal{V}$, i.e., $t_j \in \mathcal{A}_i$ and $\mathcal{A}_i \subseteq \mathcal{V}$. For a concrete example of this notation consider DBpedia, wherein the entity `dbr:Canada` is annotated by the tags ⟨`dbo:capital,dbr:Ottawa`⟩, ⟨`dbo:currency,dbr:Canadian_dollar`⟩, ⟨`rdf:type,dbo:Location`⟩, and ⟨`rdf:type, dbo:Country`⟩ amongst others. In this view, the knowledge base $\mathcal{G}$ may be represented as the set of document-tag tuples $\mathcal{G} = \{\langle d, t \rangle \in \mathcal{D} \times \mathcal{V}\}$, where $\langle d, t \rangle$ is the tuple that relates document d with tag t. We refer to this notation as the tuple structure for the remainder of the chapter.

Information in knowledge graphs is often structured using an ontology, which provides semantics to the knowledge graph's triples through an axiomatic foundation which defines how entities and predicates associate with one another. A key component of most ontologies is the class taxonomy which organizes classes through a set of class subsumption axioms. These subsumption axioms may be thought of as is-a relations between classes. For instance, in the DBpedia class hierarchy, the subsumption axioms {`dbo:Person` → `dbo:Artist`} and {`dbo:Artist` → `dbo:Painter`} imply that `dbo:Painter` is a `dbo:Artist` and that `dbo:Artist` is a `dbo:Person`. Furthermore, since class subsumption axioms are transitive, `dbo:Painter` is a `dbo:Person`. This taxonomy oftentimes takes the form of a rooted tree with a root class of which all other classes are considered logical descendants of.

The problem of class taxonomy induction from knowledge graphs involves generating subsumption axioms from triples to build the class taxonomy. We notice that in most knowledge graphs, subjects are related to their class type by one predicate. This has the effect of reducing the knowledge graph's class identifying triples to a single dimension. The property can be exploited in the tuple structure, since all class identifying predicates are the same, they can be ignored without loss of information. For instance, in DBpedia the predicate which relates subjects to their class is `rdf:type`. Thus, when compiling a dataset of class identifying tuples, we can treat the tags $\langle\texttt{rdf:type,dbo:Country}\rangle$ and `dbo:Country` as equivalent. Therefore, the tuple $\langle\texttt{dbr:Canada}, \texttt{dbo:Country}\rangle$ preserves all information required to induce a class taxonomy. This can be exploited by tag hierarchy induction methods which take documents and their tags as input.

3.4 Proposed Method

Our proposed method uses class frequencies and co-occurrences to calculate similarity between tags. This approach, inspired by the method proposed by Schmitz, relies on the intuition that subclasses will co-occur in documents with their superclasses more often than with classes they are not logical descendants of. Unlike Schmitz's method which uses this assumption to generate candidate subsumption axioms, our method uses similarity to choose a parent tag which already exists in the taxonomy. In this step, which draws inspiration from Heymann and Garcia-Molina, tags are greedily added to the taxonomy in order of decreasing generality. Thus, subsumption axioms induced by our method have to abide by the following rules:

- The parent tag has a higher generality than the child tag.
- The parent tag is the tag with the highest similarity to the child tag from the tags that exist in the taxonomy when the child tag is being added.

We can populate the induced class taxonomy with documents, which has the effect of hierarchically clustering the knowledge graph's subject entities. The process for this is to merely find the class, in the hierarchy, to which the document belongs to and assign it to that class. We can then treat each class as a cluster and its constituent documents as cluster elements. The result of this is a hierarchical structure of clusters annotated by tags and with strong inheritance properties.

As previously mentioned, our approach leverages the tuple structure of a knowledge graph to induce a class taxonomy in the form of a rooted tree. As such, the first step is data preprocessing wherein all of a knowledge graph's class identifying triples are converted to tuple structure.

3.4.1 Class Taxonomy Induction Procedure

Before describing the taxonomy induction procedure for our method, we define measures which are calculated on the knowledge graph as required input for our algorithm.

- The number of documents annotated by tag t_a is denoted as D_a.
- The number of documents annotated by both tags t_a and t_b is denoted as D_{ab}. We note that this measure is symmetrical, i.e. $\mathrm{D}_{ab} = \mathrm{D}_{ba}$.
- The generality of tag t_a, denoted as G_a, measures how general the concept described by the tag is and how high it belongs in the taxonomy. The generality is defined as:

  $$\mathrm{G}_a = \sum_{t_b \in \mathcal{V}_{-t_a}} \frac{\mathrm{D}_{ab}}{\mathrm{D}_b} \tag{3.1}$$

  Where $\mathcal{V}_{-t_a}$ is the set of all tags excluding tag t_a.

Algorithm 1 Procedure for Class Taxonomy Induction

Input: knowledge graph in tuple structure in a form of sets $\mathcal{D}$ and $\mathcal{V}$; document counts annotated by tag(s) D; generality of tags G; decay factor α

Output: induced class taxonomy subsumption axioms $\mathcal{T}$ and $\mathcal{T}*$

```
Sort tags in order of decreasing generality $G_i$, create $\mathcal{V}_{sorted}$
Initialize taxonomy with root tag equal to the tag with highest generality, $\mathcal{T} = \{\{\emptyset \rightarrow \mathcal{V}_{sorted}[0]\}\}$
Initialize the set of tags that have already been added to the taxonomy, $\mathcal{T}* = \{\mathcal{V}_{sorted}[0]\}$
for b = 1, 2, ..., $|\mathcal{V}_{sorted}|$ do
    $t_b = \mathcal{V}_{sorted}[b]$
    $maxSimTag = \mathcal{V}_{sorted}[0]$
    $maxSimValue = 0$
    for $t_a \in \mathcal{T}*$ do
        Calculate $S_{a \rightarrow b}$ using Equation 3.2
        if $S_{a \rightarrow b} > maxSimValue$ then
            $maxSimTag = t_a$
            $maxSimValue = S_{a \rightarrow b}$
        end if
    end for
    $\mathcal{T} = \{maxSimTag \rightarrow t_b\} \cup \mathcal{T}$
    $\mathcal{T}* = t_b \cup \mathcal{T}*$
end for
```

Having calculated the aforementioned measures, we proceed by sorting tags in the order of decreasing generality and store them as $\mathcal{V}_{sorted}$. The first element of this list, $\mathcal{V}_{sorted}[0]$, is semantically the most general of all tags and becomes the root tag of the taxonomy. The taxonomy, $\mathcal{T}$, is represented as a set of subsumption axioms between parent and child tags. Formally, each subsumption between parent tag, t_{parent}, and child tag, t_{child}, is represented by $\{t_{parent} \rightarrow t_{child}\}$ such that $\{t_{parent} \rightarrow t_{child}\} \in \mathcal{T}$. The taxonomy is therefore initialized with the root tag as $\mathcal{T} = \{\{\emptyset \rightarrow \mathcal{V}_{sorted}[0]\}\}$ where $\emptyset$ represents a null value, i.e. no parent.

Following initialization, the remaining tags are added to the taxonomy in terms of decreasing generality by calculating the similarity between the tag being added, t_b, and all the tags already in the taxonomy, $\mathcal{T}*$. The tag $t_a \in \mathcal{T}*$ that has the highest similarity with tag t_b becomes the parent of t_b and $\{a \rightarrow b\}$ is added to $\mathcal{T}$. The similarity between tags t_a and t_b, denoted as $\mathrm{S}_{a\to b}$, measures the degree to which tag t_b is the direct descendant of tag t_a. It is calculated as the degree to which tag t_b is compatible with tag t_a and all the ancestors of t_a:

$$\mathrm{S}_{a\to b} = \sum_{t_c \in \mathcal{P}_a} \alpha^{l_a - l_c} \frac{\mathrm{D}_{b,c}}{\mathrm{D}_b} \tag{3.2}$$

Where $\mathcal{P}_a$ is the path in the taxonomy from the root tag $\mathcal{V}_{sorted}[0]$ to tag t_a. l_a and l_c denote the levels in the hierarchy of tags t_a and t_c, respectively. The levels are counted from the root tag starting at zero. Thus, the level of $\mathcal{V}_{sorted}[0]$, denoted as $l_{\mathcal{V}_{sorted}[0]}$, is equal to zero, the levels of its children are equal to one, and so on. The decay factor, α, is a hyperparameter that controls the effect ancestors of tag t_a have on its similarity when calculating $\mathrm{S}_{a\to b}$. By setting the value of α such that $0 < \alpha < 1$, we ensure that the effect is lower the more distant an ancestor tag is. The cases were $\alpha = 0$ and $\alpha = 1$ correspond to ancestors having no effect and equal effect on the similarity, respectively. We explore the effect various α values have on the induced class taxonomy in the following section. The full details of our method's procedure are outlined in Algorithm 1.

3.4.2 Hierarchical Clustering Procedure

We can use the induced taxonomy as the foundation of a hierarchical clustering of documents, i.e. the knowledge graph's subject entities. The taxonomy is used to initialize the clusters such that each tag in the taxonomy becomes a cluster and the hierarchical relations between tags are extended to the clusters. The tags may then be seen as annotations for each cluster. We exploit this in our notation such that c_a is the cluster initialized from tag t_a. Documents are assigned to clusters by the degree to which they belong to a cluster. Belonging of document d_i to cluster c_a, denoted $\mathrm{B}_{i\to a}$, is calculated as the Jaccard coefficient between the document's tags, $\mathcal{A}_i$, and the tags encountered in the path from the root cluster to cluster c_a, denoted

Algorithm 2 Procedure for Hierarchical Clustering

Input: knowledge graph in tuple structure in a form of sets $\mathcal{D}$ and $\mathcal{V}$; class taxonomy as subsumption axioms $\mathcal{T}$ and $\mathcal{T}*$; paths in hierarchy to clusters $\mathcal{P}_a$; decay factor α

Output: cluster hierarchy as subsumption axioms $\mathcal{T}$; cluster members $\mathcal{C}$

```
Initialize all clusters $\mathcal{C}$ as empty
for $d_i \in \mathcal{D}$ do
    $maxBelClus = None$
    $maxBelValue = 0$
    for $c_a \in \mathcal{T}*$ do
        Calculate $\mathrm{B}_{i \to a}$ using Equation 3.3
        if $\mathrm{B}_{i \to a} > maxBelValue$ then
            $maxBelClus = c_a$
            $maxBelValue = \mathrm{B}_{i \to a}$
        end if
    end for
    $\mathcal{C}_{maxBelClus} = \mathcal{C}_{maxBelClus} \cup d_i$
end for
Prune cluster hierarchy defined by $\mathcal{T}$ and $\mathcal{C}$ recursively
```

$\mathcal{P}_a$. Formally, this is:

$$\mathrm{B}_{i \to a} = \frac{|\mathcal{A}_i \cap \mathcal{P}_a|}{|\mathcal{A}_i \cup \mathcal{P}_a|} \tag{3.3}$$

Each document is added to the cluster to which it has the highest degree of belonging. We denote the set of documents that belong to cluster c_a as $\mathcal{C}_a$. The process of assigning documents to clusters may be parallelized to increase performance.

This process may induce a hierarchy containing empty clusters which need to get pruned. Pruning is performed by traversing the hierarchy depth first and removing all empty clusters. In addition, non-empty clusters which have empty parent clusters are reattached as the children of their first non-empty ancestor. If a non-empty cluster has no non-empty ancestors, it becomes the child of the root. The root cluster is never removed, regardless of whether it is empty or not. The hierarchical clustering process is summarized in Algorithm 2.

3.5 Evaluation

Evaluation of class taxonomy induction methods is difficult as there may be several equally valid taxonomies for a dataset. Previous works such as Gu et al. [70] and Wang et al. (2009) [71] have opted for human evaluation, wherein domain experts assess the correctness of relations between classes. Wang et al. (2012) [62] used domain experts to rank entire paths on a three point scale. Others, such as Liu et al. [72] and Almoqhim et al. [73], compare class relations against a gold standard taxonomy. In this approach, a confusion matrix between class subsumption axioms is calculated between the induced and gold standard taxonomies. When a gold standard taxonomy can be established, it is the preferred evaluation method as it provides an objective measurement; as such, it is the one we use in our work. We use the confusion matrix to derive the harmonic mean between precision and recall, the F_1 score [74], as our evaluation metric:

$$\begin{aligned} precision &= \frac{TP}{TP+FP} && (3.4)\\ recall &= \frac{TP}{TP+FN} && (3.5)\\ F_1 &= 2 * \frac{precision * recall}{precision + recall} && (3.6) \end{aligned}$$

Where TP, FP, and FN are the number of true positives, false positives, and false negatives, respectively. Since the F_1 score is also used in evaluating the quality of the cluster hierarchy, we use the notation Tax-F_1 to refer to the F_1 score calculated on the induced and gold standard taxonomies.

We evaluate the hierarchical clustering by calculating the F_1 score of: the belonging of documents to clusters (Doc-F_1); and how well clusters represent the tags in the vocabulary (Tag-F_1). Doc-F_1 and Tag-F_1 highlight the trade-off between large, heterogeneous clusters on a strongly heritable hierarchy (favoured by Doc-F_1) and smaller homogeneous clusters on a less heritable hierarchy (favoured by Tag-F_1). For obtaining the former, each cluster inherits all the documents of its descendant clusters and the F_1 score is calculated such that a document is correctly assigned to a cluster if both document and cluster are annotated by the same tag. The latter is obtained in a way similar to the technique used in Nickel et al. [49]. As before, each cluster inherits all the documents of its descendant clusters and the F_1 score between each tag and each cluster is calculated. The F_1 that is highest among the clusters becomes the score of the tag.

For the remainder of this section, we first evaluate the effect of our method's hyperparameter, α, on each of the four datasets and provide suggestions for selecting the α value when applying our method to other datasets. This is followed by a comparison our method to the aforementioned Heymann and Garcia-Molina method, Schmitz method, as well as results from the literature. We also provide visualizations of excerpts from the class taxonomies induced by our method on the Life, DBpedia, and IIMB datasets. Finally, our method's computational complexity and the effect of dataset size on induced taxonomies are evaluated.

3.5.1 Datasets

We evaluate the method on four real-world datasets generated from public online knowledge bases: Life, DBpedia, WordNet, and IIMB. All four datasets as well as their respective gold standard class taxonomies were generated or acquired during the month of November 2019.

Life

The Life dataset was generated by querying the Catalogue of Life: 2019 Annual Checklist (CoL) [75], an online database that indexes living organisms by their taxonomic classification. One hundred thousand living organisms were randomly selected from the GBIF Type Specimen Names [76], an online checklist of 1226904 organisms, and queried on CoL at each of their taxonomic ranks to generate the document-tag tuples. The resulting dataset takes the form such that each organism is a document and its membership at each taxonomic rank is a tag related by `is-a`. For instance, the document `Canis_latrans` (coyote) will have the tags ⟨`is-a, Mammalia`⟩ and ⟨`is-a, Canidae`⟩. Furthermore, to anchor the class taxonomy to a root tag, we added the tag ⟨`is-a, LivingOrganism`⟩ to every document. We note that even though the number of taxonomic ranks is fixed, most organisms in the database are not defined on all of them. As such, the number of tags per document varies from two to ten. In total, there are 100000 documents and 37368 unique tags. Since the dataset itself is classified in the correct taxonomic order, the Life gold standard taxonomy could simply be obtained by querying for subsumption axioms from the dataset.

DBpedia

The DBpedia dataset was generated by randomly querying for 50000 unique subjects in DBpedia for which there exists a triple where the subject is related to a DBpedia class object (an object having the prefix `dbo:`) via the predicate `rdf:type`. These 50000 subjects become the documents in the tuple structure. Following this step, all the triples for each document having the tag form ⟨`rdf:type, dbo:*`⟩ were queried to make the document-tag tuples. (`dbo:*` represents any object with the prefix `dbo`.) In total, 205793 triples were used to create the dataset with 418 unique tags. The DBpedia gold standard taxonomy was taken from the DBpedia ontology class mappings which can be found on the DBpedia website[1]. At the time of querying, the ontology had 765 classes, 418 of which were present in the dataset. This difference made it necessary to include only those subsumption axioms for which parent and child tags exist in the dataset when computing the confusion matrix. This is similar to the dataset generated in Ristoski et al. [52] where the number of classes present in their dataset was 415.

[1] http://mappings.dbpedia.org/server/ontology/classes/

Dataset	# Triples	# Documents	# Tags
Life	726042	100000	37368
DBpedia	205793	50000	418
WordNet	392846	50000	1752
IIMB	4793	1416	82

Table 3.1: Summary of the datasets used in this chapter.

WordNet

The WordNet dataset was generated by querying DBpedia for subjects of types that exist in WordNet [77], an English language lexical database. Fifty thousand subjects having a WordNet class object related by `rdf:type` were queried. In DBpedia, WordNet class objects use the `yago:` prefix, giving the tag format ⟨`rdf:type, yago:*`⟩. This process yielded a dataset comprised of 50000 documents and 1752 unique tags generated from 392846 triples. To generate the WordNet gold standard taxonomy, DBpedia was queried to learn the relations between WordNet classes through the `rdfs:subClassOf` predicate. In this process, `yago:PhysicalEntity100001930` is set as the root class and the taxonomy is built by recursively querying for subclasses using `rdfs:subClassOf` as the predicate. This process builds a taxonomy of 30722 tags. To fit the 1752 tags present in the dataset, it was necessary to collapse the gold standard taxonomy. This was done by removing tags in the gold standard taxonomy that are missing in the dataset and adopting orphaned tags with the nearest ancestor existing in the dataset.

IIMB

The IIMB dataset [78] is a benchmark created by the 2010 Ontology Alignment Evaluation Initiative to evaluate instance matching techniques and tools. The dataset contains 1416 documents and 4793 triples which describe facts about popular movies including: titles, genres, actors, locations, etc. The dataset is structured into five top-level tags to which all other tags belong: Location, Language, Film, Creature, and Budget. We manually added a root tag to anchor the dataset such that there are 82 unique tags in total.

3.5.2 Hyperparameter Sensitivity

We evaluate our method's sensitivity to the decay factor, α, by performing a hyperparameter sweep on each of the four datasets. In this process, our method is applied five times on each dataset for α values starting at $\alpha = 0.05$ and increasing by increments of 0.05 up until $\alpha = 0.95$. This process is analogous to increasing the relative importance of ancestor tags when calculating tag similarity. Furthermore, since similarity is calculated as a summation, increasing α will favour placing tags lower in the taxonomy. The F_1 scores are calculated and their means at each α value are displayed graphically in Figure 3.1. For clarity, we omit graphing the mean F_1 scores at $\alpha = 0$ as the values are disproportionately low for all four datasets ($F_1 < 0.1$). This is because when $\alpha = 0$, the similarity gets reduced to $\mathrm{S}_{a \to b} = \mathrm{D}_{a,b}/\mathrm{D}_b$ which has the effect of inducing shallow taxonomies with most tags as children of the root tag.

For class taxonomy induction, cursory inspection of the Tax-F_1 scores shows that there is no clear behaviour that α exhibits which is constant across datasets. This is also apparent when comparing the optimal α values: 0.95, 0.70, 0.35, and 0.4 for Life, DBpedia, WordNet, and IIMB datasets, respectively. Furthermore, we notice that as α increases, the trend follows three different patterns: stable, generally increasing, and generally decreasing. A possible reason for the relative stability of α on the Life dataset is its consistency. Due to the strict requirements for source datasets to be included in CoL, all entries are well scrutinised. As such, tags will always appear with their ancestors in the same documents. For example, all 893 instances of the tag Mammalia co-occur with the tag's ancestors Animalia, Chordata, and LivingOrganism. In this scenario, there is less information to be gained by incorporating information from higher up in the taxonomy. On the other hand, the

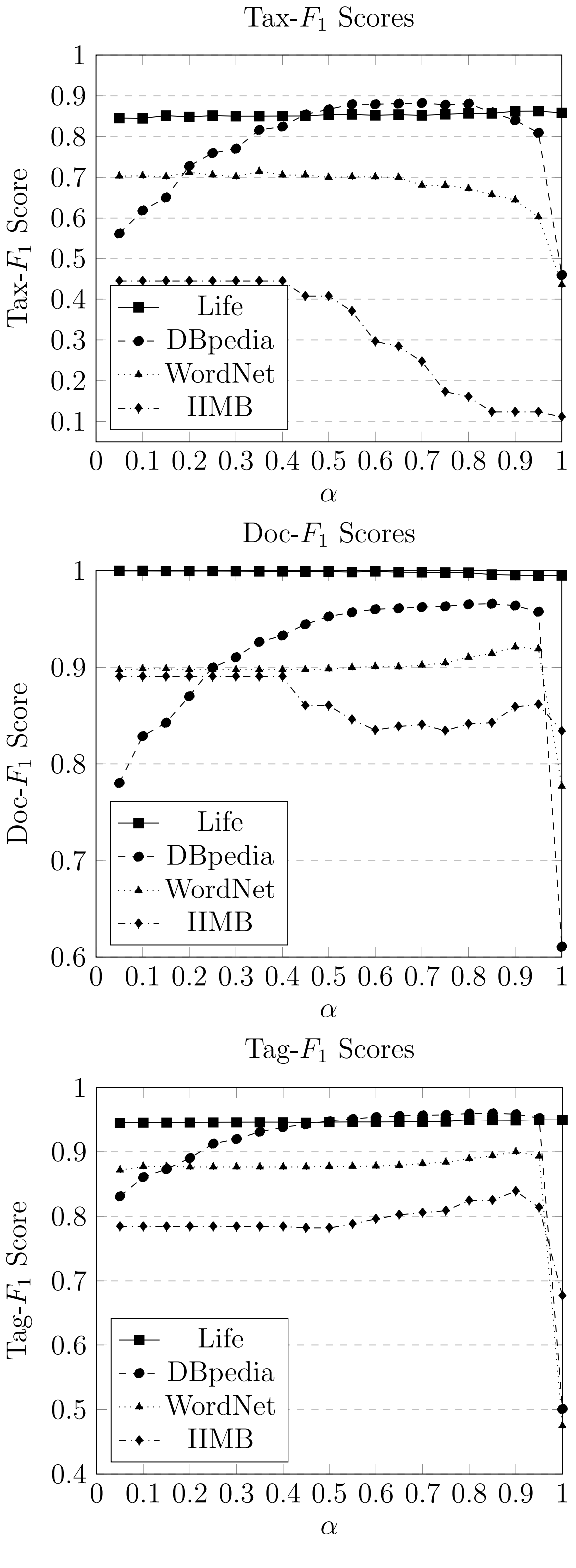


Figure 3.1: Sensitivity to α as per Tax-F_1, Doc-F_1, and Tag-F_1 on the Life, DBpeida, WordNet, and IIMB datasets.

DBpedia dataset shows improvement with increasing α values until a peak is reached and Tax-F_1 declines. The increase in induced taxonomy quality with increasing α values is consistent with the assumption that taking into account a potential parent's path is advantageous when selecting a parent. The decline in Tax-F_1 after $\alpha = 0.8$ can be explained by distant ancestor tags having too strong an influence in assigning parent tags to children. One possible explanation for better Tax-F_1 scores of lower α values on WordNet and IIMB is our method's overall lower Tax-F_1 scores on these datasets. Errors in the induced taxonomy propagate downwards and their effect increases with the value of α. Thus, in a taxonomy with many errors, it is advantageous to place a relatively higher value on the similarity between the direct parent tag and its child, as is done with lower α values.

Broadly, the measures of performance of hierarchical clustering, Doc-F_1 and Tag-F_1, follow a similar pattern to Tax-F_1. The main reason for this is that hierarchical clustering is initialized by taxonomy induction. As such, errors present in the taxonomy propagate to the clustering procedure. We note two exceptions to this: Doc-F_1 and Tag-F_1 scores on the WordNet dataset; and Tag-F_1 scores on the IIMB dataset. The former exception does not show decline in clustering performance at $\alpha > 0.7$, despite initialization by lower quality taxonomies. We hypothesize that this is due to the fact that many errors in the WordNet taxonomy occur at lower levels which get pruned during hierarchical clustering and therefore do not impact Doc-F_1 and Tag-F_1 scores. The latter exception only shows a decline in Doc-F_1 scores at $\alpha > 0.4$, with Tag-F_1 scores increasing. This is because higher α values induced a deeper hierarchy which introduces more errors when higher level clusters inherit the documents of their descendants. Unlike Doc-F_1, Tag-F_1 is resistant to these errors since only the highest F_1 score is considered in the pairwise comparison between tags and clusters.

In general, it is difficult to predict the optimal α value a priori, however there are a few rules of thumb to guide this process when applying our method. When there is no prior information about a nature of the dataset or its expected class taxonomy,

Method	**Life**	**DBpedia**	**WordNet**	**IIMB**
Heymann and Garcia-Molina	–	0.7982	0.5918	0.2025
		±0.0159	±0.0114	±0.0068
Schmitz	0.8423	0.8013	0.7943	0.5211
	±0.0000	±0.0000	±0.0000	±0.0000
Paulheim and Fümkranz[2] [52, 79]	–	0.1410	–	–
Ristoski et al.[2] [52]	–	0.5210	–	–
Völker and Niepert[2] [48]	–	0.9950	–	–
Our method	0.8625	0.8824	0.7144	0.4444
	±0.0040	±0.0052	±0.0069	±0.0000

Table 3.2: Doc-F_1 (mean ± standard deviation) on the Life, DBpedia, WordNet, and IIMB datasets.

we suggest using α values around 0.5 as these values perform well (although not optimally) in our experiments. Datasets which are complex, or have low co-occurence rates between ancestor and descendent tags will favour lower α values as these ensure errors will propagate less through the taxonomy. On the other hand, well structured datasets will be less affected by varying α values.

3.5.3 Taxonomy Induction

In our experiments, we applied our proposed method to each of the aforementioned datasets at the α values determined optimal in the previous subsection. Each dataset was applied five times to account for the stochasticity in sorting tags of equal generality. The results of our method as well as those of the comparison methods are summarized in Table 3.2. We implemented Heymann and Garcia-Molina, and Schmitz methods to the best of our understanding and performed hyperparameter exploration for their respective hyperparameters on each dataset. After obtaining the optimal hyperparameters, we ran the methods five times on each dataset and collected the results. We note that Heymann and Garcia-Molina was not able to terminate sufficiently fast enough for us to obtain results on the Life dataset. In the table we also included the results reported in previous work applied on the DBpedia dataset. Although the DBpedia dataset was derived similarly to our own, it is not identical and thus conclusions in comparing these methods to our proposed method should be drawn cautiously. We indicate these entries in the table with a footnote[2].

In general, all tag hierarchy methods achieve encouraging results and our method outperforms the others on two of the four datasets. We notice that since Tax-F_1 measures the balance between precision and recall values, this suggests that our method is both capable of inducing subsumption axioms (recall) while ensuring these axioms are correct (precision). Furthermore, closer inspection of the results reveals that many of the errors can be categorized by two types, which we illustrate by using results from the DBpedia dataset. In the first, the order between parent and child tags are reversed as in the induced {`dbo:Guitarist` → `dbo:Instrumentalist`} when the correct order is {`dbo:Instrumentalist` → `dbo:Guitarist`}. In the second, a tag is misplaced as the child of its sibling, for instance, the gold standard classification of educational institutions is {{`dbo:EducationalInstitution` → `dbo:University`}, {`dbo:EducationalInstitution` → `dbo:College`}} while our induced taxonomy gives the following: {{`dbo:EducationalInstitution` → `dbo:University`}, {`dbo:University` → `dbo:College`}}. Finally, our induced taxonomy includes subsumption axioms which are considered incorrect as per the gold standard but may not be to a human evaluator. An example of this is that our method induced the subsumption axiom {`dbo:SportFacility` → `dbo:Stadium`} while the gold standard considers {`dbo:Venue` → `dbo:Stadium`} to be the correct parent for `dbo:Stadium`. We provide an excerpt of our induced class taxonomies on the Life and DBpedia datasets in Figure 3.2.

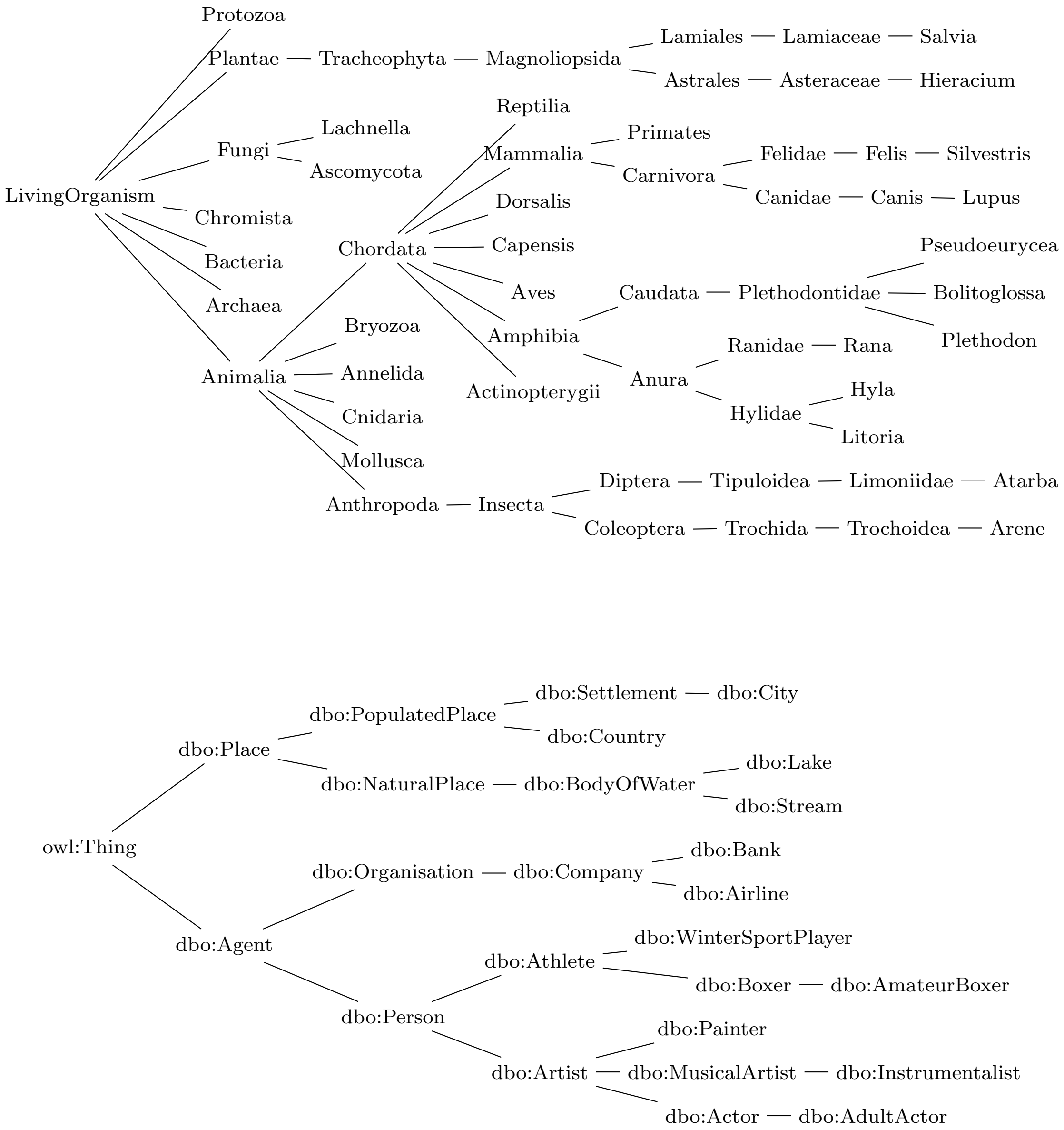


Figure 3.2: Excerpts of the induced class taxonomies for the Life (top) and DBpedia (bottom) datasets. (Read left to right.)

Dataset	**Doc-F_1**	**Tag-F_1**
Life	0.9949 ± 0.0000	0.9499 ± 0.0000
DBpedia	0.9624 ± 0.0000	0.9572 ± 0.0000
WordNet	0.8977 ± 0.0000	0.8765 ± 0.0000
IIMB	0.8903 ± 0.0000	0.7843 ± 0.0000

Table 3.3: Hierarchical clustering results (mean ± standard deviation) on the Life, DBpedia, WordNet, and IIMB datasets.

3.5.4 Hierarchical Clustering

As before, we apply our hierarchical clustering scheme on each of the four datasets at optimal α values as per the Tax-F_1 score. We repeat this process five times for each dataset and report the results in Table 3.3. We notice no variance in results despite our model's stochasticity in sorting tags. This is because the Doc-F_1 and Tag-F_1 metrics are insensitive to the ordering errors between parents and children discussed earlier. Furthermore, Doc-F_1 is higher than Tag-F_1 on all datasets. This suggests that our method is better at inducing clusters with strong inheritance properties and a high degree of consistency between cluster members and cluster annotation than it is at representing every class in the taxonomy with a cluster. Closer inspection of clustering errors shows that the majority of errors are the result of errors carried over from the taxonomy induction step. Specifically, the most common type of error is due to missing or incorrect ancestors in the paths of documents' clusters. Missing ancestors result in a false negative whereas incorrect ancestors result in a false positive, decreasing F_1 scores.

Figure 3.3 provides an excerpt of hierarchical clustering on the IIMB dataset. Recall that the induced class taxonomy showed a poor Tax-F_1 score on this dataset. Despite this, the hierarchical clustering obtained from this taxonomy scores highly on Doc-F_1 and Tag-F_1 and qualitative assessment confirm that it is well structured and

[2] The result for this method was obtained from the literature.

coherent. This highlights problems with using gold standards, namely: there may be multiple valid ways of structuring a taxonomy; and there may be a disconnect between how the data ought to be structures and how it is structured. Both of these problems are manifest in the IIMB dataset.

3.5.5 Computational Complexity Analysis

One of the most salient issues that arises when applying class taxonomy induction methods to real-world knowledge graphs is that of scalability. As mentioned previously, DBpedia, Yago, and WikiData have upwards of one billion triples each, thus for a method to operate on these datasets, it has to be computationally efficient. It is important to note, however, that in inducing a class taxonomy, it is not necessary to use all the triples available in the knowledge graph but rather to only use as many as is required to achieve an acceptable result. We discuss this idea in the following subsection.

The most computationally taxing procedure in taxonomy induction using our method is that of calculating the number of documents annotated by two tags, $\mathrm{D}_{a,b}$, which has a worst case time complexity of $\mathcal{O}(|\mathcal{D}||\mathcal{V}|^2)$, where $|\mathcal{D}|$ and $|\mathcal{V}|$ are the number of documents and tags, respectively. It is important to note, however, that the worst case only occurs when all documents are annotated by all tags. In this scenario, every subject in a knowledge graph is of every class type in the ontology. The average computation complexity of our algorithm is $\mathcal{O}(|\mathcal{D}|\overline{|\mathcal{A}|}^2)$ where $\overline{|\mathcal{A}|}$ is the average number of tags that annotate a document. In our experiments our method was faster to terminate than both the Heymann and Garcia-Molina and Schmitz methods on all four datasets.

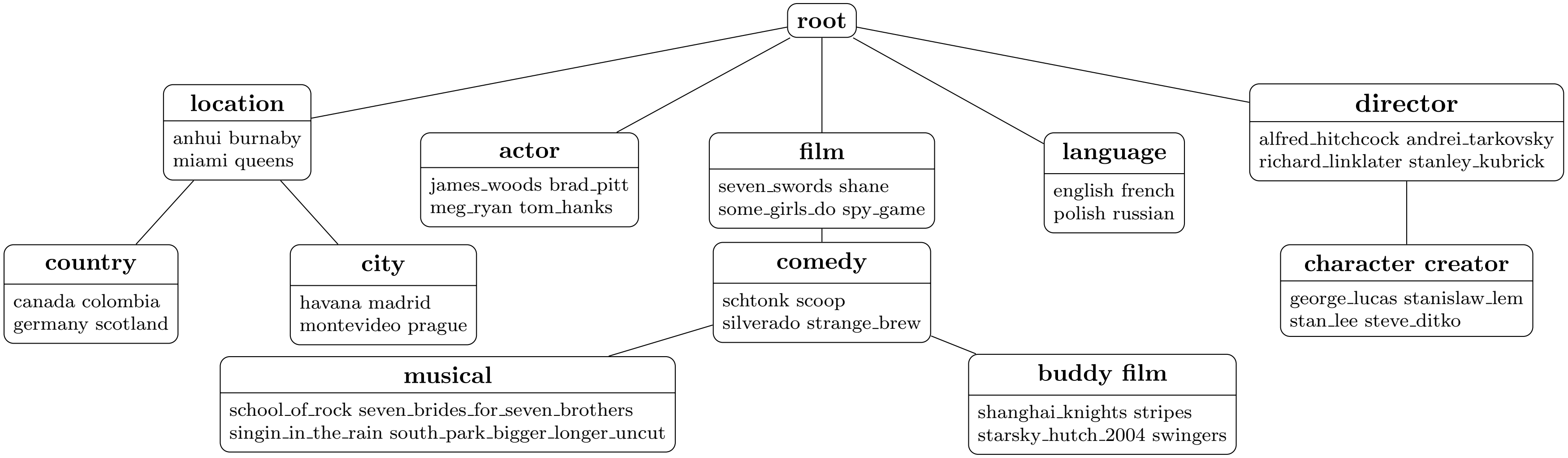


Figure 3.3: Excerpt of the cluster hierarchy induced on the IIMB dataset. Node top indicates cluster's tag; bottom indicates cluster's constituent subjects.

# Documents	# Tags	# Triples	Optimal α	Time (sec)	F_1
100000	428	422860	0.65	1.6311	0.8810
90000	427	379444	0.65	1.5131	0.8808
80000	425	336084	0.45	1.3340	0.8826
70000	424	292791	0.55	1.1248	0.8847
60000	423	249383	0.70	0.9767	0.8783
50000	418	205793	0.70	0.8556	0.8824
40000	414	164470	0.70	0.6545	0.8783
30000	408	123408	0.55	0.5564	0.8716
20000	392	82381	0.65	0.3652	0.8791
10000	365	41081	0.65	0.2001	0.8425
5000	326	20481	0.70	0.1161	0.8354
2500	284	10330	0.60	0.0670	0.8372
1000	211	4097	0.35	0.0280	0.7632

Table 3.4: Summary of our method's results on DBpedia datasets at various document counts, $|\mathcal{D}|$.

Hierarchical clustering of documents involves a pairwise comparison between the documents and classes in the taxonomy. Thus, the time complexity of performing hierarchical clustering given the induced class taxonomy is $\mathcal{O}(|\mathcal{D}||\mathcal{V}|)$, allowing for fast execution even on large datasets. We note that the two metrics used for evaluating the clustering are relatively costly. Specifically, Doc-F_1 has a time complexity of $\mathcal{O}(|\mathcal{V}|)$ and Tag-F_1 has a time complexity of $\mathcal{O}(|\mathcal{V}^2|)$.

3.5.6 Effect of Dataset Size on Induced Taxonomy

As mentioned previously, although a method's scalability to large knowledge graphs is important in the context of the Semantic Web, it's not the case that larger datasets will produce better taxonomies. To demonstrate this, we applied our method to DBpedia datasets at differing document counts. Each dataset was derived the same way as described in the Datasets subsection, such that all of the smaller DBpedia datasets are strict subsets of the larger ones. A summary of the results is displayed in Table 3.4. We note that runtime measures the execution of our method without including time for input and output. We notice that although larger datasets obtain higher Tax-F_1 scores, the incremental increase in Tax-F_1 diminishes, and the scores plateau after 20,000 documents. However, relying on Tax-F_1 score as the sole comparison metric may be misguiding since it is calculated on the tags which exist in the dataset. Thus since there are 211 unique tags in the DBpedia 1,000 dataset and 428 unique tags in the DBpedia 100,000 dataset, the induced taxonomy of the latter will be over twice as large as the former.

3.6 Conclusions

In this chapter, we described the problem of inducing class hierarchies from knowledge graphs and its significance to the Semantic Web community. In our contribution to this research area, we proposed an approach to the problem by marrying the fields of class taxonomy induction from knowledge graphs with tag hierarchy induction from documents and tags. To this end, we reshaped the knowledge graph to a tuple structure and applied two existing tag hierarchy induction methods to show the viability of such an approach. Furthermore, we proposed a novel method for inducing class taxonomies that relies solely on class frequencies and co-occurrences and can thus be applied on knowledge graphs irrespective of their content. We demonstrated our method's ability to induce class hierarchies by applying it on four real-world datasets and evaluating it against their respective gold standard taxonomies. Finally, we showed how a class taxonomy may be used as the foundation for a simple hierarchical clustering scheme. This scheme was applied to the aforementioned datasets and evaluated on two metrics. Results demonstrate that our approach is capable of inducing high quality class taxonomies as well as hierarchical clusterings and can be reliable applied to large-scale knowledge graphs.

Chapter 4

Probabilistic Coarsening for Knowledge Graph Embeddings

This section summarizes the paper *Probabilistic Coarsening for Knowledge Graph Embeddings* which is intended for publication at an artificial intelligence or semantic web conference.

4.1 Introduction

As mentioned in Chapter 2, embeddings are among the most common operations on knowledge graphs. Despite this, relatively less work has been done on methods of preprocessing a knowledge graph prior to embedding to yield better results. This provides the motivation for this work which investigates coarsening as a tool for knowledge graph embedding. Specifically, a simple embedding meta-strategy that can be applied to any arbitrary embedding method is proposed. The strategy first reduces an input knowledge graph – henceforth referred to as a base graph – to a coarsened graph along with a mapping between the entities in each knowledge graph. Coarse embeddings are then learned on the coarse graph and mapped back down as base embeddings. They may then be fine tuned on the base graph to reintroduce information that was lost in the coarsening procedure. Figure 4.1 outlines the flow of this approach.

The rationale for coarsening is multifactorial, as pointed out in Chen et al. [80]

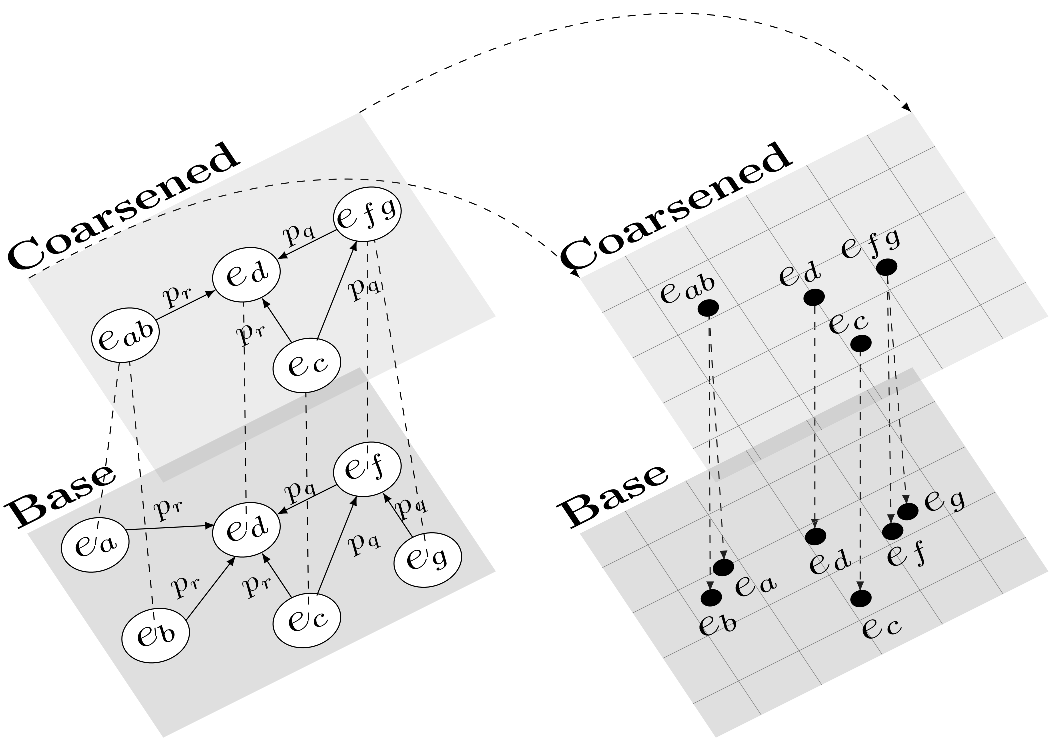


Figure 4.1: Toy example demonstrating the proposed embedding strategy. The logical flow is guided by dashed line arrows, starting in the bottom left corner and proceeding clockwise.

and Liang et al. [81]. Specifically:

- Coarsening reduces knowledge graph size whilst preserving global structure, potentially revealing higher-order features.
- Training schemes which rely on stochastic gradient descent may learn embeddings that fall in local minima. Initializations learned on the coarse graph may be more resistant to this problem.
- Structurally equivalent entities are embedded jointly in coarse graphs, reducing training complexity.

We evaluate our strategy on the entity classification task using four real-world datasets and perform a pairwise comparison against common embedding methods. The results indicate that embedding on a coarse graph produces faster and in many cases higher quality embeddings.

In summary, the contributions of the work in this chapter are as follows. To the best of our knowledge, we are the first to use coarsening as an explicit preprocessing strategy for generating knowledge graph embeddings. To this end, we propose a novel probabilistic coarsening procedure that reduces knowledge graph size while preserving its global structure. The results of our empirical evaluation allow us to conclude that coarsening is a recommended strategy regardless of the underlying embedding method being used.

4.2 Related Work

Research in graph representation, including graph embeddings, has a long history rooted in mathematics with early methods discussed in a survey by Archdeacon [82]. More recently, deep learning has been leveraged for graph embeddings to achieve state-of-the-art results. For instance, DeepWalk [83], Large-scale Information Network Embedding [84], and node2vec [85] sample random walks on a graph and treat them as input words to the skip-gram language model [86]. The intuition behind this approach is that nodes which are sampled in the same random walks are more similar semantically and should have similar embeddings. Another class of deep approaches, Graph Convolution Networks (GCN) [87, 88], utilize the convolution operator to learn neighbourhood information for graph entities. Extensions and derivatives of GCNs are ample; for a comprehensive discussion of these methods we refer readers to Wu et al. [89]. Autoencoders use neural networks to reduce an input to a latent embedding before reconstructing the embedding back to the original input. This approach has shown success in generating graph embeddings in works by Bellini et al. [90] and Simonovsky and Komodakis [91].

Meta-strategies which preprocess graphs into hierarchies before applying embedding methods have been shown to produce higher quality embeddings at reduced training complexity. The first of these proposed, HARP [80], sequentially reduces the input graph into a hierarchy of progressively coarser graphs. These coarse graphs are then embedded starting with the coarsest graph such that their embeddings serve as the initialization of the graph directly below it in the hierarchy. This idea was extended in Multi-Level Embedding [81] and GOSH [92], both of which modify the coarsening procedure.

Knowledge graphs present structural traits which render the aforementioned methods ill-suited for their embedding. Specifically, knowledge graphs are directed and labelled in their relations between entities. In light of this, much research has been devoted to developing methods which account for these complexities. For instance, RDF2Vec [93] uses breadth-first graph walks on the skip-gram model to generate embeddings. GCNs have been extended to knowledge graphs with the Relational Graph Convolution Network (R-GCN) [94] which aggregates predicate specific convolutions of the original model. ConvE [95] also leverages the convolution operator in a neural framework by stacking embeddings as a matrix and convolving them in two dimensions. Translation based methods such as TransE [96] apply the intuition that subject embeddings should be near object embeddings when translated by valid corresponding predicates. Factorization models such as the aforementioned RESCAL [50] and DistMult [97] learn embeddings by factorizing the knowledge graph adjacency tensor into the product of entity embeddings and relation specific translation matrices. Deep reinforcement learning has also shown promise in this domain with MINERVA [98] which learns knowledge graph paths to find the correct entity in incomplete triples. A recent and comprehensive discussion of knowledge graphs embedding methods can be found in Ji et al. [99].

4.3 Problem Description

Recall from Chapter 2 that the task of knowledge graph embedding is to find a function, f, which maps each entity to the embedding space $f : \mathcal{E} \mapsto \mathbb{R}^{|\mathcal{E}| \times d}$ where d

is the dimensionality of the embedding space such that $d << |\mathcal{E}||\mathcal{R}|$.

Given an arbitrary knowledge graph embedding function, f, the task is to find a mapping which reduces a base graph, $\mathcal{G}$, to a coarse graph, $\mathcal{G}'$, such that $|\mathcal{G}'| < |\mathcal{G}|$. This non-injective surjective mapping, denoted $\Psi : \mathcal{E} \mapsto \mathcal{E}'$ where $\mathcal{E}'$ is the set of entities in the coarse graph, should preserve the global structure of the base graph and be computationally efficient so as to not bottleneck the proposed strategy. Having found Ψ, the proposed strategy can trivially be carried out as outlined earlier.

The tuple structure introduced in Chapter 3 is extended such that each subject is now identified by both incoming and outgoing relations. Thus, a tag t, is defined as a predicate-entity pair that describes another entity, $t := \langle r_p, e_j \rangle \mid \langle e_i, r_p \rangle$. Order between a tag's predicate and entity corresponds to whether the tag is incoming or outgoing. Thus, each triple in $\mathcal{G}$ corresponds to two entity-tag mappings. As before, these mappings are expressed as sets such that each entity e_i has a corresponding set of tags which annotate it, denoted $\mathcal{A}_i$.

4.4 Proposed Strategy

Our proposed strategy embeds a base graph via an intermediary coarse graph. In this process, the coarse graph is first generated from the base graph before being embedded. Coarse embeddings are then mapped back down to the base graph and fine tuned. Our strategy may be divided into the following three steps:

1. **Probabilistic graph coarsening** reduces the base graph to a smaller, coarsened graph and returns an entity mapping between the two graphs.

2. **Coarse graph embedding** applies a predetermined embedding method on the coarse graph to obtain coarse embeddings.

3. **Reverse mapping and fine tuning** maps coarse embeddings back down to the base graph to obtain base embeddings. Base embeddings may be fine tuned on the base graph.

The remainder of this section describes each of the these steps in detail. The intuition for this process is described visually in Figure 4.1 while its sequence is outlined in Algorithm 3.

4.4.1 Probabilistic Graph Coarsening

In this step, the base graph is reduced to create a coarse graph, denoted $\mathcal{G}'$, such that $|\mathcal{G}'| < |\mathcal{G}|$. This procedure involves collapsing structurally similar entities in $\mathcal{G}$ to one entity cluster in $\mathcal{G}'$. Relations in $\mathcal{G}$ are extended to $\mathcal{G}'$ such that a cluster's relations are the union of its constituent entities' relations. Collapsing entities is divided into two stages, designed to preserve the first order and second order proximities of the base graph [84]. This allows the base graph to be reduced of structural redundancies, making it more computationally manageable and potentially revealing its global and most salient features. The mapping between base entities and coarse entities is represented by $\Psi : \mathcal{E} \mapsto \mathcal{E}'$. Coarsening is demonstrated visually on the left half of Figure 4.1.

Collapsing First Order Neighbours

Preserving first order proximity refers to the notion that entities should be embedded proximally to their first order (i.e. one-hop) neighbours. By collapsing entities with their first order neighbours, proximity is ensured as collapsed entities share identical embeddings. In undirected, single predicate graphs, edge collapsing [80, 100, 101] finds the largest subset of edges such that no two edges are incident to the same vertex. Vertices incident to each edge in this set are then collapsed, yielding a graph coarsened to preserve first order proximity. Edge collapsing may be applied to knowledge graphs by assuming undirected graph relations. This approach proves too liberal in its coarsening, however, since the cost of coarsening is increased in knowledge graphs due to loss of predicate information. Graphs without labelled edges do not suffer from this issue. In response, we restrict edge collapsing to entities whose collapsing incurs no loss of predicate information other than predicates which are incident to both entities. Formally, entity e_a is collapsed with e_b if:

$$\{t \in \mathcal{A}_a : e_b \notin t\} \subseteq \{t \in \mathcal{A}_b : e_a \notin t\} \tag{4.1}$$

First order entity collapsing is demonstrated in the lower left quadrant of Figure 4.1 where entity e_g is collapsed with its neighbour e_f to form entity cluster e_{fg} in the coarse graph. We note that entities e_a and e_b are also valid candidates for first order collapsing with e_d. This demonstrates the necessity of initially performing second order neighbour collapsing since e_a and e_b are structurally equivalent and thus more similar to one another than to e_d. As such, first order collapsing is performed after second order collapsing as seen in Algorithm 3 where it is captured in lines 11 to 18.

Collapsing Second Order Neighbours

Second order (i.e. two-hop) neighbours are two entities which share a first order neighbour. The rationale for preserving second order proximity is discussed in [84] and predicated on the intuition that entities which have many common first order neighbours tend to exhibit similar structural and semantic properties. As such, they should be proximal to one another in the embedding space. In Figure 4.1, we see that second order neighbours e_a and e_b have identical tag sets (i.e. $\mathcal{A}_a = \mathcal{A}_b$) and are thus structurally equivalent. Collapsing these entities and embedding them jointly ensures preservation of second order proximity in the embedding space while incurring no loss of information. We apply this reasoning to our coarsening procedure. Namely, if two second order neighbours exhibit a high degree of structural similarity, we collapse them in the coarse graph. We measure the similarity between a pair of second order neighbours as the Jaccard coefficient between their tag sets:

$$\text{Sim}(e_a, e_c) = \frac{|\mathcal{A}_a \cap \mathcal{A}_c|}{|\mathcal{A}_a \cup \mathcal{A}_c|} \tag{4.2}$$

Where $\text{Sim}(e_a, e_c)$ is the similarity between e_a and e_c such that $0 \leq \text{Sim}(e_a, e_v) \leq 1$. Second order neighbours are collapsed if their similarity is greater than or equal to a threshold, α, which is chosen such that $0 < \alpha \leq 1$. The value of α dictates the coarseness of the graph with lower α values resulting in smaller, coarser graphs. This process may be seen as a relaxation of Structural Equivalence Matching (SEM) proposed in Liang et al. [81] which collapses second order neighbours only if they are structurally equivalent. In other words, SEM is analogous to our method at $\alpha = 1$; collapsing entities e_a and e_c when $\mathcal{A}_a = \mathcal{A}_c$. Second order collapsing is summarized in lines 2 to 10 of Algorithm 3.

Neighbour Sampling

The pairwise comparison between entities and their first and second order neighbourhoods has a worst case time complexity of $O(|\mathcal{E}|^2)$ and is thus computationally infeasible for large scale knowledge graphs. To overcome this, we propose a scheme for sampling neighbours using constrained random walks. To obtain a first order neighbour for entity e_a, we first sample a tag from its tag set:

$$t_1 \sim \text{Uniform}(\mathcal{A}_a) \tag{4.3}$$

Where t_1 represents one hop in a random walk on the base graph. The first order neighbour e_b is then extracted from t_1:

$$e_b = t_1 \cap \mathcal{E} \tag{4.4}$$

Note that since $t_1 \cap \mathcal{E}$ is a singleton set, we can abuse the $=$ symbol such that $e_b = \{e_b\}$. The predicate $p_r = t_1 \cap \mathcal{P}$ is used as a constraint in sampling a second order neighbour for e_a. Specifically, given e_a and its first order neighbour e_b on predicate p_r, we only sample second order neighbours which are incident to e_b on p_r:

$$t_2 \sim \text{Uniform}(\{t \in \mathcal{A}_b : p_r \in t \wedge e_a \notin t\}) \tag{4.5}$$

The second order neighbour e_c is extracted from t_2 analogously to e_b:

$$e_c = t_2 \cap \mathcal{E} \tag{4.6}$$

Sampled neighbours are collapsed if they meet the aforementioned requirements, resulting in a stochastically derived coarse graph.

We sample $\eta \geq 1$ neighbours for each entity resulting in a $O(|\mathcal{E}|\eta)$ time complexity for our sampling scheme. As a hyperparameter of coarsening, η is chosen a priori allowing for flexibility to account for knowledge graph size. In practice we see that even small values of η yield encouraging results. The intuition behind this may be summarized as follows:

- Entities which meet the criteria for collapsing are likely to have smaller neighbourhoods.

- Entities that belong to smaller neighbourhoods have a higher chance of getting sampled as candidates for collapsing.

This allows our strategy to be performed with little added computational overhead. We note that reading a dataset has a time complexity of $O(|\mathcal{G}|)$ which may itself be more computationally taxing than coarsening on dense knowledge graphs.

4.4.2 Coarse Graph Embedding

Having coarsened the base graph, coarse embeddings are obtained by applying an arbitrary embedding method on the coarse graph. Since the coarse graph has all the properties of its base counterpart, no additional changes to the embedding method are necessary, merely a different input. Due to there being fewer entities in the coarse graph than its base counterpart, coarse embeddings may require fewer training steps resulting in faster training times. We use the notation $f(\mathcal{G}')$ to denote the embedding of coarse graph $\mathcal{G}'$ to yield coarse embeddings $\mathbf{E}'$:

$$\mathbf{E}' = f(\mathcal{G}') \tag{4.7}$$

Line 19 in Algorithm 3 places this step in the context of our whole strategy.

Algorithm 3 Coarse knowledge graph embeddings

```
Input: base graph $\mathcal{G}$; collapsing threshold $\alpha$; random walk count $\eta$
Output: base embeddings $\mathbf{E}$
Initialize $\mathcal{G}'$ and $\Psi$
for all $e_a \in \mathcal{E}$ do
    for iteration in 1,2,...,$\eta$ do
        Obtain second order neighbour $e_c$ using (4.5) and (4.6)
        Calculate Sim($e_a$, $e_c$) using (4.2)
        if Sim($e_a$, $e_c$) $\geq \alpha$ and $e_c \notin \Psi$ then
            Collapse $e_a$ with $e_c$; Update $\mathcal{G}'$ and $\Psi$
        end if
    end for
end for
for all $e_a \in \mathcal{E}$ do
    for iteration in 1,2,...,$\eta$ do
        Obtain first order neighbour $e_b$ using (4.3) and (4.4)
        if (4.1) holds for $e_a$ and $e_a \notin \Psi$ then
            Collapse $e_a$ into $e_b$; Update $\mathcal{G}'$ and $\Psi$
        end if
    end for
end for
Obtain coarse embeddings $\mathbf{E}'$ using (4.7)
for all $e_a \in \mathcal{E}$ do
    Reverse map base embedding $\mathbf{E}[e_a]$ using (4.8)
end for
Fine tune base embeddings $\mathbf{E}$ using (4.9)
```

4.4.3 Reverse Mapping and Fine Tuning

Coarse embeddings are extended down as base embeddings $\mathbf{E}$ by reversing the mapping obtained in the coarsening step:

$$\mathbf{E}[e_a] = \mathbf{E}'[\Psi(e_a)] \tag{4.8}$$

Where $\mathbf{E}[e_a]$ indexes the base embedding for e_a. A consequence of reverse mapping is that entities which were coarsened together share identical embeddings. In applications which rely on the distinction between these entities, this property is not desired. As such, base embedding may be fine tuned by embedding $\mathbf{E}$ with respect to the base graph using $\mathbf{E}'$ as initialization. This ensures that structural information which was lost in the coarsening process is reintroduced to base embeddings and collapsed entities become delineated. Furthermore, the training process may be less likely to get stuck in local minima due to its global initializations. We use the following notation to capture fine tuning $\mathbf{E}$ using $\mathbf{E}'$ as initialization:

$$\mathbf{E} = f(\mathcal{G}|\mathbf{E}') \tag{4.9}$$

Reverse mapping and fine tuning are described in lines 20 to 23 of Algorithm 3 as the final steps in our strategy.

4.5 Evaluation

We evaluate our strategy on the entity classification task as performed in Schlichtkrull et al. [94] and Ristoski and Paulheim [93] which involves embedding a knowledge graph and using the embeddings to infer entity labels. Our strategy is compared pairwise against the baseline methods used in the embedding step. This allows us to measure whether our coarsening strategy is justified in comparison to using the baseline methods conventionally. We use three baseline methods to evaluate our strategy: RDF2Vec, R-GCN, and TransE. These methods were selected to capture the diversity of approaches to knowledge graph embedding. We use the notation C(x) to refer to our strategy applied to baseline embedding method x.

4.5.1 Datasets

We use four canonical datasets from Schlichtkrull et al. in our evaluation: MUTAG, AIFB, BGS, and AM. Each dataset consists of a knowledge graph in Resource Description Framework format and labels for a subset of its entities. We mirror Schlichtkrull et al. in removing knowledge graph relations which are on predicates that correlate strongly with its labels. Statistics for each dataset are provided in Table 4.1. What follows is a brief description of each dataset.

- **MUTAG** depicts the properties and interactions of molecules which may or may not be carcinogenic. We remove the labeling predicate `isMutagenic` from

Dataset	**MUTAG**	**AIFB**	**BGS**	**AM**
Triples	74227	29043	916199	5988321
Entities	23644	8285	333845	1666764
Predicates	23	45	103	133
Labeled	340	176	146	1000
Classes	2	4	2	11

Table 4.1: Summary of datasets used in the evaluation.

the dataset.

- **AIFB** reports the work done at the AIFB research group and labels its members by affiliation. We remove predicates `employs` and `affiliation`.
- **BGS** captures geological data from the island of Great Britain and is used to predict the lithogenicity of rocks. As such, we remove the `hasLithogenesis` predicate.
- **AM** describes and categorizes artifacts in the Amsterdam Museum. We remove the `materials` predicate as it correlates with artifact labels.

4.5.2 Procedure

Embeddings were learned using each of the baseline embedding methods on the base graph and on the coarse graph as per our strategy. To assess the quality of these embeddings, entity classification was performed by training a support vector machine on 80% on labeled entities and testing on the remaining 20% using splits provided in Ristoski et al. We use the accuracy of classification on the testing entities as the metric of our of strategy's performance. To account for stochasticity in this process, embeddings were learned and evaluated ten times for each dataset.

To obtain optimal results, we set aside 20% of the training entities as valida-

Method	**MUTAG**	**AIFB**	**BGS**	**AM**
RDF2Vec	0.7500	0.9111	0.7828	0.8758
	±0.0392	±0.0117	±0.0327	±0.0143
C(RDF2Vec)	**0.7956**	**0.9167**	**0.8828**	**0.8778**
	±0.0340	**±0.0000**	**±0.0178**	**±0.0211**
Change	**6.1%***	0.6%	**12.8%***	0.2%
R-GCN	**0.7397**	0.9528	0.8345	**0.8833**
	±0.0286	±0.0264	±0.0424	**±0.0197**
C(R-GCN)	0.7294	**0.9694**	**0.8690**	0.8828
	±0.0242	**±0.0088**	**±0.0317**	±0.0138
Change	-1.4%	**1.7%***	**4.1%***	-0.1%
TransE	0.7397	0.8722	0.6793	0.4207
	±0.0422	±0.0397	±0.0371	±0.0143
C(TransE)	**0.7412**	**0.9056**	**0.7759**	**0.4955**
	±0.0368	**±0.0299**	**±0.0335**	**±0.0179**
Change	0.3%	**3.8%***	**14.2%***	**17.7%***

Table 4.2: Results of pairwise comparison between our strategy and baseline embedding methods as measured by accuracy (mean ± standard deviation) obtained on testing entities for each dataset. Asterisk (*) indicates superior performance as per Student's t-test at 0.05 level of significance. Underline indicates top performance on dataset, regardless of baseline method.

tion for hyperparameter selection and to prevent overfitting. In magnanimity, we used embedding hyperparameters which were selected on validation results obtained on the base graphs. For coarsening hyperparameters, we performed exploration on $\alpha \in \{0.25, 0.5, 0.75, 1\}$ and used $\eta = 10$ in all of our experiments. The optimal hyperparameters for each dataset and baseline method may be found with our published code.

Dataset	**MUTAG**	**AIFB**	**BGS**	**AM**
Triples	52179	20134	501722	4080981
Change	-29.7%	-30.7%	-45.2%	-31.8%
Entities	16115	2801	78335	944759
Change	-31.8%	-66.2%	-76.5%	-43.3%
Predicates	23	43	97	129
Change	0%	-4.4%	-5.8%	-3.0%

Table 4.3: Percent reduction in coarse graphs relative to base graphs at $\alpha = 0.5$ and $\eta = 10$.

4.5.3 Results

The results of entity classification are summarized in Table 4.2. Our strategy improved on the baseline in ten of the twelve experiments, seven of which were statistically significant. Furthermore, we were able to achieve state-of-the-art performance on three of the four datasets, albeit using different baseline methods. Our strategy appears to perform worse on R-GCN relative to the other baselines. The reason for this may be that coarsening produces graphs with a larger proportion of highly connected hub entities, which is a structural weakness of R-GCN as pointed out by its authors. Finally, we see that datasets with knowledge graphs that have a higher degree of reduction at $\alpha = 0.5$ and $\eta = 10$ perform better. This is because not all knowledge graphs are equally suitable candidates for coarsening. Namely, knowledge graphs which exhibit a high degree of structural equivalency between entities loose less information in the coarsening process. We see this in the AIFB and BGS datasets where more than half of their entities get collapsed in the coarsening step as shown in Table 4.3.

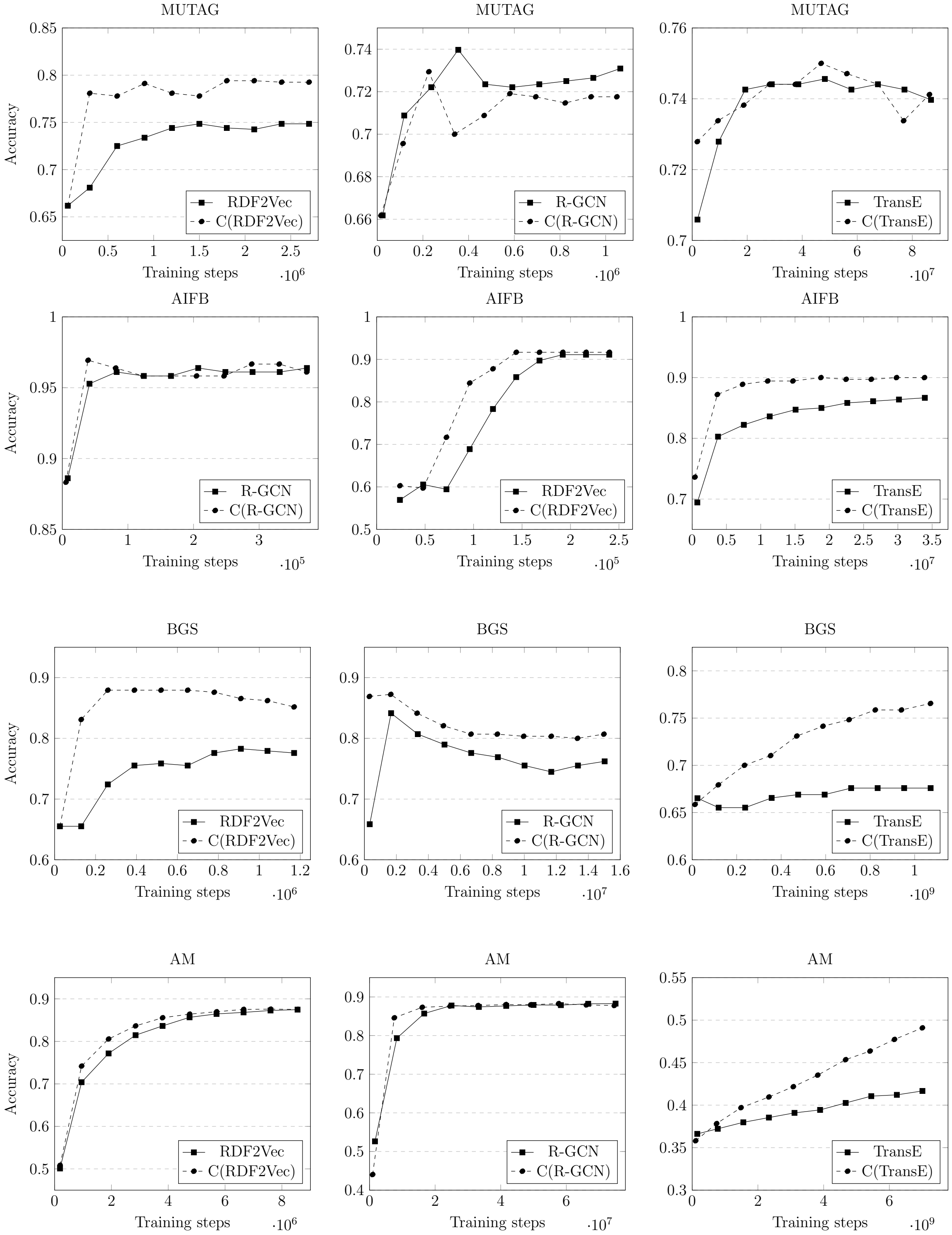


Figure 4.2: Pairwise comparison between baseline method and our strategy demonstrating performance (accuracy) as a function of the number of training steps performed for each dataset.

Figure 4.2 plots the performance of our strategy compared to baselines when increasing the number of training steps performed. Due to computational constraints, we trained the embeddings up to fifty epochs. It is possible that given enough training, baseline methods could catch up in performance to our coarsening strategy as can be seen on the AM dataset using TransE. This, however, still demonstrates that coarse graphs produce quality embeddings faster than embedding on base graphs. This is further confirmed by RDF2Vec on AIFB and AM, and R-GCN on MUTAG which show similar trendlines with the coarsened counterpart requiring fewer training steps. This suggests that our strategy is faster to train than its baseline counterparts.

4.6 Conclusions

In this chapter, we introduced a simple meta-strategy for embedding knowledge graphs that relies on coarsening as a preprocessing step to obtain a reduced knowledge graph prior to embedding. To this end, we adapted existing graph coarsening concepts to knowledge graphs and introduced a novel entity collapsing and neighbour sampling scheme. Our evaluation demonstrates that such an approach results in faster and oftentimes more accurate knowledge graph embeddings. Coupled with the fact that our strategy incurs little overhead costs, we conclude that graph coarsening is a recommended preprocessing step before applying any existing knowledge graph embedding method.

Chapter 5

Neural Blockmodelling for Knowledge Graphs

This chapter is a modified extension of the paper *Neural Blockmodelling for Multilayer Networks* published in the 2021 International Joint Conference on Neural Networks [102]. The purpose of the work presented in this chapter is to serve to demonstrate the viability of using blockmodels in the context of knowledge graphs.

5.1 Introduction

In this chapter we focus on the fusion of two common approaches to knowledge graph modelling: blockmodelling and embeddings. As mentioned in Chapter 2, blockmodels impose prior distributions on the structure of the graph. This allows for flexibility in modelling since different structural priors can shape the graph to meet the demands of a specific task. This approach is limited, however, in that it is difficult to jointly model entities and learn deep entity representations. Furthermore, they usually rely on complicated posterior inference schemes to learn the model such that minor structural changes may not be easy to solve. Embedding models, on the other hand, are model capable of capturing deep entity representations and thus achieve state-of-the-art performance on different measures of knowledge graph modelling. However, due to them only performing entity embeddings, they have to be used in conjunction with other classification or clustering methods since they do not perform these tasks explicitly.

In this regard, we propose a neural model in an attempt to overcome the shortcomings and leverage the strengths of the aforementioned approaches. Our model allows for simultaneously learning entity embeddings while assigning each entity with a community membership distribution and modelling community relations. To handle knowledge graphs, we propose an embedding for each predicate in the graph for mixing with community relations when generating the graph. We are able to achieve this by organizing each of these components in a neural network architecture. To the best of our knowledge, this fusion of blockmodelling and embedding approaches is the first of its kind in a knowledge graph context. We evaluate our model by performing three tasks which aim to demonstrate the different capabilities of our model: link prediction, entity classification, and community detection.

5.2 Related Work

We divide our discussion of related work into two classes of approaches to graph modelling: blockmodels and embedding models. Both of these approaches provide inspiration for our model and may be viewed as a separate components of it which we combine in a neural network framework.

5.2.1 Stochastic Blockmodels

The seminal work in this area is the Stochastic Blockmodel [103] which partitions entities into a fixed number of communities and models the relations between them as those of their communities. Community relations are modelled via a community relations matrix which assigns a degree to all pairwise relations between the communities in the model. This idea was extended to the infinite case allowing for an a priori unspecified number of communities via the Chinese restaurant process [20] in the Infinite Relational Model (IRM) [104]. A variant which relaxes the notion of community membership to allow for entities belonging to multiple communities is the aptly named Mixed Membership Stochastic Blockmodel (MMSB) [15]. By allowing for mixed membership, the model is better able to capture entities whose belonging to a community is not crisp. For instance, the belonging of tomatoes to the community of fruits is not perfect since it can be considered a vegetable in certain contexts such as in cooking. This idea was generalized to the infinite case in the Dynamic Infinite Mixed Membership Stochastic Blockmodel [105]. We provide a detailed description of the MMSB in the next section. All of the aforementioned models, however, operate on graphs wherein entities are related to one another through the same type of predicate, making them unsuitable for application to knowledge graphs without modification.

The underlying structure of a knowledge graph is that of a multilayer graph wherein entities interact with one another through different types of relations, represented as different types of predicates in the graph. These predicates may be thought of as separate layers of graphs which share the same entities. Multilayer graphs have also received considerable attention in stochastic blockmodelling. Perhaps the simplest approach is to aggregate the layers in the multilayer graph to a single layer before applying a conventional blockmodelling approach as was done in Berlingerio et al. [106]. A closely related approach is to model each layer in the graph independently as done in Barigozzi et al. [107] and aggregate the results afterwards. These approaches offer limited success as they don't capture the interlayer dependencies in the multilayer graph and treat each layer as equally valuable in its content during modelling, as pointed out by Paul and Chen [108]. To remedy this, the authors propose a multilayer extension of the aforementioned Stochastic Block Model, aptly named the Multi-Layer Stochastic Blockmodel, which modifies the original community relations matrix to a community relations tensor to account for graph multilayeredness. Analogously, a multilayer extension for the MMSB was proposed by De Bacco et al. [109].

Dynamic blockmodels are a subclass of multilayer blockmodels which are designed to model the changes that occur in a graph over time. These temporal changes, or time-steps, correspond to different predicates in a graph. The key difference in the approach of temporal blockmodels with respect to multilayer blockmodels is that they model graph relations based on previous time-steps whereas multilayer blockmodels model graph relations irrespective of temporality. The Dynamic Mixed Membership Stochastic Blockmodel (dMMSB) [110] extends the MMSB for this purpose by using a Markov model to capture the temporal changes in a graph. The Deep Dynamic Mixed Membership Stochastic Blockmodel (DDBM) [111] factorizes the community relations matrix to avoid the problem of missing community relations. In addition to this, it utilizes a long short-term memory cell to model the dynamic changes in a graph. The Fragmentation Coagulation Mixed Membership Stochastic Blockmodel (fcMMSB) [112] uses the discrete fragmentation coagulation process [113] to allow for modelling appearances and disappearances of communities over time. A comprehensive review of stochastic blockmodels and their applications is provided by Lee and Wilkinson [114].

5.2.2 Embedding Models

A brief overview of embedding models for knowledge graphs was provided in the discussion of related works in Chapter 4. We supplement this by drawing attention to several embedding models for multilayer graphs which are used in the comparison procedure of our evaluation.

To the best of our knowledge, the first methods for specifically generating deep embeddings on multilayer graphs are the three variants proposed in Principled Multilayer Network Embedding (PMNE) [115]. The first two variants rely on aggregation, either on the level of graph relations or embeddings. The third extends random walk sampling to account for multiple relation types and runs an optimization algorithm similar to node2vec. Multiplex Network Embedding (MNE) [116] models embeddings as the sum of a base embedding, which is shared across relation types, and relation-specific embeddings. As with the previous methods, optimization is performed by sampling random walks and applying the skip-gram algorithm. A multilayer extension of DeepWalk, M-DeepWalk [117], was proposed which first performs DeepWalk on all the relation layers together and then use a deep neural network to refine the embeddings.

5.3 Problem Description

Our model is formulated as a blockmodel and must therefore generate a knowledge graph as its output. Specifically, we want to generate the adjacency tensor of a knowledge graph, $\mathbf{G}$, as defined earlier. Recall that in this formulation, there exists a relation from entity e_i to entity e_j on predicate r_r if there is a corresponding value of 1 in the adjacency tensor. Thus, given the indices for e_i, e_j, and r_r our model will output a probability value of seeing corresponding relation in the knowledge graph.

5.4 Model Description

We introduce our model by first describing the model from which it draws inspiration, the aforementioned MMSB. Indeed, our model may be viewed as a multilayer extension of the MMSB in a neural framework. Introducing our model in this context allows us to better draw attention to the similarities and differences between the two models.

5.4.1 Mixed Membership Stochastic Blockmodel

Recall that in blockmodelling, relations between entities are modelled as those of their respective communities. In the MMSB, the relation probabilities between communities are captured in the $K \times K$ community relations matrix, $\mathbf{C}$, where K is the predefined number of communities. The probability of a relation from an entity in community t_p to an entity in community t_q is thus equal to the entry c_{pq}, indicating the value at the p^{th} row and q^{th} column of $\mathbf{C}$. Community relations are drawn from the Beta distribution:

$$c_{pq} \sim \text{Beta}(\lambda_1, \lambda_2) \tag{5.1}$$

Where λ_1 and λ_2 are parameters for the Beta distribution and thus hyperparameters of the model. Entity membership to communities is modelled by a K-dimensional community membership vector, $\mathbf{a}_i$, such that its values specify the probability distribution of entity e_i belonging to each of the K communities. It is modelled by the Dirichlet distribution, a conjugate prior of the multinomial distribution:

$$\mathbf{a}_i \sim \text{Dirichlet}(\alpha) \tag{5.2}$$

Where α is the parameter of the Dirichlet distribution. When modelling g_{ij}, these community membership vectors are used to obtain two community indicator vectors, $z_{i\rightarrow j}$ and $z_{i\leftarrow j}$. The arrows indicate the sender's (→) and receiver's (←) communities in the relation from entity e_i to entity e_j. This means that entity e_i's community indicator is dependent on the entity it is interacting with, entity e_j, as well as its role (sender or receiver) in the relation. Community indicators are drawn from the multinomial distribution parametrized by the community memberships:

$$z_{i\rightarrow j} \sim \text{Multinomial}(\mathbf{a}_i) \tag{5.3}$$

$$z_{i\leftarrow j} \sim \text{Multinomial}(\mathbf{a}_j) \tag{5.4}$$

With these blocks in place, $z_{i\rightarrow j}$ and $z_{i\leftarrow j}$ are used to index $\mathbf{C}$ for the community relation probability, which is then extended as the probability of a relation from entity e_i to entity e_j. Given the relation probability, we can then generate the graph from the Bernoulli distribution:

$$g'_{ij} \sim \text{Bernoulli}(z_{i\rightarrow j}\mathbf{C}z_{i\leftarrow j}) \tag{5.5}$$

Posterior inference is performed on the parameters using Markov chain Monte Carlo methods such as Gibbs sampling or variational inference. Figure 5.1 depicts the graphical representation of the MMSB using plate notation.

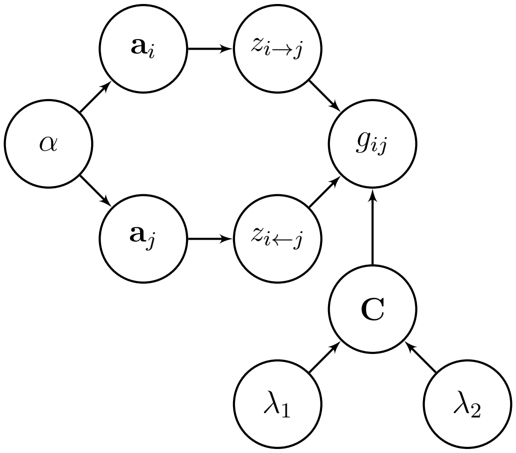


Figure 5.1: Plate diagram for the MMSB.

5.4.2 Proposed Model

Structurally, our model is an adaptation of the MMSB with five key differences:

- Entities are assigned an embedding that's used to generate community membership distributions.
- Community indicators are equivalent to community memberships; we do not draw them from community memberships.
- Relations between two communities are modelled by a community relations vector as opposed to a scalar variable.
- A predicate embedding is mixed with community relations to account for different relations in the graph.
- Our model is structured in a neural framework, eliminating the need to solve for posterior inference as parameters can be inferred with stochastic gradient descent.

To clarify the connections between the two models, we adopt the notation introduced in the MMSB when denoting similar structures in our model.

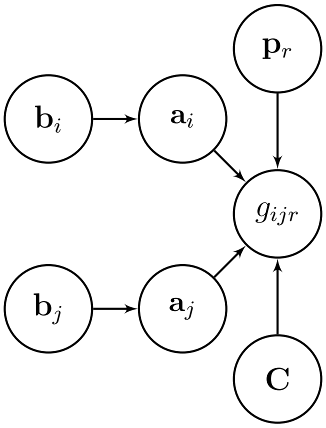


Figure 5.2: Graphical representation of MNB.

Entities are embedded in an E-dimensional embedding space via embedding vectors such that each entity e_i has a corresponding embedding $\mathbf{b}_i$. Entity embeddings are then used to generate the entity's community membership vector, $\mathbf{a}_i$, which is a K-dimensional probability distribution analogous to the one of the MMSB. The embedding dimension, E, and the number of communities, K, are hyperparameters of the model and chosen independently of one another. The embeddings are transformed into community memberships using an artificial neuron with a Softmax activation function:

$$\mathbf{a}_i = \text{Softmax}(\mathbf{R}\mathbf{b}_i + \mathbf{s}) \tag{5.6}$$

Where $\mathbf{R}$ is a $K \times E$ weight matrix and $\mathbf{s}$ is a K-dimensional bias weight. Since community memberships are not sampled, they are deterministic and independent of the other entity in the relation. Furthermore, this allows for a more relaxed understanding of membership since a entity may draw information from several communities in modelling an relation.

Community relations are modelled in a $K \times K \times A$ community relations tensor, $\mathbf{C}$. In this view, the relation from community t_p to community t_q is captured by the A-dimensional vector $\mathbf{C}_{pq}$. A is a hyperparameter of the model which determines the size of the community relation vector. The community relation vector for the entity relation from entity e_i to entity e_j is indexed by the community memberships of the

Algorithm 4 Generative Procedure for MNB

```
Input: Knowledge graph as adjacency tensor $\mathbf{G}$; entity embedding dimension $E$; number of communities $K$; predicate embedding dimension $A$
Output: Generated $\mathbf{G}'$
  Initialize all $\mathbf{R}$ and $\mathbf{s}$ randomly
  Initialize all $\mathbf{b}$ and $\mathbf{a}$ randomly
  repeat
    for $e_i, e_j \in \{1, 2, ..., |\mathcal{E}|\} \times \{1, 2, ..., |\mathcal{E}|\}$ do
      Obtain $\mathbf{a}_i$ using (5.6)
      Obtain $\mathbf{a}_j$ using (5.6)
      for $r_r \in \{1, 2, 3, ..., |\mathcal{R}|\}$ do
        Obtain $g'_{ijr}$ using (5.8)
      end for
    end for
    for mini-batch do
      Update model parameters via Adam on (5.10)
    end for
  until convergence
  for $e_i, e_j, r_r \in \{1, 2, ..., |\mathcal{E}|\} \times \{1, 2, ..., |\mathcal{E}|\} \times \{1, 2, ..., |\mathcal{R}|\}$ do
    Draw graph relation using (5.9)
  end for
```

respective entities, $\mathbf{a}_i$ and $\mathbf{a}_j$:

$$\mathbf{c}_{\mathbf{a}_i\mathbf{a}_j} = \mathbf{a}_j{}^T(\mathbf{a}_i{}^T\mathbf{C}) \tag{5.7}$$

Where the superscript T indicates the transpose operation.

To account for knowledge graph predicates, we introduce the A-dimensional predicate embedding, $\mathbf{p}_r$, which mixes with the community relation vector to predict the value of graph relations. Intuitively, it may be thought of as a modifier which changes the way communities relate with one another in the context of a predicate. The dot product between the community relation vector and the predicate embedding models the probability of a link from entity e_i to entity e_j on the predicate r_r, denoted by g'_{ijr}. To allow for this probabilistic interpretation, we apply an element-wise logistic sigmoid function, σ, on each vector before taking the dot product:

$$g'_{ijr} = \sigma(\mathbf{c}^T_{\mathbf{a}_i\mathbf{a}_j})\sigma(\mathbf{p}_r) \tag{5.8}$$

We can use this to generate the graph by sampling from the Bernoulli distribution:

$$g''_{ijr} \sim \text{Bernoulli}(g'_{ijr}) \tag{5.9}$$

Where g''_{ijr} is a binary value in the generated graph's adjacency tensor.

Since our model is structured as a neural network, its parameters are inferred by a neural compatible optimization method. For this we choose Adaptive Moment Estimation stochastic gradient descent optimization (Adam) [118] and use mean squared error as our loss function, $\mathcal{L}$, to calculate the error in model predictions:

$$\mathcal{L}(\mathbf{X}, \mathbf{X}') = \frac{1}{|\mathcal{E}|^2|\mathcal{R}|} \sum_{i=1}^{|\mathcal{E}|} \sum_{j=1}^{|\mathcal{E}|} \sum_{r=1}^{|\mathcal{R}|} (g_{ijr} - g'_{ijr})^2 \tag{5.10}$$

The generative process is summarized in Algorithm 4. A graphical representation of our model is captured in Figure 5.2.

5.5 Evaluation

Our model is compared against both blockmodels and embedding models where possible. Specifically, we use existing implementations of MMSB, dMMSB, fcMMSB, DDBM, DeepWalk, node2vec, PMNE, and MNE. When evaluating the MMSB, we applied it to each predicate independently. For PMNE, we use the results of the best performing variant for each dataset.

We use five real-world datasets to evaluate the performance of our model: Trade [119]; Vickers-Chan [120]; Lazega [121]; Krebs [122]; and Twitter [123]. Datasets were selected based on previous use in the literature and to be freely available for comparison and reproduction. Their statistics are displayed in Table 5.1. What follows is a brief summary of each dataset.

Trade

This dataset summarizes the trade relations between countries such that each predicate in the graph describes the presence of trade between countries on a specific

Dataset	**# Entities**	**# Predicates**	**# Triples**	**# Classes**
Trade	24	5	1428	2
Vickers-Chan	29	3	740	2
Lazega	71	3	2571	3
Krebs	56	5	2037	5
Twitter	1000	2	25967	None

Table 5.1: Summary of datasets used in the evaluation.

commodity such as manufactured goods or raw minerals. Countries are classified by energy consumption, specifically whether or not they use more than 1000 kilo coal equivalents per capita.

Vickers-Chan

This dataset describes the relations between Australian seventh grade students. Predicates represent a student's relations with respect to questions regarding whom they're friends with and whom they work with. Students are classified by gender.

Lazega

This dataset documents the social network of employees in American law firms. Each employee was asked how they relate to one another in different social contexts, creating the predicates in the graph. Employees are classified by the office they work out of.

Krebs

Krebs' dataset consists of the social relations between IT Department employees in a Fortune 500 company. Employees were asked to rate their interactions with other employees in various contexts on a five point scale. We binarized the values such that any degree of interaction translates to a link between two employees. Employees are classified by their departments.

Twitter

The Twitter dataset describes the relations between Twitter users who tweeted around the time of discovery of the Higgs boson particle. Interactions between users exist on two predicates capturing which users follow one another and which users have retweeted one another at least once. The dataset was truncated to the first 1000 users in our experiments. Users do not have ground truth classes.

5.5.1 Link Prediction

In the link prediction task, a subset of graph relations are unknown and the goal is to predict their value by modelling the remaining, known, relations. For our evaluation, we generated these incomplete graphs by masking relations in the aforementioned datasets. Specifically we created disjoint training and testing sets composed of 80% and 20% of graph relations, respectively. Each of the models was then trained on the training set and evaluated by generating the testing set. The metric we used to measure the performance of the models is the area under the receiver operating characteristic curve (AUC) which is calculated by plotting the predictions' true positive rate against the false positive rate at different thresholds. Higher AUC values indicate better performance. To account for stochasticity in the training process, we repeat this process five times for each dataset. The resulting means and standard deviations are summarized in Table 5.2.

We observe that model performance on the link prediction task is highly dataset dependent with relative AUC scores varying widely between models, as evidenced by three different models achieving top performance on the five datasets. This suggests that there is currently no model which could reliably outperform the others but rather that there is a class of state-of-the-art models with comparable performances. With this in mind, the results indicate that our model is consistent with state-of-the-art methods on the link prediction task. Moreover, it demonstrates higher aggregate performance despite only being the best performing on the Trade and Lazega datasets.

Method	Trade	Vickers-Chan	Lazega	Krebs	Twitter
Blockmodels					
MMSB	0.8679	0.8153	0.8202	0.8335	0.7752
	±0.0418	±0.0420	±0.0246	±0.0759	±0.0825
dMMSB	0.8768	0.8513	0.8155	0.8401	0.9166
	±0.0102	±0.0171	±0.0040	±0.0271	±0.0023
fcMMSB	0.7746	0.7926	0.7642	0.8092	0.9030
	±0.0422	±0.0390	±0.0246	±0.0135	±0.0033
DDBN	0.8525	0.8924	0.8386	0.9276	0.8589
	±0.0145	±0.0127	±0.0056	±0.0048	±0.0040
Embeddings					
DeepWalk	0.5782	0.8340	0.7978	0.8269	0.6027
	±0.0185	±0.0183	±0.0048	±0.0099	±0.0016
node2vec	0.5377	0.8214	0.7821	0.8183	0.6015
	±0.0295	±0.0203	±0.0052	±0.0079	±0.0022
PMNE	0.6207	0.8457	0.8142	0.8887	0.7081
	±0.0295	±0.0138	±0.0084	±0.0052	±0.0027
MNE	0.5444	0.8707	0.8246	0.8544	0.6001
	±0.0696	±0.0103	±0.0078	±0.0050	±0.0036
Our model	0.8797	0.8707	0.8527	0.8948	0.8980
	±0.0130	±0.0172	±0.0028	±0.0071	±0.0064

Table 5.2: Link prediction AUC scores (mean ± standard deviation) on various datasets.

Our model may benefit in this regard from its relatively higher number of hyperparameters which allow it to better fit each dataset. We note, however, that this comes at the cost of hyperparameter tuning which makes it less “out of the box” ready.

We notice that blockmodels and embedding models tend to behave differently from one another on certain datasets. For instance, the mean AUC of blockmodels on the Trade dataset is 0.8430 compared to the 0.5703 of embedding models. The Krebs dataset, on the other hand, shows similar performance between the two approaches with mean AUCs of 0.8526 and 0.8471 for blockmodels and embedding models, respectively. The reason for such dataset sensitivity is not clear and does not appear to be associated with graph entity count, predicate count, or density. One possible explanation for the difference is that the community structures underlying certain datasets lend themselves more naturally to the blockmodelling approach. However, the question of why some methods perform substantially better than other methods on certain datasets is an open question which requires further investigation but is outside of the scope of this work.

5.5.2 Entity Classification

The entity classification task evaluates the quality of entity embeddings by measuring how well they separate entities according to their classes. In this process, each model was trained on all the interactions in a graph and learned entity embeddings were extracted. Two embedding dimensions were considered in this task, $E = 2$ and $E = 10$. They were then classified using a linear support-vector machine classifier using each dataset's ground truth entity classes with a 80% to 20% training to testing split. Recall that ground truth classes are only known for Trade, Vickers-Chan, Lazega, and Krebs datasets. Furthermore, only models which produce embeddings could be considered in this part of the evaluation, excluding all other blockmodels. Due to the small sizes of testing data, we repeated the process ten times for each dataset. We used accuracy as the evaluation metric, defined as the fraction of correctly classified entities over all entities. The results are displayed in Table 5.3. We also provide visualizations of the learned embeddings in two dimensional space in Figure 5.3.

Overall, our model performs comparably to other embedding models on entity classification. As in the link prediction task, we notice that no model consistently outperforms the others, suggesting a degree of dataset and embedding size sensitivity.

Method	Trade	Vickers-Chan	Lazega	Krebs
Deepwalk				
$E = 2$	0.3200 ± 0.1600	1.0000 ± 0.0000	0.7733 ± 0.0998	0.6833 ± 0.1856
$E = 10$	0.3200 ± 0.0980	1.0000 ± 0.0000	0.9333 ± 0.0596	0.8667 ± 0.1130
node2vec				
$E = 2$	0.3200 ± 0.0980	1.0000 ± 0.0000	0.7067 ± 0.1236	0.7167 ± 0.0850
$E = 10$	0.3200 ± 0.0980	1.0000 ± 0.0000	0.9200 ± 0.0267	0.8333 ± 0.0913
PMNE				
$E = 2$	0.3200 ± 0.0980	0.9000 ± 0.0816	0.9333 ± 0.0422	0.7333 ± 0.0133
$E = 10$	0.3200 ± 0.0980	0.9667 ± 0.0667	0.9200 ± 0.0499	0.8500 ± 0.0333
MNE				
$E = 2$	0.3800 ± 0.2088	1.0000 ± 0.0000	0.9333 ± 0.0667	0.7167 ± 0.1546
$E = 10$	0.3600 ± 0.1200	1.0000 ± 0.0000	0.9267 ± 0.0554	0.8500 ± 0.1041
MNB				
$E = 2$	0.7200 ± 0.1600	1.0000 ± 0.0000	0.7867 ± 0.0778	0.7333 ± 0.0624
$E = 10$	0.8600 ± 0.2010	1.0000 ± 0.0000	0.9200 ± 0.0581	0.8083 ± 0.1057

Table 5.3: Entity classification accuracy scores (mean ± standard deviation) on various datasets.

For instance, our model shows top performance on the Trade dataset, consistent with results from the link prediction task, yet slightly underperforms on other datasets. All models perform well on the Vickers-Chan dataset as shown by the corresponding embeddings in Figure 5.3, suggesting the classes are highly separable on this dataset. In general, that larger embedding dimensions performed better than the smaller one. The reason for this is that larger embeddings can encode more feature information for each entity and a higher dimensionality allows for easier linear separability. We conclude that our model is able to generate embeddings of similar quality compared to state-of-the-art embedding methods.

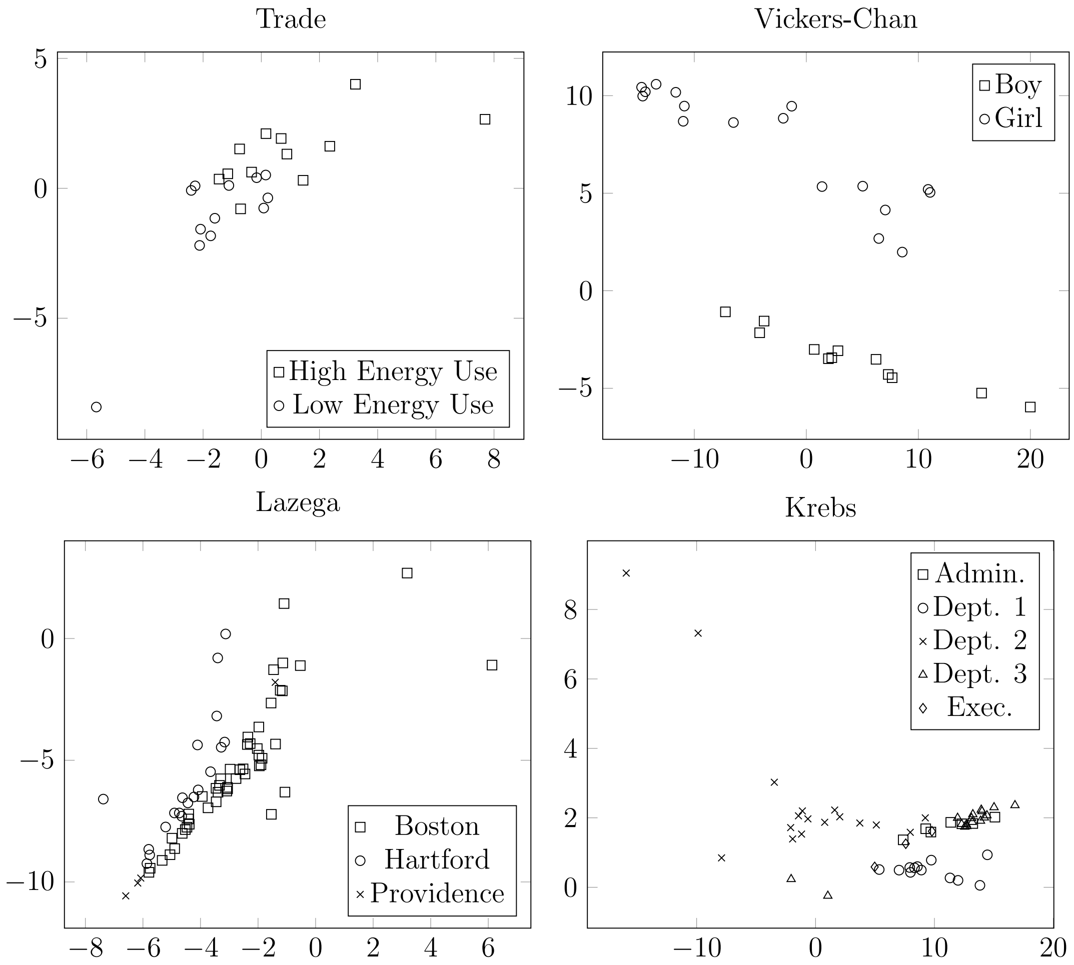


Figure 5.3: Scatterplots of two dimensional entity embeddings on various datasets.

5.5.3 Community Detection

The community detection task is related to the entity classification task in that it evaluates a model's ability to cluster entities based on similar properties. The key difference between the two is that while entity classification evaluates a model's entity embeddings and their ability to implicitly cluster entities, the community detection task evaluates a model's explicit clusterings via community membership assignments. In our evaluation procedure, each model was first trained on all relations in the graph and the most probable community for each entity was extracted. In probabilistic models, this was done by sampling two hundred community membership indicators and using the most frequently occurring community. In neural methods this was done by taking the community with the highest membership degree. Due to the absence of ground truth classes for varying K values, we used internal clustering metrics to evaluate the inferred communities, namely conductance [124] and normalized cut size [125]. Both of these metrics measure the degree of connectivity between intra-cluster entities with respect to inter-cluster entities such that lower values indicate better performance. Specifically, conductance measures the quotient between the volumes of two sets of entities. We define the conductance on each predicate in the knowledge graph, $conductance_r$ as follows:

$$conductance_r = \frac{\sum_{i \in \mathcal{E}'} \sum_{j \in \mathcal{E}''} g_{ijr}}{\min(\sum_{i \in \mathcal{E}'} \sum_{j \in \mathcal{E}} g_{ijr}, \sum_{i \in \mathcal{E}''} \sum_{j \in \mathcal{E}} g_{ijr})} \tag{5.11}$$

Where $\mathcal{E}'$ and $\mathcal{E}''$ are the entities in the knowledge graph partitioned such that $\mathcal{E} = \mathcal{E}' \cup \mathcal{E}''$ and $\emptyset = \mathcal{E}' \cap \mathcal{E}''$. The normalized cut is calculated as the size of the partition cut times the sum of the volume reciprocals of the two partitions:

$$NC_r = \left(\sum_{i \in \mathcal{E}'} \sum_{j \in \mathcal{E}''} g_{ijr} \right) \left(\frac{1}{\sum_{i \in \mathcal{E}'} \sum_{j \in \mathcal{E}} g_{ijr}} + \frac{1}{\sum_{i \in \mathcal{E}''} \sum_{j \in \mathcal{E}} g_{ijr}} \right) \tag{5.12}$$

The average conductance and normalized cut across all predicates was used to obtain the final metrics across the entire knowledge graph.

We evaluated each model on two community sizes: $K = 2$ and $K = 4$. For fcMMSB we set the number of groups (G) analogously to the number of communities, $G = 2$ and $G = 4$. As in the link prediction task, we applied the MMSB predicate by predicate and took the mean of the results. Since embedding models do not perform explicit community detection, we only compare against other blockmodels. Each model was trained five times. The results are summarized in Table 5.4.

In general, our model tends to outperform the other blockmodels on both metrics and at both community sizes. The largest difference in relative performance is seen on the datasets with the most predicates, Trade and Krebs, suggesting that mixing a relation embedding with community interactions is a viable approach to capturing relationally invariant communities. Furthermore, models with fewer communities outperform those with more communities. One possible explanation for this is that a higher number of communities increases the number of model parameters thus increasing the propensity to overfit the data. We note that our model induces the least coherent communities on the Twitter dataset. Interestingly, however, the other models which performed well on the link prediction task on this dataset (dMMSB and fcMMSB) also perform relatively poorly. This suggests that there exist peculiarities in the Twitter dataset wherein communities which are generate entity relations with high performance are not the ones with low conductances and normalized cut sizes.

5.5.4 Predicate Embeddings

Predicate embeddings provide insight into the information modelled by each knowledge graph predicate since they act as a predicate specific modifier on community and entity relations. Thus, predicates with similar semantics should be embedded close to one another in the predicate embedding space. We compared the similarities between predicates and their embeddings quantitatively by looking at the average similarities of each pairwise predicate subgraph and embedding. Due to the NP-Hard time complexity of computing subgraph similarities via the Graph Edit Distance [126], we adopted a simplified approach which compares the degree of similarity between predicate subgraphs as the proportion of same entries in their respective adjacency tensors. For embedding similarity, we calculated the average cosine similarity between each embedding and the others for each dataset. The results of this procedure are described in Table 5.5. We notice the high degree of dissimilarity in the `minerals` predicate embedding on the Trade dataset is reflected in that it is the least similar subgraph relative to the other ones. The `get on with` predicate on the Vickers-Chan dataset and the `work with` predicate on the Lazega dataset follow the same phenomenon. Predicate embedding similarity on the Krebs dataset is reflected in the similarity between its predicate subgraphs.

	Trade		Vickers-Chan		Lazega		Krebs		Twitter	
Method	Cond.	NC	Cond.	NC	Cond.	NC	Cond.	NC	Cond.	NC
MMSB										
$K = 2$	1.6780	2.5277	1.0622	1.6465	1.0008	1.7353	1.3459	1.9175	1.2950	2.1856
	±0.3111	±0.1982	±0.3411	±0.4282	±0.1518	±0.1933	±0.4599	±0.6069	±0.2486	±0.2131
$K = 4$	2.0239	2.5731	1.2673	1.6738	1.3843	1.8314	1.4023	1.8012	1.6420	2.1556
	±0.2510	±0.2517	±0.1925	±0.2397	±0.1235	±0.1614	±0.1989	±0.2607	±0.1751	±0.1907
dMMSB										
$K = 2$	1.7184	2.5604	1.2788	1.7502	1.2291	1.8821	1.6324	2.2407	3.2061	3.7244
	±0.2422	±0.1937	±0.3864	±0.1590	±0.1458	±0.1097	±0.3734	±0.2137	±1.9318	±1.5897
$K = 4$	1.9284	2.4654	1.3704	1.7691	1.4409	1.9122	1.6047	2.0690	2.3241	2.8486
	±0.1880	±0.1731	±0.2819	±0.3008	±0.0750	±0.0916	±0.1561	±0.1797	±0.9502	±0.9661
DDBN										
$K = 2$	1.8312	2.0096	1.3398	1.6351	1.8124	2.0123	1.5638	1.6492	1.1670	1.8951
	±0.0000	±0.0000	±0.0000	±0.0000	±0.1721	±0.0913	±0.0000	±0.0000	±0.1218	±0.1522
$K = 4$	4.3723	4.6180	1.3142	1.6433	1.1887	1.5504	0.8777	1.1465	1.5977	1.8010
	±0.2296	±0.2055	±0.1131	±0.1043	±0.0404	±0.0420	±0.2552	±0.2749	±0.2244	±0.3806
fcMMSB										
$G = 2$	2.2501	2.7666	1.0812	1.6067	1.4995	1.9915	1.0538	1.6878	1.4734	1.6958
	±0.9639	±0.7634	±0.2471	±0.2686	±0.2405	±0.0967	±0.2148	±0.2854	±1.3274	±1.2327
$G = 4$	2.1971	2.6536	1.3488	1.7595	1.3570	1.7906	1.2889	1.6787	0.9198	1.1103
	±0.5501	±0.4846	±0.2156	±0.2649	±0.1093	±0.1353	±0.1891	±0.2336	±0.2550	±0.2310
MNB										
$K = 2$	1.4900	2.5698	0.7606	1.3161	1.1047	1.7754	0.4386	0.6692	2.8168	3.0646
	±0.0000	±0.0000	±0.0000	±0.0000	±0.0330	±0.0440	±0.0000	±0.0000	±0.2837	±0.3090
$K = 4$	1.7313	2.2945	1.1643	1.5456	1.3035	1.7083	0.8908	1.1395	3.4614	3.7755
	±0.0289	±0.0269	±0.0367	±0.0381	±0.0815	±0.0974	±0.1480	±0.1941	±0.4343	±0.4530

Table 5.4: Community detection conductance (Cond.) and normalized cut (NC) scores (mean ± standard deviation) on various datasets.

The low similarity between predicate embeddings coupled with the high similarity in predicate subgraphs on the Twitter dataset may be explained by the fact there are only two subgraphs allowing for overfitting. This is supported by evidence in the following subsection which shows that a predicate embedding dimension of $A = 1$ performs the best on the link prediction task on this dataset, implying overfitting at higher A values.

5.5.5 Hyperparameter Sensitivity

Hyperparameter selection is an important consideration for our model as it balances the trade-off between underfitting and overfitting of the data. Thus, for each task and dataset, we performed hyperparameter exploration on the model's three hyperparameters from the sets $E \in \{2, 4, 8\}$, $K \in \{2, 4, 8, 16\}$, and $A \in \{1, 2, 4, 8, 16\}$. The best performing hyperparameters are summarized in Table 5.6.

We notice that hyperparameters appear to be more sensitive to the task performed rather than the dataset in our model. Furthermore, close inspection of the hyperparameter exploration results suggests that E is the least sensitive of the three. High A values correlate negatively with community detection performance, perhaps due to higher values allowing for more expressive community interactions and thus lead to overfitting.

Drawing conclusions or "rules of thumb" about a priori hyperparameter selection is difficult, thus we suggest at least a cursory hyperparameter tuning when applying our model to new datasets. The hyperparameters from the aforementioned sets are a good starting point and the constraint $E \leq K \leq A$ may be applied to reduce the search space.

Dataset and Predicates	**Subgraph Similarity**	**Embedding Similarity**
Trade		
manufactured goods	0.7552	0.1338
foods	0.7253	0.3247
crude materials	0.7418	0.2781
minerals	0.6194	-0.8303
diplomacy	0.6949	0.3212
Vickers-Chan		
get on with	0.7782	-0.9993
friends with	0.8424	-0.0002
work with	0.8347	0.0003
Lazega		
advice from	0.8430	0.2725
friends with	0.8193	0.2446
work with	0.8112	-0.2760
Krebs		
work with BP 1	0.8870	0.9914
work with BP 2	0.9035	0.9949
advice from	0.8893	0.9956
expertise from	0.8984	0.9876
discuss with	0.8831	0.9902
Twitter		
follows	0.9744	-0.9963
retweets	0.9744	-0.9963

Table 5.5: Predicate subgraph and embedding similarities.

Task and Dataset	***E***	***K***	***A***
Link Prediction			
Trade	8	4	8
Vickers-Chan	2	8	16
Lazega	8	8	16
Krebs	8	16	16
Twitter	8	16	1
Node Classification			
Trade	–	8	8
Vickers-Chan	–	4	8
Lazega	–	8	8
Krebs	–	4	4
Community Detection			
Trade	2	–	1
Vickers-Chan	4	–	4
Lazega	4	–	4
Krebs	2	–	1
Twitter	2		1

Table 5.6: Optimal hyperparameters for task and dataset.

5.5.6 Conclusions

In this chapter we proposed a novel method for knowledge graph modelling which fuses blockmodels with graph embedding models in a neural architecture. Our model utilizes the block structure of other blockmodelling methods, namely decomposing the graph into communities of entities to model their interactions. Furthermore, it leverages entity embeddings to generate community memberships, drawing inspiration from embedding models. Finally, our model handles knowledge graph predicates by mixing a predicate embedding with the community relations. In doing so, our model "fills in the gaps" left behind by both blockmodels and embedding models. We demonstrated this by evaluating it on three tasks using real-world datasets: link prediction, entity classification, and community detection. On each of these tasks our model achieves better than or comparable performance with respect to state-of-the-art blockmodels and embedding models. Moreover, only our model is able to perform these tasks explicitly, highlighting its flexibility.

Chapter 6

Hierarchical Blockmodelling for Knowledge Graphs

This chapter describes the work presented in the paper *Hierarchical Blockmodelling for Knowledge Graphs*. As mentioned earlier, this paper is still unpublished and intended for publication in the Semantic Web Journal.

6.1 Introduction

Entity clustering refers to the task of grouping together entities in a knowledge graph which share similar properties. The measure by which entities are judged to be similar varies and is one of the key considerations when devising an approach to their clustering. Obtaining an entity clustering allows for the discovery of structures which are implicit in the knowledge graph and provides insight into the number and types of categories which exist in the data. The process operates on unlabelled data and is therefore a type of unsupervised learning. As such, it is one of the first and most useful operations applied to a knowledge graph when performing exploratory analysis. Hierarchical clustering of a knowledge graph's entities extends the clustering task by imposing a hierarchical organization to the clusters themselves. This allows not only to discover which entities are semantically similar as per the clustering but also how entities relate to one another hierarchically.

In this regard, this chapter proposes a generative model for knowledge graphs which induces a clustering of entities and organizes it hierarchically. Similar to the approach described in Chapter 5, the proposed model is formulated as a stochastic blockmodel albeit this time in a fully probabilistic framework. In broad strokes, this model operates by decomposing a knowledge graph into a set of probability distributions which are then sampled from to generate the knowledge graph. As a byproduct of this sampling process, a hierarchical clustering of knowledge graph entities is induced. To the best of our knowledge, our approach is the first to apply stochastic blockmodels to knowledge graphs and one of a very few probabilistic graphical models to be used for the purpose of knowledge graph hierarchy induction.

6.2 Related Work

The work presented in this chapter lies at the intersection of two areas in artificial intelligence which deal with modelling graph data: stochastic blockmodelling and hierarchy induction. The related works for the former and latter of the pair were largely discussed in Chapters 5 and 3, respectively. As such, this section focuses on what was previously left out, specifically the hierarchical extensions to the probabilistic methods discussed earlier.

A hierarchical extension of the IRM, the aptly named Hierarchical Infinite Relational Model [127], was developed to overcome the overfitting problem in the original. The MMSB was extended to the hierarchical case in the Multiscale Community Blockmodel [16]. This model is closely related to the one proposed in this chapter and operates by placing nCRP and stick breaking priors over the community relations of the original model. These ideas are explored further when defining our model in Section 6.4. In another method which bears similarity to our own, Zhang et al. [17] use a non-parametric Bayesian approach to induce a hierarchy of topic communities. This model draws heavily from topic models used in natural language processing such as those of Blei et al. [24] and Paisley et al. [128]. Despite a similar statistic framework and inference scheme, the hierarchy induced by this work differs significantly from our own. For instance, relations between communities are not modelled and entities are never explicitly assigned to communities. Along similar lines is GMMSchema [129] which uses a Gaussian mixture model to generate a schema graph which can be viewed as a hierarchical abstraction of the original knowledge graph.

6.3 Problem Description

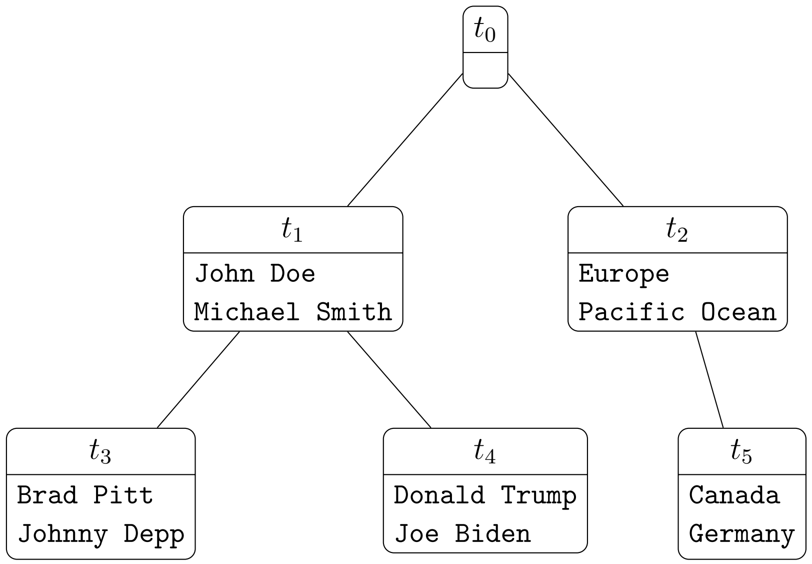
The task of hierarchical clustering in the context of knowledge graphs may be defined by its goal, namely to find a surjective mapping from the set of entities to the set of clusters (henceforth referred to as communities). Formally, we seek to find f such that $f : \mathcal{E} \mapsto \mathcal{T}$ where $\mathcal{E}$ and $\mathcal{T}$ are the sets of entities and communities, respectively, as introduced in Chapter 2. Entities which share the same community after clustering should share similar semantics. Note that this does not suggest that they must be proximal in the knowledge graph. To highlight this, consider the triples ⟨`Henry_Ford`, `dbo:birthPlace`, `dbr:Michigan`⟩ and ⟨`Henry_Ford`, `dbo:spouse`, `dbr:Clara_Bryant_Ford`⟩ introduced in Figure 2.1. Despite both `dbr:Michigan` and `dbr:Clara_Bryant_Ford` being equidistant in proximity to `Henry_Ford`, their semantic similarity is not. This feature of knowledge graphs, which separates them from simple graphs in which proximity is of foremost importance, was discussed in the context of second order neighbours in Chapter 4.

In addition to finding a mapping from entities to communities, hierarchical clustering necessitates a hierarchical organization of its communities. Although the ways in which this is formulated varies, in this chapter we will draw upon the notion of subsumptions introduced in Chapter 3. Specifically, we use $\{t_p \rightarrow t_q\}$ to denote that community t_p subsumed community t_q. In other words, community t_p is the parent of community t_q in the hierarchy. Furthermore, given our model formulation and its use of the nCRP, we restrict the hierarchy implied by its subsumptions to take on the structure of a tree.

6.4 Proposed Model

Like all stochastic blockmodels, our model is defined as a set of probability distributions such that when these distributions are sampled from, they generate the adjacency tensor of the knowledge graph. The choice of these distributions makes assumptions about the underlying structure that governs the graph's relations. In devising our model, we assume a hierarchy of entity communities which are captured in the form of a tree. The entities in these communities interact with one another as a function of their membership to a community. In other words, relations are modelled at the community level and extended downwards to their constituent entities. Unlike most stochastic blockmodels, these community relations are modelled with respect to a predicate in the knowledge graph. This allows the model to capture structures extending beyond those implied by mere relation density. Thus, in order generate the knowledge graph's adjacency tensor, we need to know its hierarchical community structure, its entities' memberships to communities, and the relations between its communities. The induction of these components, which may be seen as a byproduct of the generative process, is the objective of our model. We note that the communities' constituent entities do not conform to is-a relationships as would be implied by the hierarchy. This is because the hierarchy is imposed on the communities themselves as opposed to their constituent entities. An example of this is highlighted in Figure 6.1 where the entity `Canada` is a descendant of the entity `Pacific Ocean`. Of course, `Canada` is not a `Pacific Ocean` however the concept modelled by community t_5, namely countries, is an instance of the concept modelled by community t_2, namely locations.

6.4.1 Community Memberships



Entity	**Path**	**Level**	**Community**
Brad Pitt	t_1 t_3	2	t_3
Canada	t_2 t_5	2	t_5
Donald Trump	t_1 t_4	2	t_4
Europe	t_2 t_5	1	t_2
Germany	t_2 t_5	2	t_5
Joe Biden	t_1 t_4	2	t_4
John Doe	t_1 t_4	1	t_1
Johnny Depp	t_1 t_3	2	t_3
Michael Smith	t_1 t_3	1	t_1
Pacific Ocean	t_2 t_5	1	t_2

Figure 6.1: Toy example depicting a potential hierarchy induced by our model. The table on right side captures the path and level sampled for each entity in the knowledge graph as well as its corresponding community. The left side provides a visualization of this hierarchy.

Entities are assigned to communities through the conjunction of two variables: entity paths and level indicators. Paths define the tree structure over the community hierarchy by sampling from the nCRP as described in Chapter 2. We thus denote an entity path as $\mathbf{p}_i$ for entity e_i, such that $\mathbf{p}_i := [p_i^1, p_i^2, \ldots, p_i^L]$ where p_i^l represents the community at level l. We draw attention to the fact that this definition omits the root community from the path, namely p_i^0, since all entities must pass through it. It also allows a hierarchy with a depth of L to have entity path vectors of dimension L, simplifying the notation. Entity paths are drawn from the nCRP, denoted as $\mathbf{p}_i \sim \text{nCRP}(\gamma)$. Thus, all the entity paths sampled in the model form a $|\mathcal{E}| \times L$ matrix which we denote as $\mathbf{P}$. γ is the aforementioned hyperparameter of the nCRP and is responsible for controlling the probability of generating a new branch in the hierarchy as the path is being sampled. When a new branch is generated at level l such that $l < L$, $L - l$ new communities are also generated and populated solely by the sampling entity. Furthermore, if a path is resampled such that its corresponding entity obtains a new path which leaves behind empty communities, those empty communities and removed from the hierarchy. As such, the number of communities in the hierarchy is subject to constant change throughout the sampling process.

Having sampled entity paths, in order for entities to be assigned to communities, their levels must be obtained. Entity levels are modelled by two variables in our approach: level memberships and level indicators. Level memberships, denoted $\mathbf{a}_i$ for entity e_i, capture the probability of the entity's belonging to each of the L levels. As such, all the level memberships in our model form a $|\mathcal{E}| \times L$ matrix, $\mathbf{A}$. This is similar to the mixed-membership property of the MMSB wherein an entity has a membership distribution over all communities. The difference, as pointed out by Ho et al. [16], is that in hierarchical models this distribution is restricted to communities along the entity's sampled path, otherwise the process of obtaining paths, and indeed the hierarchy itself, would lose its meaning. Level memberships are drawn from the stick breaking process, $\mathbf{a}_i \sim \text{Stick}(\mu, \sigma)$ with hyperparameters μ and σ. Recall that this process yields an infinite distribution and must therefore be truncated to a dimension of L to correspond with the depth of the tree. The truncation is performed by removing all probabilities at levels greater than L and renormalizing. The distribution captured by an entity's level membership is used to sample its level indicator. The level indicator indicates the level to which an entity belongs and thus, in conjunction with its path, assigns it to a community. Level indicators are drawn in the context of a relation between two entities. Specifically, when modelling the probability of a relation from entity e_i to entity e_j we draw two level indicators, one for the sender entity and one for the receiver entity denoted as $z_{i\to j}$ and $z_{i\leftarrow j}$, respectively. The sender and receiver level indicators correspond to the levels of entities e_i and e_j in the context of their pairwise interaction. Thus, our model samples $|\mathcal{E}|^2$ sender and receiver level indicators each leading to two $|\mathcal{E}| \times |\mathcal{E}|$ matrices $\mathbf{Z}_\to$ and $\mathbf{Z}_\leftarrow$ for all the senders and receivers, respectively. To simplify notation in our inference procedure, we concatenate these matrices to form a $|\mathcal{E}| \times |\mathcal{E}| \times 2$ level indicator tensor, $\mathbf{Z}$. Since level memberships are themselves probability distributions, they may be sampled from directly to indicate an entity's level. Specifically, level indicators are drawn from multinomial distributions, namely $z_{i\to j} \sim \text{Multinomial}(\mathbf{a}_i)$ and $z_{i\leftarrow j} \sim \text{Multinomial}(\mathbf{a}_j)$, which yield one of the L levels in the hierarchy.

The interplay between paths and levels when assigning entities to communities may be summarized as follows: paths identify a hierarchy of candidate communities and level indicators select one of the candidates for the entity. This dynamic is captured in the toy example in Figure 6.1.

6.4.2 Community Relations

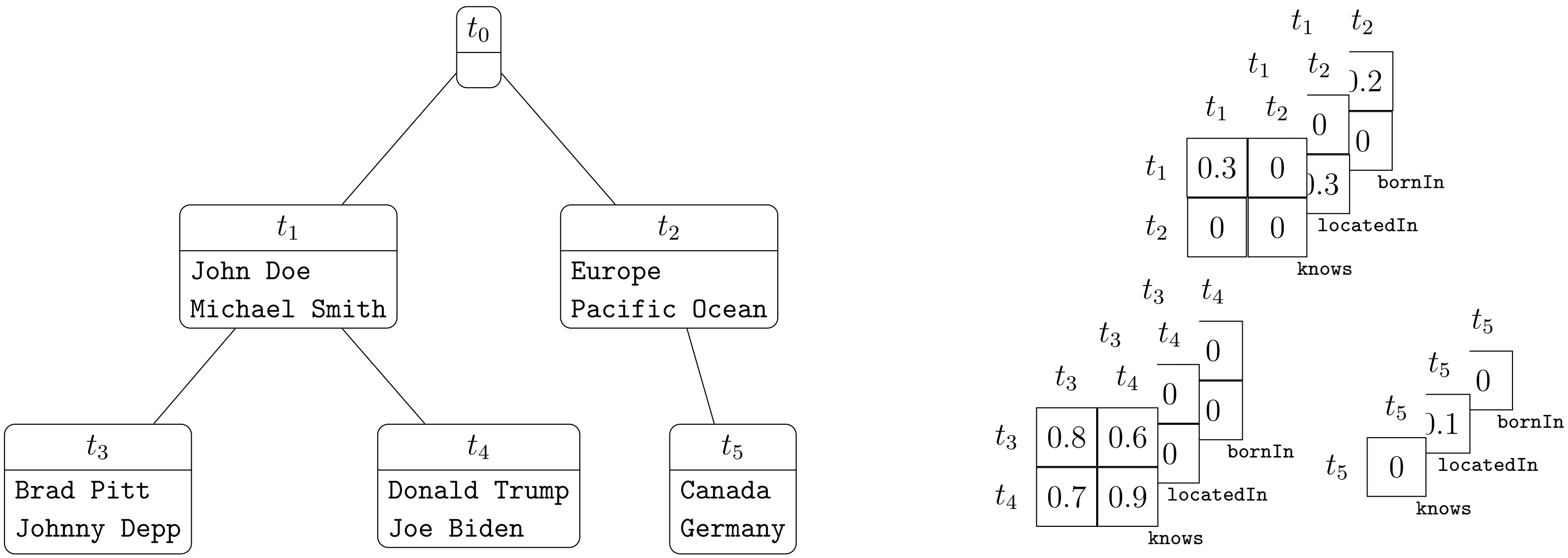


Figure 6.2: The potential community relations induced by our model on the toy example introduced earlier. The hierarchy on the left of the figure has three sibling groups and three predicates: `knows`, `locatedIn`, and `bornIn`. The three tensors on the right correspond to the community relations of the three sibling groups.

Community relations describe the degree to which entities in any two communities are likely to interact with one another through a specific predicate. In other words, they model the probability of observing a value of one in the knowledge graph's adjacency tensor. These relations are captured by a $|\mathcal{T}| \times |\mathcal{T}| \times |\mathcal{R}|$ tensor, denoted $\mathbf{C}$, where $\mathcal{T}$ is the set of all communities in the hierarchy and $\mathcal{R}$ is the set of all relations in the knowledge graph. We note that because the communities in $\mathcal{T}$ are a result of sampling from the nCRP and are thus subject to change with each successive sample, the dimensionality of $\mathbf{C}$ is also subject to change in the sampling process. This presents a challenge to our sampling scheme since it is possible to sample communities via the nCRP for which there are no community relation values. We overcome this issue through the marginalization of community relations as discussed in the subsequent subsection. The community relation c_{pqr} is an entry in $\mathbf{C}$ and captures the probability of a relation between entities in community t_p with entities in community t_q through predicate r_r. As such, the value of c_{pqr} is bounded to $1 \geq c_{pqr} \geq 0$. In order to preserve the hierarchical structure that was induced by sampling paths and levels, the community relations must be limited to take on non-zero values only when interacting with communities which are proximal to them in the hierarchy. This restriction is vital as allowing for a relation between any two communities in the hierarchy would render it meaningless and our model would be reduced to a fixed size mixed membership stochastic blockmodel such as the ones described in Chapter 2.

In restricting the values of community relations we take an approach similar to that of the Multiscale Community Blockmodel. Specifically, we borrow the concept of a sibling group which refers to a set of communities that share the same parent in the hierarchy. Only the community relations between communities in the same sibling group are modelled in our approach. Thus, when obtaining the relation degree of two entities whose communities have the same parent, it's sufficient to merely access the corresponding value in $\mathbf{C}$. When their communities do not share the same parent, a coarsening procedure is applied to obtain a relation degree. The coarsening procedure traverses the paths of the two entities to find the deepest pair of communities which are in the same sibling group. Formally, to obtain the community relation degree from entity e_i to entity e_j on predicate r_r, we define the function $\Psi(i, j, r)$ as follows:

$$\Psi(i, j, r) = \begin{cases} c_{p_i^{z_{i \to j}} p_j^{z_{i \leftarrow j}} r} & p_i^{z_{i \to j} - 1} = p_j^{z_{i \leftarrow j} - 1} \\ c_{\Phi(i \mid j) \Phi(j \mid i) r} & p_i^{z_{i \to j} - 1} \neq p_j^{z_{i \leftarrow j} - 1} \end{cases} \tag{6.1}$$

Wherein $\Phi(i \mid j)$ and $\Phi(j \mid i)$ are functions that find the ancestor communities of entities e_i and e_j which share the same sibling group, respectively. These values are obtained by indexing the entities' paths on the level at which they diverge from their partner. This process is made clear in their definitions:

$$\begin{aligned} \Phi(i \mid j) &= p_i^{min(\{l \,:\, p_i^l \neq p_j^l\})} \\ \Phi(j \mid i) &= p_j^{min(\{l \,:\, p_i^l \neq p_j^l\})} \end{aligned} \tag{6.2}$$

This approach differs from the Multiscale Community Blockmodel in that while there are restricted entries in $\mathbf{C}$, these values are never accessed. Instead, all communities are coarsened to an ancestor which allows for their relation to take on a non-restricted value.

Community relations are drawn from the Beta distribution parameterized by $\lambda > 0$ and $\eta > 0$, and denoted as $c_{pqr} \sim \text{Beta}(\lambda, \eta)$. This ensures that community relations take on probability values which can be used in conjunction with the Bernoulli distribution. λ and η are hyperparameters of our model and determine the density of the generated knowledge graph such that increasing λ values with respect to η yields denser results. Figure 6.2 provides a visualization of a potential sampling of community relations. We note that the three tensors correspond to the three sibling groups for which community relations take on non-restricted values. The diagonal and off-diagonal values in these tensors represent the intra and inter community relations, respectively. Thus, based on these values, there is a probability of 0.8 that `Brad Pitt knows Johnny Depp` and a probability of 0.6 that `Brad Pitt knows Donald Trump`.

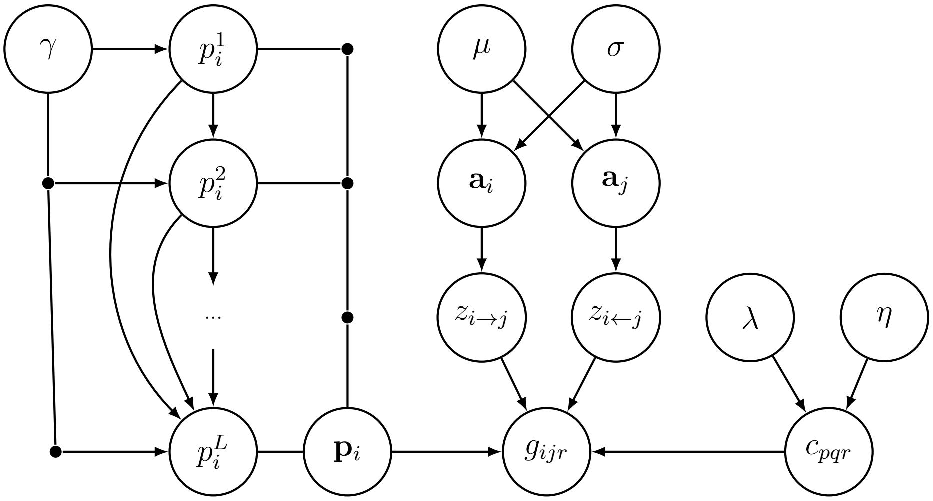


Figure 6.3: Plate diagram for our model.

We provide an exploration of the recovered community relations on real-world data in Section 6.5.

6.4.3 Generative Process

The generative process of our model refers to the sequential sampling of components which allow for the generation of the target knowledge graph. In other words, the goal is to draw a binary value for each $g_{ijr} \in \mathbf{G}$ such that it equals the knowledge graph's adjacency tensor. But before this can be done, it's necessary to sample the variables it is dependent on. The first components sampled in the generative process are the paths and level memberships for each entity in the knowledge graph from the nCRP and stick distributions, respectively. Having drawn the paths, we now have the set of communities in the hierarchy and can draw community relations from the Beta distribution. At this point in the generative process, entities are not yet assigned to communities. The community memberships for these entities have been drawn, however, allowing for the sampling of community levels for each pair of entities in the knowledge graph from the multinomial distribution. With the community levels drawn, all the components for generating the knowledge graph are in place. The binary value for the relation from entity e_i to entity e_j on predicate r_r is drawn from the Bernoulli distribution using each entity's respective community's relations, namely $g_{ijr} \sim \text{Bernoulli}(\Psi(i, j, r))$. The plate diagram for this process is illustrated in Figure 6.3 and the formal definition is as follows:

- For each entity in the knowledge graph; $e_i \in \mathcal{E}$
    - $\mathbf{p}_i \sim \text{nCRP}(\gamma)$
    - $\mathbf{a}_i \sim \text{Stick}(\mu, \sigma)$
- For each sender community in the hierarchy; $t_p \in \mathcal{T}$
    - For each receiver community in the hierarchy; $t_q \in \mathcal{T}$
        - For each predicate in the knowledge graph; $r_r \in \mathcal{R}$
            - $c_{pqr} \sim \text{Beta}(\lambda, \eta)$
- For each sender entity in the knowledge graph; $e_i \in \mathcal{E}$
    - For each receiver entity in the knowledge graph; $e_j \in \mathcal{E}$
        - $z_{i \to j} \sim \text{Multinomial}(\mathbf{a}_i)$
        - $z_{i \leftarrow j} \sim \text{Multinomial}(\mathbf{a}_j)$
        - For each predicate in the knowledge graph; $r_r \in \mathcal{R}$
            - $g_{ijr} \sim \text{Bernoulli}(\Psi(i, j, r))$

We note that this process is unsupervised and does not impose any assumptions about the partition of entities to communities or the structure of the hierarchy other than to limit its depth. In fact, the depth is the only constraint imposed on the generative process. The other hyperparameters which must be specified a priori – namely γ, μ, σ, λ, and η – merely influence the prior distributions of our model. They may pull the latent variables in the assumed direction but only insofar as the data allows it. This, recall, is due to the sampling of latent variables from their posterior distribution which is conditioned on the data. As a result, with a strong enough likelihood, the effects of the hyperparameters and the prior relatively diminish. The joint distribution of our model may be expressed as follows:

$$\begin{aligned}\mathbb{P}(\mathbf{G}, \mathbf{C}, \mathbf{P}, \mathbf{Z}, \mathbf{A} \mid \gamma, \mu, \sigma, \lambda, \eta) = &\prod_{g_{ijr}} \mathbb{P}(g_{ijr} \mid c_{pqr}, \mathbf{p}_i, z_{i\rightarrow j}, z_{i\leftarrow j}) \prod_{c_{pqr}} \mathbb{P}(c_{pqr} \mid \lambda, \eta) \\ &\prod_{\mathbf{p}_i} \mathbb{P}(\mathbf{p}_i \mid \gamma) \prod_{z_{i\rightarrow j}} \mathbb{P}(z_{i\rightarrow j} \mid \mathbf{a}_i) \prod_{z_{i\leftarrow j}} \mathbb{P}(z_{i\leftarrow j} \mid \mathbf{a}_j) \\ &\prod_{\mathbf{a}_i} \mathbb{P}(\mathbf{a}_i \mid \mu, \sigma) \qquad (6.3)\end{aligned}$$

As with most stochastic blockmodels, the exact inference for our model is intractable and must be approximated using an inference scheme. For this we adopt collapsed Gibbs sampling, an extension of the aforementioned Gibbs sampling.

6.4.4 Collapsed Gibbs Sampling

Collapsed Gibbs sampling refers to an extension of Gibbs sampling in which a subset of model variables are marginalized over and therefore do not need to be sampled directly. These variables are said to be collapsed out of the Gibbs sampler. Collapsing of these variables is done analytically via integration and ensures a faster mixing process. This is because the calculation of probability distributions for sampling is generally computationally expensive. Having fewer variables then leads to a faster arrival at the desired stationary distribution. Furthermore, the calculation of probability distributions which have not been collapsed out of the sampling process is generally faster in collapsed Gibbs sampling. This is because in regular Gibbs sampling draws are made from the full conditionals of variables. In collapsed Gibbs sampling, collapsed variables have been integrated out of the process and the remaining variables are conditioned on a lower-dimensional space. Collapsing of variables is usually tractable when they are the conjugate prior of their dependent variables. In our model, community relations and level memberships are both conjugate priors of their dependant variables, namely level indicators and entity relations, respectively. We leverage these conjugacies to marginalize over these two variables in our sampling process. After marginalization, the sampling equations may be derived for the remaining variables.

Marginalizing Community Relations

In order to marginalize out community relations, it is necessary to find a closed form solution which allows for integration during path sampling. To this end, we can leverage the Bernoulli-Beta conjugacy which ensures that given a Bernoulli likelihood and Beta prior, the posterior will also be drawn from the Beta distribution. Employing this conjugacy is possible due to the formulation of our model in which entity relations are drawn from the Bernoulli distribution and community relations assume a Beta prior. We see this explicitly when applying Bayes' theorem to obtain the posterior as follows:

$$\mathbb{P}(c_{pqr} \mid \mathbf{C}_{-(pqr)}, \mathbf{G}, \mathbf{P}, \mathbf{Z}, \lambda, \eta) = \frac{\mathbb{P}(\mathbf{G} \mid \mathbf{C}, \mathbf{P}, \mathbf{Z}, \lambda, \eta)\mathbb{P}(c_{pqr} \mid \mathbf{C}_{-(pqr)}, \lambda, \eta)}{\int_{c_{pqr}} \mathbb{P}(\mathbf{G} \mid \mathbf{C}, \mathbf{P}, \mathbf{Z}, \lambda, \eta)\mathbb{P}(c_{pqr} \mid \mathbf{C}_{-(pqr)}, \lambda, \eta) \, \mathrm{d}c_{pqr}} \tag{6.4}$$

Where $\mathbb{P}(\mathbf{G} \mid \mathbf{C}, \mathbf{P}, \mathbf{Z}, \lambda, \eta)$ is the likelihood of generating entity relations and $\mathbb{P}(c_{pqr} \mid \mathbf{C}_{-(pqr)}, \lambda, \eta)$ is the prior placed on community relations. $\mathbf{C}_{-(pqr)}$ indicates the community relations tensor $\mathbf{C}$ without c_{pqr}. Before proceeding we introduce helper variables $\#^{c_{pqr}=1}$ and $\#^{c_{pqr}=0}$ to indicate the number of existing and non-existing relations between entities from community t_p to community t_q on predicate r_r, respectively:

$$\begin{aligned} \#^{c_{pqr}=1} &= \left| \left\{ g_{xyz} \in \mathbf{G} \ : \ \Psi(x, y, z) = c_{pqr} \wedge g_{xyz} = 1 \right\} \right| \\ \#^{c_{pqr}=0} &= \left| \left\{ g_{xyz} \in \mathbf{G} \ : \ \Psi(x, y, z) = c_{pqr} \wedge g_{xyz} = 0 \right\} \right| \end{aligned} \tag{6.5}$$

We can now derive a closed-form solution for the posterior of community relations by applying the distributions defined in our model:

$$\begin{aligned} \mathbb{P}(c_{pqr} \mid \mathbf{C}_{-(pqr)}, \mathbf{G}, &\mathbf{P}, \mathbf{Z}, \lambda, \eta) \\ &\overset{(1)}{=} \frac{\Big(\prod_{g_{xyz} \in \mathbf{G}} \mathrm{Bernoulli}(c_{pqr}, 1 - c_{pqr}) \Big) \Big(\mathrm{Beta}(\lambda, \eta) \Big)}{\int_{c_{pqr}} \Big(\prod_{g_{xyz} \in \mathbf{G}} \mathrm{Bernoulli}(c_{pqr}, 1 - c_{pqr}) \Big) \Big(\mathrm{Beta}(\lambda, \eta) \Big) \, \mathrm{d}c_{pqr}} \end{aligned}$$

$$\begin{aligned}
&\overset{(2)}{=} \frac{\left(c_{pqr}^{\#^{c_{pqr}=1}}(1-c_{pqr})^{\#^{c_{pqr}=0}}\right)\left(\frac{c_{pqr}^{\lambda-1}(1-c_{pqr})^{\eta-1}}{\mathrm{B}(\lambda,\eta)}\right)}{\displaystyle\int_{c_{pqr}}\left(c_{pqr}^{\#^{c_{pqr}=1}}(1-c_{pqr})^{\#^{c_{pqr}=0}}\right)\left(\frac{c_{pqr}^{\lambda-1}(1-c_{pqr})^{\eta-1}}{\mathrm{B}(\lambda,\eta)}\right)\,\mathrm{d}c_{pqr}} \\
&\overset{(3)}{=} \frac{c_{pqr}^{\#^{c_{pqr}=1}+\lambda-1}(1-c_{pqr})^{\#^{c_{pqr}=0}+\eta-1}}{\displaystyle\int_{c_{pqr}} c_{pqr}^{\#^{c_{pqr}=1}+\lambda-1}(1-c_{pqr})^{\#^{c_{pqr}=0}+\eta-1}\,\mathrm{d}c_{pqr}} \\
&\overset{(4)}{=} \frac{c_{pqr}^{\#^{c_{pqr}=1}+\lambda-1}(1-c_{pqr})^{\#^{c_{pqr}=0}+\eta-1}}{\mathrm{B}(\#^{c_{pqr}=1}+\lambda,\#^{c_{pqr}=0}+\eta)} \\
&= \mathrm{Beta}(\#^{c_{pqr}=1}+\lambda,\#^{c_{pqr}=0}+\eta) \qquad (6.6)
\end{aligned}$$

Such that in the derivation above: (1) is the Bayes' theorem definition as per Equation 6.4 using the probability distributions defined in our model; (2) uses the probability masses and densities of the Bernoulli and Beta distributions as per Equations A.1 and A.3; (3) is obtained by applying power rules and dividing out the Beta function which is constant with respect to c_{pqr} in the integral; and (4) utilizes the integral form of the Beta function as derived in Equation B.1. The posterior as defined in Equation 6.6 allows for community relations to be integrated out when sampling paths. As such they are not sampled directly in the inference process.

Marginalizing Level Memberships

There are two ways in which to approach marginalizing level memberships in our model. Firstly, Sethuraman [25] showed that the realization of the stick breaking process follows the Dirichlet distribution. We can leverage this because, in practice, the dimensionality of the level memberships gets bounded to the depth of the tree, L. It is therefore possible to model level memberships with an L dimensional Dirichlet distribution. As discussed in Ho et al. [16], this prior has the disadvantage of either being too expressive or not expressive enough depending on its parameterization. Regardless, we show the marginalization of this case in Appendix C. In this subsection, however, we focus on the infinite case using the stick breaking process as defined in our model. To this end, we use the multinomial-stick conjugacy to obtain a stick breaking posterior which is used as the prior for the level indicators later on. The posterior is defined as follows:

$$\begin{aligned}\mathbb{P}(\mathbf{a}_i \mid \mathbf{A}_{-i}, \mathbf{Z}, \mu, \sigma) &= \frac{\mathbb{P}(\mathbf{Z} \mid \mathbf{A}, \mu, \sigma)\mathbb{P}(\mathbf{a}_i \mid \mathbf{A}_{-i}, \mu, \sigma)}{\int_{\mathbf{a}_i} \mathbb{P}(\mathbf{Z} \mid \mathbf{A}, \mu, \sigma)\mathbb{P}(\mathbf{a}_i \mid \mathbf{A}_{-i}, \mu, \sigma)\ \mathrm{d}\mathbf{a}_i} \\ &= \frac{\text{Multinomial}(\mathbf{a}_i)\text{Stick}(\mu, \sigma)}{\int_{\mathbf{a}_i} \text{Multinomial}(\mathbf{a}_i)\text{Stick}(\mu, \sigma)\ \mathrm{d}\mathbf{a}_i}\end{aligned} \tag{6.7}$$

Where $\mathbb{P}(\mathbf{Z} \mid \mathbf{A}, \mu, \sigma)$ and $\mathbb{P}(\mathbf{a}_i \mid \mathbf{A}_{-i}, \mu, \sigma)$ are the likelihood and prior of level memberships, respectively. We use the definitions from our model and replace these with the multinomial and stick breaking distributions. Before proceeding, we define $\mathbf{z}_{i*} = \{z_{x\leftrightarrow y} \in \mathbf{Z} : x = i \vee y = i\}$ representing all level indicators for entity e_i. In this notation $z_{i\leftrightarrow j} := z_{i\rightarrow j} \vee z_{i\leftarrow j}$ is used as shorthand for any level indicator relating entity e_i with entity e_j regardless of which entities are taking on the sender and receiver roles. This allows for defining two helper variables, $\#^{\mathbf{z}_{i*}=l}$ and $\#^{\mathbf{z}_{i*}>l}$, to indicate the number of indicators in $\mathbf{z}_{i*}$ at and below level l in the hierarchy:

$$\begin{aligned}\#^{\mathbf{z}_{i*}=l} &= \left|\left\{z_{i\leftrightarrow j} \in \mathbf{z}_{i*} : z_{i\leftrightarrow j} = l\right\}\right| \\ \#^{\mathbf{z}_{i*}>l} &= \left|\left\{z_{i\leftrightarrow j} \in \mathbf{z}_{i*} : z_{i\leftrightarrow j} > l\right\}\right|\end{aligned} \tag{6.8}$$

With these definitions in place, we can derive the stick breaking posterior of level memberships:

$$\mathbb{P}(\mathbf{a}_i \mid \mathbf{A}_{-i}, \mathbf{Z}, \mu, \sigma)$$

$$\overset{(1)}{=} \frac{\left(\frac{\Gamma(\sum\limits_{l=1}^{\infty} \#^{\mathbf{z}_{i*}=l} + 1)}{\prod\limits_{l=1}^{\infty} \Gamma(\#^{\mathbf{z}_{i*}=l} + 1)} \prod\limits_{l=1}^{\infty} \mathbf{a}_i^{\#^{\mathbf{z}_{i*}=l}}\right)\left(\sum\limits_{l=1}^{\infty} v_l \prod\limits_{k=1}^{l-1}(1 - v_k)\mathbb{I}(a_i^l = l)\right)}{\int_{\mathbf{V}} \left(\frac{\Gamma(\sum\limits_{l=1}^{\infty} \#^{\mathbf{z}_{i*}=l} + 1)}{\prod\limits_{l=1}^{\infty} \Gamma(\#^{\mathbf{z}_{i*}=l} + 1)} \prod\limits_{l=1}^{\infty} \mathbf{a}_i^{\#^{\mathbf{z}_{i*}=l}}\right)\left(\sum\limits_{l=1}^{\infty} v_l \prod\limits_{k=1}^{l-1}(1 - v_k)\mathbb{I}(a_i^l = l)\right)\mathrm{d}^{\infty}\mathbf{V}}$$

$$\overset{(2)}{=} \frac{\left(\frac{\Gamma(\sum\limits_{l=1}^{\infty} \#^{\mathbf{z}_{i*}=l}+1)}{\prod\limits_{l=1}^{\infty}\Gamma(\#^{\mathbf{z}_{i*}=l}+1)} \prod\limits_{l=1}^{\infty}\left(v_l \prod\limits_{k=1}^{l-1}(1-v_k)\right)^{\#^{\mathbf{z}_{i*}=l}}\right)}{\int_{\mathbf{V}} \left(\frac{\Gamma(\sum\limits_{l=1}^{\infty} \#^{\mathbf{z}_{i*}=l}+1)}{\prod\limits_{l=1}^{\infty}\Gamma(\#^{\mathbf{z}_{i*}=l}+1)} \prod\limits_{l=1}^{\infty}\left(v_l \prod\limits_{k=1}^{l-1}(1-v_k)\right)^{\#^{\mathbf{z}_{i*}=l}}\right)}$$

$$\frac{\left(\mathrm{Beta}(\mu\sigma,(1-\mu)\sigma)\prod\limits_{k=1}^{l-1}\mathrm{Beta}((1-\mu)\sigma,\mu\sigma)\right)}{\left(\mathrm{Beta}(\mu\sigma,(1-\mu)\sigma)\prod\limits_{k=1}^{l-1}\mathrm{Beta}((1-\mu)\sigma,\mu\sigma)\right)\mathrm{d}^{\infty}\mathbf{V}}$$

$$\overset{(3)}{=} \frac{\left(\frac{\Gamma(\sum\limits_{l=1}^{\infty} \#^{\mathbf{z}_{i*}=l}+1)}{\prod\limits_{l=1}^{\infty}\Gamma(\#^{\mathbf{z}_{i*}=l}+1)} v_l^{\#^{\mathbf{z}_{i*}=l}}(1-v_l)^{\#^{\mathbf{z}_{i*}>l}} \prod\limits_{k=1}^{l-1} v_k^{\#^{\mathbf{z}_{i*}>k}}(1-v_k)^{\#^{\mathbf{z}_{i*}=k}}\right)}{\int_{\mathbf{V}} \left(\frac{\Gamma(\sum\limits_{l=1}^{\infty} \#^{\mathbf{z}_{i*}=l}+1)}{\prod\limits_{l=1}^{\infty}\Gamma(\#^{\mathbf{z}_{i*}=l}+1)} v_l^{\#^{\mathbf{z}_{i*}=l}}(1-v_l)^{\#^{\mathbf{z}_{i*}>l}} \prod\limits_{k=1}^{l-1} v_k^{\#^{\mathbf{z}_{i*}>k}}(1-v_k)^{\#^{\mathbf{z}_{i*}=k}}\right)}$$

$$\frac{\left(\frac{v_l^{\mu\sigma-1}(1-v_l)^{(1-\mu)\sigma-1}}{\mathrm{B}(\mu\sigma,(1-\mu)\sigma)}\prod\limits_{k=1}^{l-1}\frac{v_k^{(1-\mu)\sigma-1}(1-v_k)^{\mu\sigma-1}}{\mathrm{B}((1-\mu)\sigma,\mu\sigma)}\right)}{\left(\frac{v_l^{\mu\sigma-1}(1-v_l)^{(1-\mu)\sigma-1}}{\mathrm{B}(\mu\sigma,(1-\mu)\sigma)}\prod\limits_{k=1}^{l-1}\frac{v_k^{(1-\mu)\sigma-1}(1-v_k)^{\mu\sigma-1}}{\mathrm{B}((1-\mu)\sigma,\mu\sigma)}\right)\mathrm{d}^{\infty}\mathbf{V}}$$

$$\overset{(4)}{=} \frac{v_l^{\#^{\mathbf{z}_{i*}=l}+\mu\sigma-1}(1-v_l)^{\#^{\mathbf{z}_{i*}>l}+(1-\mu)\sigma-1}\prod\limits_{k=1}^{l-1}v_k^{\#^{\mathbf{z}_{i*}>k}+(1-\mu)\sigma-1}(1-v_k)^{\#^{\mathbf{z}_{k*}=k}+\mu\sigma-1}}{\int_{\mathbf{V}} v_l^{\#^{\mathbf{z}_{i*}=l}+\mu\sigma-1}(1-v_l)^{\#^{\mathbf{z}_{i*}>l}+(1-\mu)\sigma-1}\prod\limits_{k=1}^{l-1}v_k^{\#^{\mathbf{z}_{i*}>k}+(1-\mu)\sigma-1}(1-v_k)^{\#^{\mathbf{z}_{k*}=k}+\mu\sigma-1}\mathrm{d}^{\infty}\mathbf{V}}$$

$$\overset{(5)}{=} \frac{v_l^{\#^{\mathbf{z}_{i*}=l}+\mu\sigma-1}(1-v_l)^{\#^{\mathbf{z}_{i*}>l}+(1-\mu)\sigma-1}\prod\limits_{k=1}^{l-1}v_k^{\#^{\mathbf{z}_{i*}>k}+(1-\mu)\sigma-1}(1-v_k)^{\#^{\mathbf{z}_{k*}=k}+\mu\sigma-1}}{\mathrm{B}(\#^{\mathbf{z}_{i*}=l}+\mu\sigma,\#^{\mathbf{z}_{i*}>l}+(1-\mu)\sigma)\prod\limits_{k=1}^{l-1}\mathrm{B}(\#^{\mathbf{z}_{i*}=k}+(1-\mu)\sigma,\#^{\mathbf{z}_{k*}=k}+\mu\sigma-1)}$$

$$= \mathrm{Beta}(\#^{\mathbf{z}_{i*}=l}+\mu\sigma,\#^{\mathbf{z}_{i*}>l}+(1-\mu)\sigma)\prod_{k=1}^{l-1}\mathrm{Beta}(\#^{\mathbf{z}_{i*}>k}+(1-\mu)\sigma,\#^{\mathbf{z}_{i*}=k}+\mu\sigma)$$

$$= \sum_{l=1}^{\infty} v_l \prod_{k=1}^{l-1}(1-v_k) \mid \mathbf{z}_{i*}$$

$$= \mathrm{Stick}(\mu\sigma,(1-\mu)\sigma) \mid \mathbf{z}_{i*} \tag{6.9}$$

Where (1) is an application of the definitions of the Multinomial and stick breaking distributions as per Equations A.2 and 2.8 to the posterior as per Equation 6.7. (2) redefines the likelihood in terms of the Beta samples of the stick breaking process and the prior leverages the mirror symmetry property of the Beta function. The summation is removed at this stage to enhance readability. (3) rearranges the likelihood and substitutes the probability density function of the Beta distribution as per Equation A.3. (4) involves cancelling out terms in the numerator and denominator which a constant with respect to the integration. (5) utilizes the integral form of the Beta function as per Equation B.1 and the remainder of the derivation merely reverses the definitions applied earlier to arrive at the definition of the stick breaking process. We use the notation $\mid \mathbf{z}_{i*}$ to denote that the distribution preceding the notation is conditioned on $\mathbf{z}_{i*}$.

Sampling Entity Paths

Entity paths are one of the two variables which remain after collapsing the Gibbs sampler and must therefore be sampled directly. To sample a path for entity e_i, it must first be removed from the hierarchy, thereby allowing for its full conditional distribution to be obtained. The set of paths after having removed path $\mathbf{p}_i$ is denoted as $\mathbf{P}_{-i}$. We derive the posterior distribution of $\mathbf{p}_i$ by applying Bayes' theorem:

$$\begin{aligned}\mathbb{P}(\mathbf{p}_i \mid \mathbf{P}_{-i}, \mathbf{G}, \mathbf{Z}, \gamma, \lambda, \eta) &= \frac{\mathbb{P}(\mathbf{G}_{i*} \mid \mathbf{G}_{-(i*)}, \mathbf{P}, \mathbf{Z}, \gamma, \lambda, \eta)\mathbb{P}(\mathbf{p}_i \mid \mathbf{P}_{-i}, \gamma)}{\int_{\mathbf{p}_i} \mathbb{P}(\mathbf{G}_{i*} \mid \mathbf{G}_{-(i*)}, \mathbf{P}, \mathbf{Z}, \gamma, \lambda, \eta)\mathbb{P}(\mathbf{p}_i \mid \mathbf{P}_{-i}, \gamma)\, \mathrm{d}\mathbf{p}_i} \\ &\propto \mathbb{P}(\mathbf{G}_{i*} \mid \mathbf{G}_{-(i*)}, \mathbf{P}, \mathbf{Z}, \gamma, \lambda, \eta)\mathbb{P}(\mathbf{p}_i \mid \mathbf{P}_{-i}, \gamma) \qquad (6.10)\end{aligned}$$

Where $\mathbf{G}_{i*} = \{g_{xyz} \in \mathbf{G} : i = x \ \vee \ i = y\}$ denotes all the triples in the knowledge graph that depend on path $\mathbf{p}_i$ and $\mathbf{G}_{-(i*)} = \mathbf{G} \setminus \mathbf{G}_{i*}$ is its complement. The integral form of the marginal distribution for generating the data is a normalizing constant for the posterior distribution. Calculating this integral is not necessary and we can instead sample paths from its proportional distribution as per Equation 6.10. The prior for sampling an entity path, $\mathbb{P}(\mathbf{p}_i \mid \mathbf{P}_{-i}, \gamma)$, is obtained from the nCRP. We note that due to the iterative nature of the Gibbs sampler, a path for entity e_i may already exist in the hierarchy from a previous iteration. As such, it must first be removed, hence the conditioning on $\mathbf{P}_{-i}$. We use $\mathbf{p}_i = t_q$ as a shorthand to indicate the path that terminates at community t_q, in other words $\mathbf{p}_i = t_q$ if $p_i^L = t_q$. Thus, the prior is calculated as follows:

$$\begin{aligned}\mathbb{P}(\mathbf{p}_i = t_q \mid \mathbf{P}_{-i}, \gamma) &= \mathbb{P}(p_i^L = t_q \mid \mathbf{P}_{-i}, \gamma) \\ &= \mathbb{E}\Big[(\mathbb{I}(p_i^L = t_q) \;\Big|\; \mathbf{P}_{-i}, \gamma)\Big] \\ &= \mathbb{P}(p_i^L = t_q \mid \mathbf{p}_i^{1\,:\,L-1}, \mathbf{P}_{-i}, \gamma) \prod_{l=1}^{L-1} \mathbb{P}(p_i^l = t_q^l \mid \mathbf{p}_i^{1\,:\,l-1}, \mathbf{P}_{-i}, \gamma)\end{aligned} \tag{6.11}$$

Where t_q^l is the ancestor community of community t_q at level l. Equation 6.11 requires the distribution for sampling a community conditioned on a partially sampled path. Recall that this is defined by the nCRP in Equation 2.6 and can be adapted here. Specifically, we calculate the probability of taking community t_q on level l having already sampled its path up to level $l-1$ as:

$$\mathbb{P}(p_i^l = t_q \mid \mathbf{p}_i^{1\,:\,l-1}, \mathbf{P}_{-i}, \gamma) = \begin{cases} \dfrac{\#_{-i}^{t_q}}{\#_{-i}^{t_q^{l-1}} + \gamma} & t_q \in \mathcal{T}_{-i} \\ \dfrac{\gamma}{\#_{-i}^{t_q^{l-1}} + \gamma} & t_q \notin \mathcal{T}_{-i} \end{cases} \tag{6.12}$$

Where $\#_{-i}^{t_q}$ extends the notation defined earlier to indicate the number of entities that have gone through community t_q in the hierarchy with path $\mathbf{p}_i$ removed. $\mathcal{T}_{-i}$ indicates all the communities in the hierarchy after path $\mathbf{p}_i$ has been removed. We note that this process requires the sampling of a path to start at the root community and proceed sequentially to a leaf community. Having obtained the prior, it's necessary to update the belief about the posterior with the data via the likelihood. The likelihood given a sampled path, $\mathbb{P}(\mathbf{G}_{i*} \mid \mathbf{G}_{-(i*)}, \mathbf{P}, \mathbf{Z}, \gamma, \lambda, \eta)$, is defined with the help of the following helper variables:

$$\mathbf{C}_{i*} = \Big\{c_{pqr} \in \mathbf{C} \;:\; (\exists g_{xyz} \in \mathbf{G}_{i*} : \Psi(x, y, z) = c_{pqr})\Big\}$$

$$
\begin{aligned}
\#_{-i}^{c_{pqr}=1} &= \left|\left\{g_{xyz} \in \mathbf{G}_{-(i*)} : \Psi(x,y,z) = c_{pqr} \wedge g_{xyz} = 1\right\}\right| \\
\#_{-i}^{c_{pqr}=0} &= \left|\left\{g_{xyz} \in \mathbf{G}_{-(i*)} : \Psi(x,y,z) = c_{pqr} \wedge g_{xyz} = 0\right\}\right| \\
\#_{i}^{c_{pqr}=1} &= \left|\left\{g_{xyz} \in \mathbf{G}_{i*} : \Psi(x,y,z) = c_{pqr} \wedge g_{xyz} = 1\right\}\right| \\
\#_{i}^{c_{pqr}=0} &= \left|\left\{g_{xyz} \in \mathbf{G}_{i*} : \Psi(x,y,z) = c_{pqr} \wedge g_{xyz} = 0\right\}\right| \qquad (6.13)
\end{aligned}
$$

The definitions above capture the following: $\mathbf{C}_{i*}$ is the set of communities dependant on a relation in $\mathbf{G}_{i*}$; $\#_{-i}^{c_{pqr}=1}$ and $\#_{i}^{c_{pqr}=1}$ are the counts of existing entity relations from communities p to q in $\mathbf{G}_{-(i)}$ and $\mathbf{G}_{i}$, respectively; and $\#_{-i}^{c_{pqr}=0}$ and $\#_{i}^{c_{pqr}=0}$ are the counts of non-existing entity relations from communities p to q in $\mathbf{G}_{-(i)}$ and $\mathbf{G}_{i}$, respectively. In the discrete space, the likelihood is understood as the joint probability of generating the data as per a probability mass function. In our model, the data is obtained by drawing from the Bernoulli distribution conditioned on community relations. Recall that these parameters are marginalized out of our model and thus never sampled directly. As such, in order to calculate the likelihood we must integrate with respect to the community relations. This is possible by leveraging Equation 6.6 which allows us to obtain a closed form solution:

$$
\begin{aligned}
&\mathbb{P}(\mathbf{G}_{i*} \mid \mathbf{G}_{-(i*)}, \mathbf{P}, \mathbf{Z}, \gamma, \lambda, \eta) \\
&= \prod_{c_{pqr} \in \mathbf{C}_{i*}} \int_{c_{pqr}} \mathbb{P}(\mathbf{G}_{i*} \mid \mathbf{G}_{-(i*)}, \mathbf{C}, \mathbf{P}, \mathbf{Z}, \gamma, \mu, \sigma, \lambda, \eta) \\
&\qquad\qquad \mathbb{P}(c_{pqr} \mid \mathbf{C}_{-(pqr)}, \mathbf{G}_{-(i*)}, \mathbf{P}, \mathbf{Z}, \lambda, \eta)\ \mathrm{d}c_{pqr} \\
&\overset{(1)}{=} \prod_{c_{pqr} \in \mathbf{C}_{i*}} \int_{c_{pqr}} \left(\prod_{g_{xyz} \in \mathbf{G}_{i*}} \mathrm{Bernoulli}(c_{pqr}, 1 - c_{pqr}) \right) \\
&\qquad\qquad \left(\mathrm{Beta}(\#_{-i}^{c_{pqr}=1} + \lambda, \#_{-i}^{c_{pqr}=0} + \eta) \right) \mathrm{d}c_{pqr} \\
&\overset{(2)}{=} \prod_{c_{pqr} \in \mathbf{C}_{i*}} \int_{c_{pqr}} \left(c_{pqr}^{\#_{i}^{c_{pqr}=1}} (1 - c_{pqr})^{\#_{i}^{c_{pqr}=0}} \right) \left(\frac{c_{pqr}^{\#_{-i}^{c_{pqr}=1} + \lambda - 1} (1 - c_{pqr})^{\#_{-i}^{c_{pqr}=0} + \eta - 1}}{\mathrm{B}(\#_{-i}^{c_{pqr}=1} + \lambda, \#_{-i}^{c_{pqr}=0} + \eta)} \right) \mathrm{d}c_{pqr} \\
&= \prod_{c_{pqr} \in \mathbf{C}_{i*}} \frac{1}{\mathrm{B}(\#_{-i}^{c_{pqr}=1} + \lambda, \#_{-i}^{c_{pqr}=0} + \eta)}
\end{aligned}
$$

$$
\begin{aligned}
& \int_{c_{pqr}} c_{pqr}^{\#_i^{c_{pqr}=1}+\#_{-i}^{c_{pqr}=1}+\lambda-1}(1-c_{pqr})^{\#_i^{c_{pqr}=0}+\#_{-i}^{c_{pqr}=0}+\eta-1}\,\mathrm{d}c_{pqr} \\
\overset{(3)}{=} & \prod_{c_{pqr}\in\mathbf{C}_{i*}} \frac{\mathrm{B}(\#_i^{c_{pqr}=1}+\#_{-i}^{c_{pqr}=1}+\lambda, \#_i^{c_{pqr}=0}+\#_{-i}^{c_{pqr}=0}+\eta)}{\mathrm{B}(\#_{-i}^{c_{pqr}=1}+\lambda, \#_{-i}^{c_{pqr}=0}+\eta)} \\
\overset{(4)}{=} & \prod_{c_{pqr}\in\mathbf{C}_{i*}} \left(\frac{\Gamma(\#_i^{c_{pqr}=1}+\#_{-i}^{c_{pqr}=1}+\lambda)\Gamma(\#_i^{c_{pqr}=0}+\#_{-i}^{c_{pqr}=0}+\eta)}{\Gamma(\#_i^{c_{pqr}=1}+\#_{-i}^{c_{pqr}=1}+\#_i^{c_{pqr}=0}+\#_{-i}^{c_{pqr}=0}+\lambda+\eta)}\right) \\
& \qquad \left(\frac{\Gamma(\#_{-i}^{c_{pqr}=1}+\#_{-i}^{c_{pqr}=0}+\lambda+\eta)}{\Gamma(\#_{-i}^{c_{pqr}=1}+\lambda)\Gamma(\#_{-i}^{c_{pqr}=0}+\eta)}\right)
\end{aligned}
\tag{6.14}
$$

In the derivation above: (1) the prior probability of drawing c_{pqr} is obtained from Equation 6.6; (2) utilizes the definitions as per Equations A.1 and A.3 as well as the helper variables introduced in Equation 6.13; (3) leverages the integral form of the Beta function as per Equation B.1; and (4) expands the Beta function to its Gamma formulation as per Equation A.3. Having derived the prior and likelihood in Equations 6.11 and 6.14, respectively, it is possible to sample from Equation 6.10 to obtain entity paths in our model. The time complexity of sampling one such path is $\mathcal{O}(|\mathcal{E}|^2|\mathcal{R}|L)$. This is due to the fact that it's necessary to obtain a sampling probability for all potential paths in the hierarchy, which has a bound of $|\mathcal{E}|L$ in the case where each entity takes a unique path. For each of these potential paths, the iteration through all $|\mathcal{E}|$ entities and $|\mathcal{R}|$ predicates is required to determine the effect on the likelihood selecting such a path would have.

Sampling Level Indicators

Level indicators are drawn from the multinomial distribution conditioned on level memberships. Recall that level memberships were marginalized over in our inference scheme using the multinomial-stick conjugacy and are thus never sampled directly. Nevertheless, we draw them indirectly when computing the prior for level indicators. As with sampling paths, we obtain the distribution proportional to that of level indicators by Bayes' rule. In what follows, we provide the derivation for the posterior of $z_{i\to j}$ and note that given its structural symmetry, $z_{i\leftarrow j}$ is derived analogously. The posterior distribution of $z_{i\to j}$ is expressed as:

$$\begin{aligned}
\mathbb{P}(z_{i\to j} \mid \mathbf{Z}_{-(i\to j)},&\mathbf{G}, \mathbf{P}, \gamma, \mu, \sigma, \lambda, \eta) \\
&= \frac{\mathbb{P}(\mathbf{g}_{ij*} \mid \mathbf{G}_{-(ij*)}, \mathbf{C}, \mathbf{P}, \mathbf{Z}, \lambda, \eta)\mathbb{P}(z_{i\to j} \mid \mathbf{Z}_{-(i\to j)}, \mu, \sigma)}{\int_{z_{i\to j}} \mathbb{P}(\mathbf{g}_{ij*} \mid \mathbf{G}_{-(ij*)}, \mathbf{C}, \mathbf{P}, \mathbf{Z}, \lambda, \eta)\mathbb{P}(z_{i\to j} \mid \mathbf{Z}_{-(i\to j)}, \mu, \sigma) \;\; \mathrm{d}z_{i\to j}} \\
&\propto \mathbb{P}(\mathbf{g}_{ij*} \mid \mathbf{G}_{-(ij*)}, \mathbf{P}, \mathbf{Z}, \lambda, \eta)\mathbb{P}(z_{i\to j} \mid \mathbf{Z}_{-(i\to j)}, \mu, \sigma) \qquad (6.15)
\end{aligned}$$

Where $\mathbf{g}_{ij*} = \{g_{xyz} \in \mathbf{G} : i = x \wedge j = y\}$ denotes the vector of relations in $\mathbf{G}$ from entity e_i to e_j across all predicates, $\mathbf{G}_{-(ij*)} = \mathbf{G} \setminus \mathbf{g}_{ij*}$ is its complement, and $\mathbf{Z}_{-(i\to j)}$ is all the level indicators excluding $z_{i\to j}$. The prior probability for sampling levels, $\mathbb{P}(z_{i\to j} \mid \mathbf{Z}_{-(i\to j)}, \mu, \sigma)$, is drawn from the derived posterior of level memberships in Equation 6.9 . We follow Blei et al. [24] and use the law of total expectations to obtain the probability of $z_{i\to j}$ realizing level l as the expectation of the size of the stick broken off at the l^{th} break. To do this we define two variables which may be seen as the directed extensions of those introduced in Equation 6.8:

$$\begin{aligned}
\#^{\mathbf{Z}_{-(i\to j)}=l} &= \left|\left\{ z_{i\to j} \in \mathbf{Z}_{-(i\to j)} : z_{i\to j} = l \right\}\right| \\
\#^{\mathbf{Z}_{-(i\to j)}>l} &= \left|\left\{ z_{i\to j} \in \mathbf{Z}_{-(i\to j)} : z_{i\to j} > l \right\}\right| \qquad (6.16)
\end{aligned}$$

With these variables in place, we obtain the prior distribution as follows:

$$\begin{aligned}
\mathbb{P}(z_{i\to j} = l \mid \mathbf{Z}_{-(i\to j)},&\mathbf{G}_{-(ij*)}, \mathbf{P}, \mu, \sigma) \\
&= \mathbb{E}\Big[\mathbb{I}(z_{i\to j} = l) \;\Big|\; \mathbf{Z}_{-(i\to j)}, \mu, \sigma\Big] \\
&\overset{(1)}{=} \mathbb{E}\left[\mathbb{E}\left[\mathbb{I}(z_{i\to j} = l) \;\Big|\; v_1, v_2, \ldots, v_l, \mathbf{Z}_{-(i\to j)}, \mu, \sigma\right]\right] \\
&\overset{(2)}{=} \mathbb{E}\left[\sum_{m=1}^{\infty} v_l \prod_{k=1}^{l-1}(1 - v_k)\mathbb{I}(m = l) \;\Big|\; \mathbf{Z}_{-(i\to j)}, \mu, \sigma\right] \\
&= \mathbb{E}\Big[v^l \mid \mathbf{Z}_{-(i\to j)}, \mu, \sigma\Big] \prod_{k=1}^{l-1} \mathbb{E}\Big[1 - v^k \mid \mathbf{Z}_{-(i\to j)}, \mu, \sigma\Big] \\
&\overset{(3)}{=} \frac{\mu\sigma + \#^{\mathbf{Z}_{-(i\to j)}=l}}{\sigma + \#^{\mathbf{Z}_{-(i\to j)}=l} + \#^{\mathbf{Z}_{-(i\to j)}>l}} \prod_{k=1}^{l-1} \frac{(1-\mu)\sigma + \#^{\mathbf{Z}_{-(i\to j)}>k}}{\sigma + \#^{\mathbf{Z}_{-(i\to j)}=k} + \#^{\mathbf{Z}_{-(i\to j)}>k}} \\
&\qquad (6.17)
\end{aligned}$$

Where (1) is derived by the application of the law of total expectation; (2) is obtained from the probability of drawing level l from the stick breaking process conditioned on the successive draws from the Beta distribution, denoted $v_1, v_2, \ldots, v_l$, as per Equation 2.8; and (3) is the expected value of drawing from the Beta distribution conditioned on $\mathbf{Z}_{-(i \to j)}$ as per the level membership posterior obtained in Equation 6.9. The likelihood, $\mathbb{P}(\mathbf{g}_{ij*} \mid \mathbf{G}_{-(ij*)}, \mathbf{P}, \mathbf{Z}, \lambda, \eta)$, is obtained analogously to entity paths. To aid in this derivation, we define two constants as follows:

$$\begin{aligned}
\#^{c_{pqr}=1}_{-(ijr)} &= \left|\left\{ g_{xyz} \in \mathbf{G}_{-(ijr)} \ : \ \Psi(x,y,z) = c_{pqr} \wedge g_{xyz} = 1 \right\}\right| \\
\#^{c_{pqr}=0}_{-(ijr)} &= \left|\left\{ g_{xyz} \in \mathbf{G}_{-(ijr)} \ : \ \Psi(x,y,z) = c_{pqr} \wedge g_{xyz} = 0 \right\}\right|
\end{aligned} \tag{6.18}$$

Such that $\#^{c_{pqr}=1}_{-(ijr)}$ and $\#^{c_{pqr}=0}_{-(ijr)}$ capture the number of existing and non-existing relations from communities t_p to t_q not including g_{ijr}. With these in place, we can derive the level indicator likelihood. This process is analogous to the one for entity paths in that we use the Bernoulli distribution for model output and integrate over community relations:

$$\begin{aligned}
&\mathbb{P}(\mathbf{g}_{ij*} \mid \mathbf{G}_{-(ij*)}, \mathbf{C}, \mathbf{P}, \mathbf{Z}, \lambda, \eta) \\
&= \int_{c_{pqr}} \mathbb{P}(\mathbf{g}_{ij*} \mid \mathbf{G}_{-(ij*)}, \mathbf{C}, \mathbf{P}, \mathbf{Z}, \gamma, \mu, \sigma, \lambda, \eta) \mathbb{P}(c_{pqr} \mid \mathbf{C}_{-(pqr)}, \mathbf{G}_{-(ij*)}, \mathbf{P}, \mathbf{Z}, \lambda, \eta) \, \mathrm{d}c_{pqr} \\
&\overset{(1)}{=} \int_{c_{pqr}} \prod_{g_{ijr} \in \mathbf{g}_{ij*}} \mathbb{P}(g_{ijr} \mid \mathbf{G}_{-(ijr)}, \mathbf{C}, \mathbf{P}, \mathbf{Z}, \gamma, \mu, \sigma, \lambda, \eta) \\
&\qquad\qquad \mathbb{P}(c_{pqr} \mid \mathbf{C}_{-(pqr)}, \mathbf{G}_{-(ijr)}, \mathbf{P}, \mathbf{Z}, \lambda, \eta) \, \mathrm{d}c_{pqr} \\
&\overset{(2)}{=} \int_{c_{pqr}} \prod_{g_{ijr} \in \mathbf{g}_{ij*}} \left(\mathrm{Bernoulli}(c_{pqr}, 1 - c_{pqr}) \right) \left(\mathrm{Beta}(\#^{c_{pqr}=1}_{-(ijr)} + \lambda, \#^{c_{pqr}=0}_{-(ijr)} + \eta) \right) \mathrm{d}c_{pqr} \\
&= \int_{c_{pqr}} \prod_{g_{ijr} \in \mathbf{g}_{ij*}} \left(c_{pqr}^{g_{ijr}} (1 - c_{pqr})^{1 - g_{ijr}} \right) \left(\frac{c_{pqr}^{\#^{c_{pqr}=1}_{-(ijr)} + \lambda - 1} (1 - c_{pqr})^{\#^{c_{pqr}=0}_{-(ijr)} + \eta - 1}}{\mathrm{B}(\#^{c_{pqr}=1}_{-(ijr)} + \lambda, \#^{c_{pqr}=0}_{-(ijr)} + \eta)} \right) \mathrm{d}c_{pqr} \\
&= \prod_{g_{ijr} \in \mathbf{g}_{ij*}} \frac{1}{\mathrm{B}(\#^{c_{pqr}=1}_{-(ijr)} + \lambda, \#^{c_{pqr}=0}_{-(ijr)} + \eta)} \\
&\qquad\qquad \int_{c_{pqr}} c_{pqr}^{g_{ijr} + \#^{c_{pqr}=1}_{-(ijr)} + \lambda - 1} (1 - c_{pqr})^{(1 - g_{ijr}) + \#^{c_{pqr}=0}_{-(ijr)} + \eta - 1} \, \mathrm{d}c_{pqr}
\end{aligned}$$

$$\begin{aligned}
&\overset{(3)}{=} \prod_{g_{ijr} \in \mathbf{g}_{ij*}} \frac{\mathrm{B}(\#^{c_{pqr}=1}_{-(ijr)} + g_{ijr} + \lambda, \#^{c_{pqr}=0}_{-(ijr)} + (1 - g_{ijr}) + \eta)}{\mathrm{B}(\#^{c_{pqr}=1}_{-(ijr)} + \lambda, \#^{c_{pqr}=0}_{-(ijr)} + \eta)} \\
&\overset{(4)}{=} \prod_{g_{ijr} \in \mathbf{g}_{ij*}} \frac{\Gamma(\#^{c_{pqr}=1}_{-(ijr)} + g_{ijr} + \lambda)\Gamma(\#^{c_{pqr}=0}_{-(ijr)} + (1 - g_{ijr}) + \eta)\Gamma(\#^{c_{pqr}=1}_{-(ijr)} + \#^{c_{pqr}=0}_{-(ijr)} + \lambda + \eta)}{\Gamma(\#^{c_{pqr}=1}_{-(ijr)} + \#^{c_{pqr}=0}_{-(ijr)} + 1 + \lambda + \eta)\Gamma(\#^{c_{pqr}=1}_{-(ijr)} + \lambda)\Gamma(\#^{c_{pqr}=0}_{-(ijr)} + \eta)} \\
&\overset{(5)}{=} \prod_{g_{ijr} \in \mathbf{g}_{ij*}} \frac{g_{ijr}(\#^{c_{pqr}=1}_{-(ijr)} + \lambda) + (1 - g_{ijr})(\#^{c_{pqr}=0}_{-(ijr)} + \eta)}{\#^{c_{pqr}=1}_{-(ijr)} + \#^{c_{pqr}=0}_{-(ijr)} + \lambda + \eta} \qquad (6.19)
\end{aligned}$$

Where (1) applies the chain rule of probability; (2) utilizes the prior for c_{pqr} obtained in Equation 6.6; (3) and (4) leverage the integral and Gamma forms the Beta function as per Equations B.1 and A.3, respectively; and (5) simplifies the preceding equation for computational reason by eliminating the Gamma function as shown in Appendix D. With the prior and likelihood derived in closed form as per Equations 6.17 and 6.19, respectively, it's possible to sample level indicators via Equation 6.15. The time complexity of sampling a level indicator is $\mathcal{O}(|\mathcal{R}|L)$ due to the $|\mathcal{R}|$ calculations that need to be performed at each of the L levels in the hierarchy.

6.4.5 Sampling Procedure

Having marginalized out community relations and level memberships as well as derived the sampling equations for entity paths and level indicators, it is possible it perform collapsed Gibbs sampling by iteratively sampling from the remaining variables' full conditional distributions. This process has a time complexity of $\mathcal{O}(|\mathcal{E}|^2|\mathcal{R}|L + |\mathcal{E}|^3|\mathcal{R}|L)$ for each iteration of the sampler where the former and latter terms are derived from the sampling complexities of the level indicators and entity paths, respectively. This makes the inference scheme infeasible for large-scale datasets. We respond to this issue by modifying one of the characteristics of collapsed Gibbs sampling, namely that samples are obtained in equal proportions. In its original formulation, one iteration of the sampler samples $|\mathcal{E}|^2$ level indicators and $|\mathcal{E}|$ entity paths. One of the assumptions underlying this process is that the relative importance of all samples is the same. Such an assumption may be ill-adapted for knowledge graphs which are oftentimes sparse in their adjacency tensors and whose entities exhibit highly imbalanced relation densities. In this regard, the placement of highly connected entities will have a disproportionate effect on model likelihood and therefore the induced hierarchy as well. Preferentially sampling these entities may result in faster arrival at a distribution from which we can obtain output samples. Consider, for instance, a knowledge graph with the entities `Thing` and `Henry Ford`. Assuming that `Thing` has a higher relation density than `Henry Ford`, its proper placement in the hierarchy may be more critical for model output than `Henry Ford`. With this in mind, we propose a stochastic sampling scheme in which samples are drawn for an entity in proportion to their probability of interacting with other entities. Specifically, we introduce a sampling probability, denoted s_i for entity e_i, which specifies the chance of sampling a variable for the corresponding entity in an iteration of the collapsed Gibbs sampler. This probability is calculated for each entity as the fraction of entities in the knowledge graph which have fewer relations than itself. Such as formulation ensures that $1 \geq s_i > 0$ which allows s_i to serve as the parameter of a Bernoulli distribution to indicate whether a variable will get sampled in the current iteration of the Gibbs sampler.

After the Gibbs sampler has been burned in, it necessary to obtain final samples to obtain a hierarchical clustering. We take multiple samples to account for the spread in the posterior distribution. A consequence of this is that samples may differ and need to be aggregated to produce a final result. In this regard, we take the mode over the final samples to arrive at a final hierarchy. The Gibbs sampling procedure is summarized in Algorithm 5.

6.5 Evaluation

The evaluation of our model is split into two parts: quantitative and qualitative. The quantitative evaluation provides objective measures of model performance whereas the qualitative evaluation assesses our model through illustrations and subjective

Algorithm 5 Collapsed Gibbs Sampling Procedure for Model Inference

```
Input: Knowledge graph adjacency tensor, $\mathbf{G}$; model hyperparameters, $\gamma$, $\mu$, $\sigma$, $\lambda$, and $\eta$; number of iterations, $iters$
Output: Paths $\mathbf{P}$; level indicators $\mathbf{Z}$;
community relations $\mathbf{C}$
Initialize level indicators using Equation 6.17
Initialize paths using Equation 6.11
for $iter = 1, 2, ..., iters$ do
    Update level indicators using Equation 6.15 if Bernoulli($s_i$)
    Update paths using Equation 6.10 if Bernoulli($s_i$)
end for
Obtain final level indicators using 6.15
Obtain final paths using 6.10
```

analysis of the results. For both types of evaluations, our model first had to be inferred before final samples could be drawn. In this regard, we trained our model on three datasets using 200 burn-in samples using hyperparameters chosen by assessing the model's log likelihood. After burn-in, ten final samples were obtained by discarding all but the third of successive samples to account for autocorrelation between samples. All models we trained to a depth of $L = 4$. Furthermore, the model was trained five times for each dataset to account for stochasticity in the inference process.

6.5.1 Datasets

Our model was evaluated on three datasets: Synthetic Binary Tree, FB15k-237, and WikiData. A summary of these datasets' statistics is provided in Table 6.1. What follows is a brief description of each dataset as well as how it was generated.

Synthetic Binary Tree

The Synthetic Binary Tree (SBT) dataset was synthetically generated to capture our model's ability to separate communities at the lowest level in the hierarchy. The generative process first constructed a binary tree with a depth of four, assigned entities to communities, and sampled relations for each entity pair. All entities were assigned uniformly to communities on the lowest level of the hierarchy, resulting in 25 entities per leaf community. The sampling probability for each entity pair was determined by the level of their lowest common ancestor. Specifically, sampling probabilities of 0, 0.1, 0.4, and 0.6 were used for levels 0, 1, 2, and 3, respectively. Two entities belonging to the same community have a sampling probability of 1 and are thus always related. The dataset was generated for two predicates which shared the aforementioned sampling probabilities. We note that even though these probabilities are identical, they do not result in a dataset in which entity relations are identical across predicates. The generative process yielded a dataset of 55880 triples, 400 entities, and 2 predicates.

FB15k-237

The FB15k-237 dataset [130] is a subset of the FB15k dataset [96], created by removing redundant and inverse triples. The original FB15k dataset is in turn a subset of a 2013 version of Freebase, from which triples were queried. The FB15k-237 dataset is comprised of 272115 triples, 14541 entities, and 237 predicates thus presenting a computation challenge to our model if modelled in whole. To address this issue, we generated a subset of the data and derived ground truth community labels in an approach insipred by Jain et al. [131]. Specifically, entities were mapped to the WordNet taxonomy [77] through the *sameAs* predicate, which relates entities from Freebase and YAGO. Triples were then extracted to contain subjects from the sets provided in Zhang et al [17]. This process yielded a subset of the data containing 103550 triples, 10018 entities, and 190 predicates. Finally, the subset was reduced even further by extracting only the triples relating to footballers, pianists, journalists, politicians, and scientists as per the identifiers `/m/05vyk`, `/m/06q2q`, `/m/0gl2ny2`, `/m/0fj9f`, `/m/0d8qb` on the predicate `/people/person/profession`. This final step resulted in a dataset with 2499 triples, 1142 entities, and 79 predicates.

Dataset	# Triples	# Entities	# Predicates
SBT	55880	400	2
FB15k-237	2499	1142	79
WikiData	2525	716	6

Table 6.1: Summary of the datasets used in this chapter.

WikiData

The WikiData dataset was generated by querying Wikidata for triples relating to people and locations. Specifically, artists and footballers corresponding to WikiData identifiers `wd:Q1028181` and `wd:Q937857` respectively were extracted. These entities were then filtered to having been born in cities in four countries: Germany, the United Kingdom, Canada, and the United States of America. Furthermore, the knowledge graph was reduced to the following predicates: instance of, place of birth, citizen of, occupation, country, and located in which are represented by the identifiers `wdt:P31`, `wdt:P19`, `wdt:P27`, `wdt:P106`, `wdt:P17`, and `wdt:P131`, respectively. Finally, the tripleset was further reduced to yield 2525 triples, 716 entities, and 6 predicates.

6.5.2 Quantitative Evaluation

In our quantitative evaluation, we first analyzed the quality of our learned hierarchical clustering by calculating two clustering quality metrics at each level of the hierarchy: the Adjusted Rand Index (ARI) [132] and Normalized Mutual Information (NMI) [133]. This type of evaluation jointly assesses the quality of the learned community hierarchy as well as the membership of entities to communities.

The ARI is an adjustment to the commonly used Rand Index (RI) [134], corrected to account for chance. Specifically, chance is factored in by calculating the expected RI given a random clustering and measuring the obtained clustering's deviation. Specifically, given an obtained entity clustering $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_o\}$ and the ground truth clustering $\mathcal{C}^* = \{\mathcal{C}_1^*, \mathcal{C}_2^*, \ldots, \mathcal{C}_t^*\}$, the ARI is calculated as follows:

$$ARI = \frac{\sum_{\mathcal{C}_i \in \mathcal{C}} \sum_{\mathcal{C}_j^* \in \mathcal{C}^*} \binom{\#_{ij}}{2} - \binom{|\mathcal{E}|}{2}^{-1} \Big(\sum_{\mathcal{C}_i \in \mathcal{C}} \binom{\#_i}{2} \sum_{\mathcal{C}_j^* \in \mathcal{C}^*} \binom{\#_j^*}{2} \Big)}{2^{-1} \Big(\sum_{\mathcal{C}_i \in \mathcal{C}} \binom{\#_i}{2} + \sum_{\mathcal{C}_j^* \in \mathcal{C}^*} \binom{\#_j^*}{2} \Big) - \binom{|\mathcal{E}|}{2}^{-1} \Big(\sum_{\mathcal{C}_i \in \mathcal{C}} \binom{\#_i}{2} \sum_{\mathcal{C}_j^* \in \mathcal{C}^*} \binom{\#_j^*}{2} \Big)} \tag{6.20}$$

Where $\#_{ij} = |\mathcal{C}_i \cap \mathcal{C}_j^*|$ is the number of entities in common between a ground truth and obtained cluster pair; $\#_i = \sum_{\mathcal{C}_j^* \in \mathcal{C}^*} \#_{ij}$ is the total number of entities in obtained cluster $\mathcal{C}_i$; and $\#_j^* = \sum_{\mathcal{C}_i \in \mathcal{C}} \#_{ij}$ is the total number of entities in ground truth cluster $\mathcal{C}_j^*$.

The NMI is a normalized extension of the Mutual Information (MI) score which quantifies the information gained about the obtained clustering given the ground truth clusters. The normalization of the MI score ensures the result is in the range $[0, 1]$ thereby allowing for its comparison against clusterings of different sizes. Utilizing the notation defined earlier, we define MI and NMI as follows:

$$\begin{aligned} MI &= \sum_{\mathcal{C}_i \in \mathcal{C}} \sum_{\mathcal{C}_j^* \in \mathcal{C}^*} \frac{|\mathcal{C}_i \cap \mathcal{C}_j^*|}{|\mathcal{E}|} \log \left(\frac{|\mathcal{E}||\mathcal{C}_i \cap \mathcal{C}_j^*|}{|\mathcal{C}_i||\mathcal{C}_j^*|} \right) \\ NMI &= \frac{MI}{\text{mean}\left(- \sum_{\mathcal{C}_i \in \mathcal{C}} \frac{|\mathcal{C}_i|}{|\mathcal{E}|} \log \left(\frac{|\mathcal{C}_i|}{|\mathcal{E}|} \right), - \sum_{\mathcal{C}_j^* \in \mathcal{C}^*} \frac{|\mathcal{C}_j^*|}{|\mathcal{E}|} \log \left(\frac{|\mathcal{C}_j^*|}{|\mathcal{E}|} \right) \right)} \end{aligned} \tag{6.21}$$

For both the ARI and NMI, higher scores indicate a clustering of higher quality. We summarize the results of our clustering as per these two metrics in Table 6.2.

In general, the results indicate that our model is capable of learning a coherent community hierarchy on each of the three datasets tested. Perhaps unsurprisingly, communities at higher levels in the hierarchy are judged as higher quality as per the two evaluation metrics. This is because the task of clustering entities at higher levels is simpler as the communities are less fine grained. For instance, on the FB15k-237 dataset, clustering at level 1 requires the distinction between `Place` and `Person` whereas level 4 requires the distinction between `AmericanFootballPlayer` and `IceHockeyPlayer`. We note that the SBT dataset is an exception to this. This is likely due to the nature of the dataset wherein entity relations are drawn at higher

	SBT		**FB15k-237**		**WikiData**	
Method	ARI	NMI	ARI	NMI	ARI	NMI
Level 1	0.3055	0.4855	0.5326	0.6646	0.8411	0.7991
	±0.0685	±0.1013	±0.1308	±0.0702	±0.2980	±0.1581
Level 2	0.5895	0.7826	0.3492	0.5083	0.8057	0.7232
	±0.2826	±0.1434	±0.2044	±0.1175	±0.2839	±0.1410
Level 3	0.7279	0.8882	0.2851	0.4329	0.4255	0.5880
	±0.1656	±0.0621	±0.1993	±0.1030	±0.2749	±0.1367
Level 4	0.8337	0.9319	0.1964	0.5334	0.3812	0.4980
	±0.1032	±0.0357	±0.0438	±0.0288	±0.2500	±0.1309
Overall	0.6141	0.7721	0.3408	0.5348	0.6134	0.6521
	±0.2577	±0.1988	±0.1867	±0.1145	±0.3341	±0.1770

Table 6.2: ARI and MNI scores (mean ± standard deviation) of our model on the SBT, FB15k-237 and WikiData datasets.

proportions between neighbouring communities at lower levels of the hierarchy. In this sense, the claim made before gets inverted and it's easier to assign communities at lower levels in the hierarchy. We also compared our model against embedding and clustering methods used in conjunction. Specifically, we first embedded each of the knowledge graphs using the RDF2VEC and TransE embedding methods. Afterwards, we applied four clustering methods: *k*-means, Agglomerative, DBSCAN, and Spectral.These results are summarized in Table 6.3 and indicate comparable or superior performance to baselines.

We can also analyze the results of the complete log likelihood as a function of the number of Gibbs samples taken in the inference process. Indeed, while this does not provide us information about the quality of the obtained results, it does verify the inference process itself. Specifically, we expect to see the log likelihood of our model to rise given more burn-in samples of the Gibbs sampler. This suggests that the likelihood of generating the knowledge graph given the current state of the sampler is

	SBT		**FB15k-237**		**WikiData**	
Method	ARI	NMI	ARI	NMI	ARI	NMI
RDF2VEC						
k-means	0.8060	0.8928	0.0109	0.1402	0.2672	0.2918
	±0.1845	±0.0707	±0.0929	±0.1052	±0.1582	±0.1040
Agglomerative	0.8750	0.9317	0.0461	0.1532	0.4674	0.5287
	±0.1254	±0.0575	±0.0860	±0.1435	±0.3281	±0.2052
DBSCAN	0.5549	0.6904	0.1468	0.2293	0.3831	0.3698
	±0.4576	±0.3032	±0.1291	±0.0561	±0.2343	±0.0935
Spectral	0.6175	0.7590	-0.0014	0.0347	0.0918	0.1021
	±0.3540	±0.2924	±0.0082	±0.03129	±0.0636	0.0297
TransE						
k-means	0.9851	0.9958	0.3559	0.4334	0.7427	0.6504
	±0.0334	±0.0066	±0.0776	±0.1096	±0.1953	±0.2468
Agglomerative	1.0000	1.0000	0.1362	0.3107	0.3799	0.3650
	±0.0000	±0.0000	±0.1379	±0.1104	±0.4037	±0.3780
DBSCAN	0.8899	0.9665	0.2768	0.2582	0.2418	0.3128
	±0.1213	±0.03829	±0.1616	±0.0728	±0.1355	±0.1143
Spectral	1.0000	1.0000	0.1400	0.2509	0.2778	0.3296
	±0.0000	±0.0000	±0.1920	±0.2418	±0.1541	±0.0153
Our method	0.6141	0.7721	0.3408	0.5348	0.6134	0.6521
	±0.2577	±0.1988	±0.1867	±0.1145	±0.3341	±0.1770

Table 6.3: ARI and MNI scores (mean ± standard deviation) of our model on the SBT, FB15k-237 and WikiData datasets as compared with baseline approaches.

increasing and learning is taking place. We can see this rise in Figure 6.4 which plots the complete log likelihoods of our model across Gibbs samples for the three datasets. We note a dips in log likelihoods on the SBT and WikiData datasets. This is likely due to the sampler being temporarily stuck in a local minimum before leaving that area in the sample space.

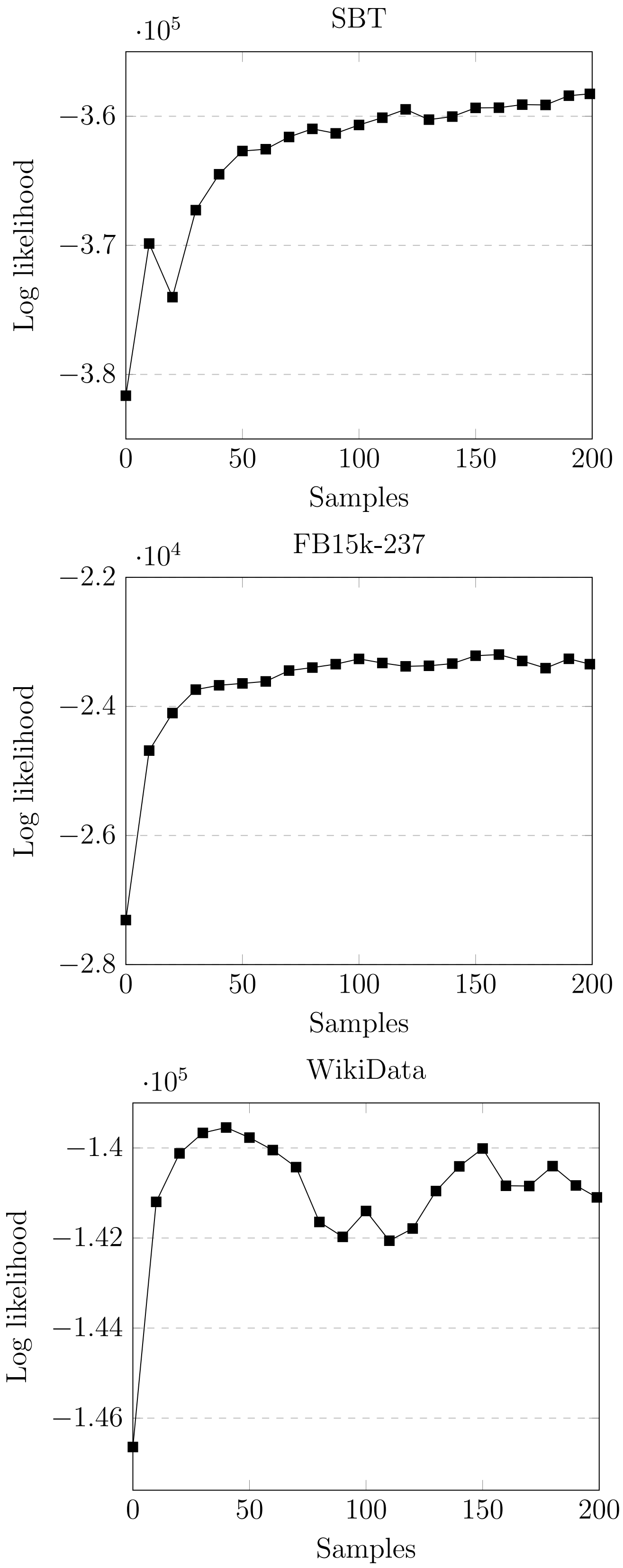


Figure 6.4: Plots of average log likelihood of our model across burn in samples on three datasets.

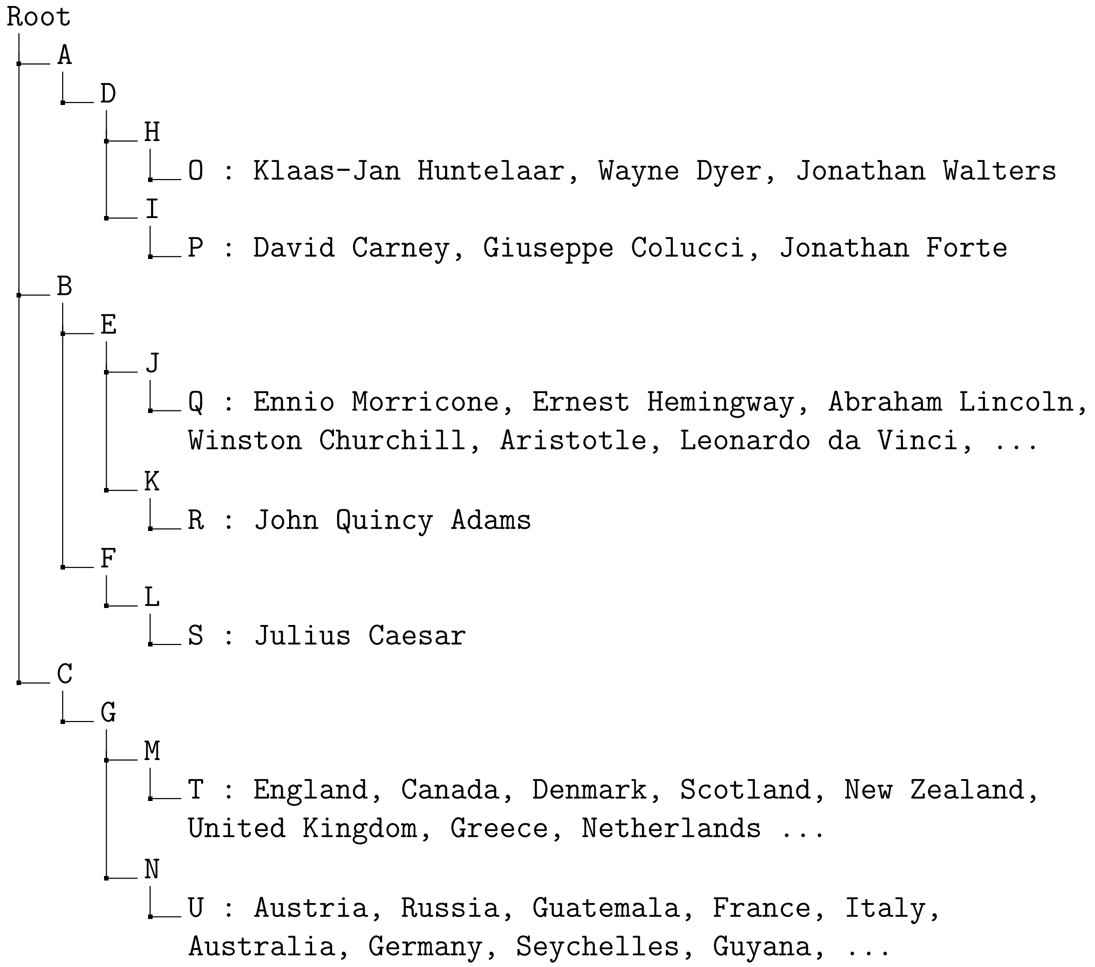

Root
A
D
H
O : Klaas-Jan Huntelaar, Wayne Dyer, Jonathan Walters
I
P : David Carney, Giuseppe Colucci, Jonathan Forte
B
E
J
Q : Ennio Morricone, Ernest Hemingway, Abraham Lincoln, Winston Churchill, Aristotle, Leonardo da Vinci, ...
K
R : John Quincy Adams
F
L
S : Julius Caesar
C
G
M
T : England, Canada, Denmark, Scotland, New Zealand, United Kingdom, Greece, Netherlands ...
N
U : Austria, Russia, Guatemala, France, Italy, Australia, Germany, Seychelles, Guyana, ...

Figure 6.5: Excerpt of our induced hierarchy on the FB15k-237 dataset.

6.5.3 Qualitative Evaluation

We begin our qualitative evaluation by investigating the results obtained on the FB15k-237 dataset as seen in the excerpt provided in Figure 6.5. Perhaps the most glaring inaccuracy is the misplacement of footballers into the subtree rooted in community A. Footballers should ideally be placed as a subtree within the subtree rooted by community B, along with the majority of other persons in the dataset. Furthermore, the footballers assigned to different communities in their subtree do not appear to have any properties which would warrant their splitting. For instance, the footballers in community O do not share properties that make them more alike to one another than to the footballers in community P. The first issue can largely be explained by the data itself. Footballers in our dataset have structural properties which differ them from the persons clustered in the community B subtree. In addition to sharing the same profession, they belong to football teams, have football specific triples such as the position they play, and are more likely than non-footballer persons in the dataset to have information relating to physical characteristics such as height and weight. In contrast, non-footballer persons have less structural similarities, even within members of their own professions. For instance, even though all scientists have their scientific contributions, this isn't reflected in the data as uniformly. This segues to another issue with the learned hierarchy, namely that the remaining occupations – pianists, journalists, politicians, and scientists – are not sufficiently split in the clustering. Indeed, the majority of them belong to community Q. A closer look at the dataset reveals that unlike with footballers, there is insufficient structural information to induce a split in our model. The final issue in the hierarchy is the splitting of nations into two communities in the third level. A cursory glance does not reveal why such a splitting took place. There is no evident geographic, social, political, or economic distinction made in the clustering. A deeper look into the data also reveals no apparent structural differences between communities M and N. It is likely, therefore, that this issue can be chalked up to model inaccuracy and arrival at a suboptimal state after inference.

An excerpt of the hierarchy learned on the WikiData dataset is captured in Figure 6.6. In general, this dataset is simpler than FB15k-237 due to there being only two professions with largely disjoint neighbourhoods. With this in mind, a major gripe with the hierarchy is the splitting of persons – namely footballers and painters – and places – namely cities and countries – too high in the hierarchy. Specifically, this happens at the first level in the presented excerpt where persons are split into communities A and B and places into communities C and D. We note, however, the high ARI and NMI scores at this level despite this error. A closer inspection reveals that not all runs of our model encounter this issue, thus the scores are higher than

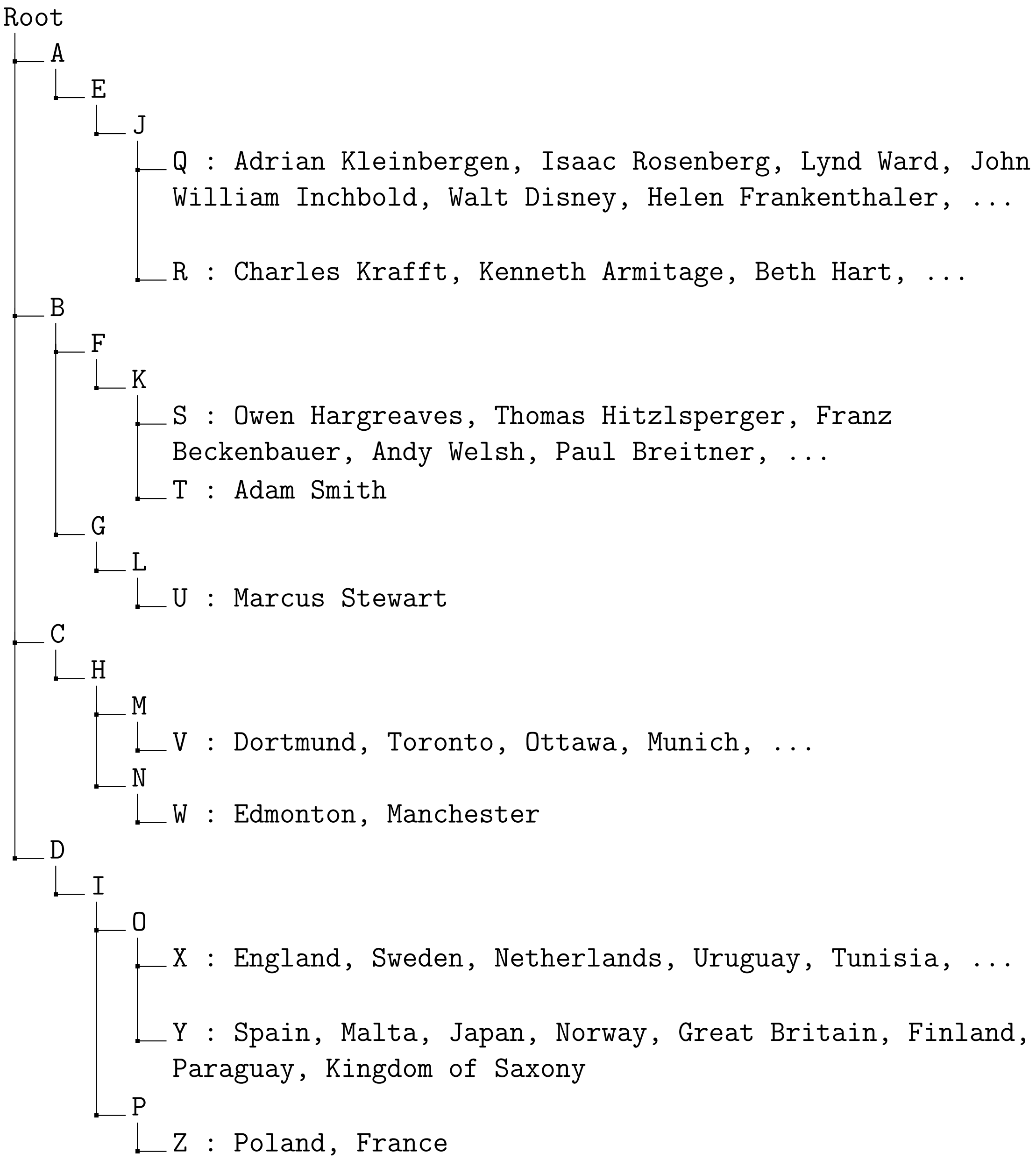

Root
A
E
J
Q : Adrian Kleinbergen, Isaac Rosenberg, Lynd Ward, John William Inchbold, Walt Disney, Helen Frankenthaler, ...
R : Charles Krafft, Kenneth Armitage, Beth Hart, ...
B
F
K
S : Owen Hargreaves, Thomas Hitzlsperger, Franz Beckenbauer, Andy Welsh, Paul Breitner, ...
T : Adam Smith
G
L
U : Marcus Stewart
C
H
M
V : Dortmund, Toronto, Ottawa, Munich, ...
N
W : Edmonton, Manchester
D
I
O
X : England, Sweden, Netherlands, Uruguay, Tunisia, ...
Y : Spain, Malta, Japan, Norway, Great Britain, Finland, Paraguay, Kingdom of Saxony
P
Z : Poland, France

Figure 6.6: Excerpt of our induced hierarchy on the WikiData dataset.

they would be had they been measured only on the hierarchy in Figure 6.6. Otherwise, the hierarchy also suffers from unsubstantiated splits at the lowest levels as seen in communities R, T, and Y. This issue may be the result of the small size of the data and the model's resultant sensitivity to the relational information provided. For instance, close to half of the entities in this dataset have have first order neighborhoods of one or two entities, giving our model little to learn from.

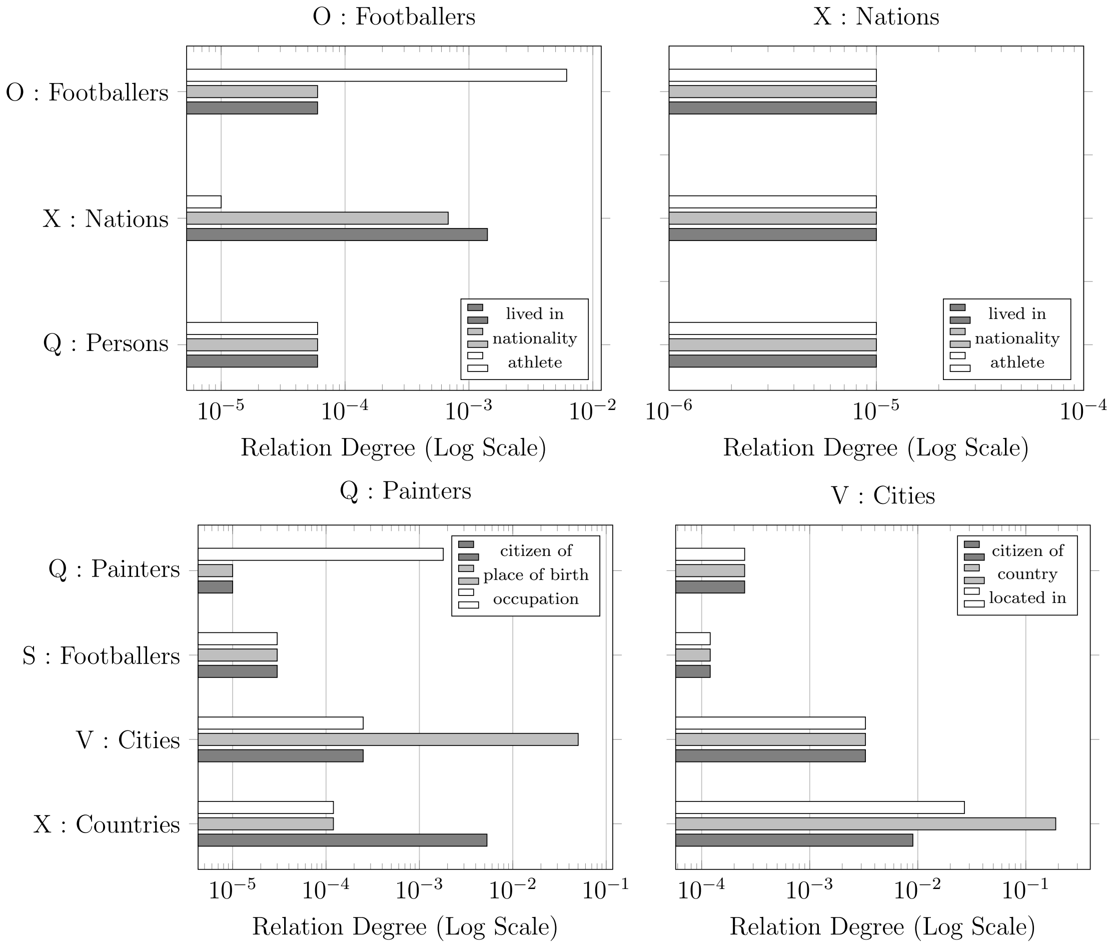


Figure 6.7: Plots of learned community relations for selected outgoing predicates for the FB15k-237 and WikiData datasets. Specifically we showcased community O (Footballers) and community T (Nations) outgoing relations for the FB15k-237 dataset (top) and community Q (Painters) and community V (Cities) outgoing relations for the WikiData dataset.

Finally, we analyze the learned community relations for the FB15k-237 and WikiData datasets as shown in Figure 6.7. The community labels correspond to the lettering in Figures 6.5 and 6.6. We added descriptive labels in aid in understanding, however these do not come from the model itself. The results indicate community relations which are largely expected. For instance, on the FB15k-237 dataset, Footballers are much more likely to be related to nations by predicates nationality and lived in than athlete. Furthermore, we see that Nations are equally unlikely to be the subjects with the other communities or predicates such as lived in, nationality, and athlete. On the WikiData dataset, we likewise see explainable results. Painters, for instance, are likely to be related to cities by place of birth and countries by nationality. They are unlikely to relate to other painters and footballers on these predicates.

6.6 Conclusion

In this chapter we examined the topic of using stochastic blockmodels to learn hierarchies from knowledge graphs. This presents a first in the knowledge graph community as these types of models have not been traditionally applied to this task. To this end we propose a hierarchical blockmodel which leverages the nCRP and stick breaking process to generate a community hierarchy. The model is fully non-parametric, capable of learning a potentially infinite number of communities on an infinite number of levels. In addition to the model itself, we propose an MCMC inference scheme leveraging collapsed Gibbs sampling. Results on three dataset suggest that our model is capable of learning coherent community hierarchies as demonstrated by quantitative and qualitative analysis. Currently the main issue with the approach is scalability to large scale datasets. Indeed, the three used in the evaluation are dwarfed by the size of existing knowledge bases like DBpedia and WikiData. In this respect, there is much potential for future work, as stochastic blockmodels lend themselves to inference schemes faster than the one proposed in this chapter. We briefly discuss these approaches in the following chapter.

Chapter 7

Conclusion

The conclusions for each of the four projects which make up this thesis have already been made in each of their respective chapters. We thus devote this section to a summary of the contributions of the thesis along with broad, overarching conclusions and potential directions for future work.

7.1 Contributions

This thesis explored different facets of learning hierarchies from knowledge graphs from inducing class taxonomies to utilizing coarsening for enhancing knowledge graph embeddings to generating a hierarchical clustering of knowledge graph entities. To this end, it proposed several novel methods for learning hierarchies and demonstrated their effectiveness on real world data. In summary, the contributions are:

- In Chapter 3, we introduced a novel method for class taxonomy induction from knowledge graphs using a greedy algorithm based on class frequencies and co-occurrences. The dearth of such methods presents an important area for contribution given the utility of the class taxonomy in organizing data. We showed that in an arena of few competitors, our method and those adjacent to it outperforms existing class taxonomy induction methods. Furthermore, we introduced the idea of treating a knowledge graph as sets of entities and tags, simplifying its structure and opening the door to methods used in natural language processing

(NLP) such as topic models [17].

- Afterwards, in Chapter 4, we developed a meta-strategy for knowledge graph embedding using coarsening as a preprocessing step. Embeddings present one of the most widely investigated knowledge graph representations in the community and their utility reaches far down to downstream tasks such as link prediction and entity classification. As such, developing an approach which enhances their learning is highly desired. In this regard, our meta-strategy proposes to carry out most of the learning on a coarsened version of the original knowledge graph, thereby reducing its computational complexity. We showed that such an approach is capable of attaining faster and oftentimes higher quality knowledge graph embeddings.

- In Chapter 5, we married stochastic blockmodels with neural networks for the purpose of modelling knowledge graphs in what is – to the best of our knowledge – the first time such a fusion has been investigated. This allows the model to perform the tasks of both of the approaches it draws inspiration from. More importantly, it highlights the utility of stochastic blockmodels in the context of knowledge graphs. In doing so, it serves as the foundation for its successor, a stochastic blockmodel for learning hierarchies in knowledge graphs.

- Finally, in Chapter 6, we proposed a stochastic blockmodel for learning hierarchies from knowledge graphs. Our approach expands upon the other works in this thesis in that it offers the induction of a hierarchical clustering of knowledge graph entities and models the degree of predicate specific relationship between clusters. Furthermore, it is formulated in a non-parametric and fully probabilistic framework, allowing for greater flexibility in learning the structure of the hierarchy. To the best of our knowledge, this the first time stochastic blockmodels have been applied to knowledge graphs for the purpose of learning hierarchies. In general, utilizing stochastic blockmodels for modelling knowledge graphs is an understudied area in the knowledge graph community. By developing stochastic blockmodels specifically suited to the domain of knowledge graphs, the foundations are set for further adoption of such models in the future.

We anticipate that these contributions will be utilized and expanded upon by the community in the future. With this in mind, the problem of learning hierarchies from knowledge graphs has not been solved. Indeed, there is no single method capable of performing all the aforementioned tasks and many of the hierarchies learned by the existing methods – whether described in this thesis or not – leave much to be desired. That is to say we have not reached a point of optimal automated learning of hierarchies from knowledge graphs. It is our expectation that future practitioners will continue to make progress in this oftentimes neglected area. This provides a segue into the final topic discussed in this thesis, namely the potential directions which could be taken in future work.

7.2 Future Directions

Regarding our taxonomy induction method, there are still challenges which, due to their scope and complexity, remain unaddressed and are thus topics for future consideration. We draw attention to two of these, namely typing errors and gold standard taxonomy evaluation. The first of these refers to the problem of typing errors that exist in a knowledge graph. Typing errors refer to incorrect information in triples which assign types to entities. For instance, the triple ⟨`dbo:Edmonton`, `rdf:type`, `dbo:Country`⟩ would constitute a typing error because Edmonton is not a country. This problem is widespread in existing knowledge bases with estimates concluding that up to 27% of the entity in DBpedia incorrectly assigned to a type [135]. This presents a significant challenge to our method since type information is what is leveraged in inducing the taxonomy. By first preprocessing the input data to remove incorrect type information, it may be possible to obtain better results. This provides a segue into the problem with how results are evaluated and interpreted when using a gold standard taxonomy. We touched on this briefly in Chapter 3 by drawing attention to the fact that there may be more than one correct way to organize a taxonomy. Furthermore, we provided examples of subsumption axioms which may are incorrect as per a gold standard taxonomy but may be considered correct by a human evaluator. Despite this, we never resolved this issue in our evaluation. Indeed, to do so would require a panel of impartial evaluators to qualitatively judge the correctness of taxonomies induced with our method and those of competitors. Finding a method for better evaluation of taxonomies thus presents an interesting topic for future work.

The most salient problem which arises when applying stochastic blockmodels to knowledge graphs is that of scalability. Indeed, these types of models are more computationally complex than other approaches on simple graphs, so the additional relational information in knowledge graphs exacerbates the problem. Having said this, there is reason for optimism. First of all, the two optimization schemes presented in this work, namely stochastic gradient descent and Gibbs sampling, generally do not scale well to large datasets. As such, it may be worth investigating their replacement for a more scalable method. Gibbs sampling, for instance, may be replaceable by variational inference which uses the evidence lower bound to guide the training process to obtain the posterior distribution. This process is generally faster than Gibbs sampling and, although not asymptotically exact, produces similar results [136]. The challenge with this approach is that the optimization equations are not easy to obtain compared to Markov chain Monte Carlo methods. Despite this, several works have already successfully applied variational inference to probabilistic graphical models. Indeed, the original inference scheme for the MMSB leveraged variational inference. Furthermore, Blei and Jordan [137] provided a variation inference scheme for Dirichlet processes and a variational inference scheme for the nCRP was proposed in Wang and Blei [138]. Departing from the variational approach, Chen et al. [139] propose an evolution of the Gibbs sampling algorithm for the nCRP with partially collapsed Gibbs sampling. This approach resulted in a 111 times increase in efficiency over the classic Gibbs sampling approach. Another line of approach to increase the scalability of stochastic blockmodels is to devise a model which does not require sampling all $|\mathcal{E}|^2|\mathcal{R}|$ relations in the knowledge graph directly. To this end, the Bernoulli-Poisson link function has been applied successfully to simple graphs [140–142]. These methods eliminate the need for quadratic time relation sampling and instead rely on density based sampling which is less computationally demanding, especially on sparse networks. Given that most knowledge graphs are highly sparse, applying such an approach appears promising.

Bibliography

[1] J. Lehmann *et al.*, “Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia,” *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015.

[2] T. Pellissier Tanon, G. Weikum, and F. Suchanek, “Yago 4: A reason-able knowledge base,” in *European Semantic Web Conference*, Springer, 2020, pp. 583–596.

[3] D. Vrandečić and M. Krötzsch, “Wikidata: A free collaborative knowledgebase,” *Communications of the ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[4] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, “Freebase: A collaboratively created graph database for structuring human knowledge,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, AcM, 2008, pp. 1247–1250.

[5] A. Singhal, *Introducing the knowledge graph: Things, not strings*, https://www.blog.google/pr knowledge-graph-things-not/, 2012, accessed January 28, 2022. (visited on 01/28/2022).

[6] R. Xie, Z. Liu, M. Sun, *et al.*, “Representation learning of knowledge graphs with hierarchical types.,” in *IJCAI*, vol. 2016, 2016, pp. 2965–2971.

[7] Z. Zhang, J. Cai, Y. Zhang, and J. Wang, “Learning hierarchy-aware knowledge graph embeddings for link prediction,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 3065–3072.

[8] A. Hogan *et al.*, “Knowledge graphs,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–37, 2021.

[9] C. Gutierrez and J. F. Sequeda, “Knowledge graphs,” *Communications of the ACM*, vol. 64, no. 3, pp. 96–104, 2021.

[10] J. Pérez, M. Arenas, and C. Gutierrez, “Semantics and complexity of sparql,” *ACM Transactions on Database Systems (TODS)*, vol. 34, no. 3, pp. 1–45, 2009.

[11] N. Francis *et al.*, “Cypher: An evolving query language for property graphs,” in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1433–1445.

[12] P. W. Holland, K. B. Laskey, and S. Leinhardt, “Stochastic blockmodels: First steps,” *Social networks*, vol. 5, no. 2, pp. 109–137, 1983.

[13] X. Wang *et al.*, "Ppisb: A novel network-based algorithm of predicting protein-protein interactions with mixed membership stochastic blockmodel," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2022.

[14] T. M. Sweet, "Modeling social networks as mediators: A mixed membership stochastic blockmodel for mediation," *Journal of Educational and Behavioral Statistics*, vol. 44, no. 2, pp. 210–240, 2019.

[15] E. M. Airoldi, D. Blei, S. Fienberg, and E. Xing, "Mixed membership stochastic blockmodels," *Advances in neural information processing systems*, vol. 21, 2008.

[16] Q. Ho, A. Parikh, L. Song, and E. Xing, "Multiscale community blockmodel for network exploration," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, 2011, pp. 333–341.

[17] Y. Zhang, M. Pietrasik, W. Xu, and M. Reformat, "Hierarchical topic modelling for knowledge graphs," in *European Semantic Web Conference*, Springer, 2022, pp. 270–286.

[18] D. J. MacKay, D. J. Mac Kay, *et al.*, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[19] E. Abbe, "Community detection and stochastic block models: Recent developments," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6446–6531, 2017.

[20] D. J. Aldous, "Exchangeability and related topics," in *École d'Été de Probabilités de Saint-Flour XIII—1983*, Springer, 1985, pp. 1–198.

[21] T. S. Ferguson, "A bayesian analysis of some nonparametric problems," *The annals of statistics*, pp. 209–230, 1973.

[22] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[23] T. Griffiths, M. Jordan, J. Tenenbaum, and D. Blei, "Hierarchical topic models and the nested chinese restaurant process," *Advances in neural information processing systems*, vol. 16, 2003.

[24] D. M. Blei, T. L. Griffiths, and M. I. Jordan, "The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies," *Journal of the ACM (JACM)*, vol. 57, no. 2, pp. 1–30, 2010.

[25] J. Sethuraman, "A constructive definition of dirichlet priors," *Statistica sinica*, pp. 639–650, 1994.

[26] J. Pitman *et al.*, "Combinatorial stochastic processes," Technical Report 621, Dept. Statistics, UC Berkeley, 2002. Lecture notes for ..., Tech. Rep., 2002.

[27] B. Pakkenberg and H. J. G. Gundersen, "Neocortical neuron number in humans: Effect of sex and age," *Journal of comparative neurology*, vol. 384, no. 2, pp. 312–320, 1997.

[28] A. Gulati, "Understanding neurogenesis in the adult human brain," *Indian journal of pharmacology*, vol. 47, no. 6, p. 583, 2015.

[29] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[32] D. H. Ballard, "Modular learning in neural networks.," in *Aaai*, vol. 647, 1987, pp. 279–284.

[33] S. B. Kotsiantis, I. Zaharakis, P Pintelas, *et al.*, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[34] G. P. Zhang, "Neural networks for classification: A survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 4, pp. 451–462, 2000.

[35] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[36] R. Andrews, J. Diederich, and A. B. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge-based systems*, vol. 8, no. 6, pp. 373–389, 1995.

[37] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*, Ieee, 2013, pp. 6645–6649.

[38] T. Bluche, J. Louradour, M. Knibbe, B. Moysset, M. F. Benzeghiba, and C. Kermorvant, "The a2ia arabic handwritten text recognition system at the open hart2013 evaluation," in *2014 11th IAPR International Workshop on Document Analysis Systems*, IEEE, 2014, pp. 161–165.

[39] J. Gonzalez-Dominguez, I. Lopez-Moreno, and H. Sak, "Automatic language identification using long short-term memory recurrent neural networks," 2014.

[40] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, "Explaining deep neural networks and beyond: A review of methods and applications," *Proceedings of the IEEE*, vol. 109, no. 3, pp. 247–278, 2021.

[41] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[42] S. Haykin, *Neural networks and learning machines, 3/E*. Pearson Education India, 2009.

[43] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," Cornell Aeronautical Lab Inc Buffalo NY, Tech. Rep., 1961.

[44] M. Minsky and S. Papert, "An introduction to computational geometry," *Cambridge tiass., HIT*, vol. 479, p. 480, 1969.

[45] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[46] M. Pietrasik and M. Reformat, "A simple method for inducing class taxonomies in knowledge graphs," in *European Semantic Web Conference*, Springer, 2020, pp. 53–68.

[47] M. Pietrasik and M. Reformat, "Path based hierarchical clustering on knowledge graphs," *arXiv preprint arXiv:2109.13178*, 2021.

[48] J. Völker and M. Niepert, "Statistical schema induction," in *Extended Semantic Web Conference*, Springer, 2011, pp. 124–138.

[49] M. Nickel, V. Tresp, and H.-P. Kriegel, "Factorizing yago: Scalable machine learning for linked data," in *Proceedings of the 21st international conference on World Wide Web*, ACM, 2012, pp. 271–280.

[50] M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data.," 2011.

[51] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," in *ACM Sigmod record*, ACM, vol. 28, 1999, pp. 49–60.

[52] P. Ristoski, S. Faralli, S. P. Ponzetto, and H. Paulheim, "Large-scale taxonomy induction using entity and word embeddings," in *Proceedings of the International Conference on Web Intelligence*, ACM, 2017, pp. 81–87.

[53] P. Heymann and H. Garcia-Molina, "Collaborative creation of communal hierarchical taxonomies in social tagging systems," Tech. Rep., 2006.

[54] D. Benz, A. Hotho, S. Stützer, and G. Stumme, "Semantics made by you and me: Self-emerging ontologies can capture the diversity of shared knowledge," 2010.

[55] P. Schmitz, "Inducing ontology from flickr tags," in *Collaborative Web Tagging Workshop at WWW2006, Edinburgh, Scotland*, vol. 50, 2006, p. 39.

[56] M. Sanderson and B. Croft, "Deriving concept hierarchies from text," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, ACM, 1999, pp. 206–213.

[57] G. Solskinnsbakk and J. A. Gulla, "A hybrid approach to constructing tag hierarchies," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, 2010, pp. 975–982.

[58] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[59] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[60] J. Tang, H.-f. Leung, Q. Luo, D. Chen, and J. Gong, "Towards ontology learning from folksonomies," in *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[61] X. Li *et al.*, "Inducing taxonomy from tags: An agglomerative hierarchical clustering framework," in *International Conference on Advanced Data Mining and Applications*, Springer, 2012, pp. 64–77.

[62] S. Wang, D. Lo, and L. Jiang, "Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, IEEE, 2012, pp. 604–607.

[63] H. Dong, W. Wang, and F. Coenen, "Learning relations from social tagging data," in *Pacific Rim International Conference on Artificial Intelligence*, Springer, 2018, pp. 29–41.

[64] D. M. Roy, C. Kemp, V. K. Mansinghka, and J. B. Tenenbaum, "Learning annotated hierarchies from relational data," in *Advances in neural information processing systems*, 2007, pp. 1185–1192.

[65] J. X. Chen and M. Z. Reformat, "Learning categories from linked open data," in *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Springer, 2014, pp. 396–405.

[66] G. J. Székely, M. L. Rizzo, N. K. Bakirov, *et al.*, "Measuring and testing dependence by correlation of distances," *The annals of statistics*, vol. 35, no. 6, pp. 2769–2794, 2007.

[67] S. K. Mohamed, "Unsupervised hierarchical grouping of knowledge graph entities," *arXiv preprint arXiv:1908.07281*, 2019.

[68] F. Martel and A. Zouaq, "Taxonomy extraction using knowledge graph embeddings and hierarchical clustering," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, 2021, pp. 836–844.

[69] Z. Ding, D. Cao, L. Liu, D. Yu, H. Ma, and F. Wang, "A method for discovering hidden patterns of cybersecurity knowledge based on hierarchical clustering," in *2021 IEEE Sixth International Conference on Data Science in Cyberspace (DSC)*, IEEE, 2021, pp. 334–338.

[70] C. Gu, G. Yin, T. Wang, C. Yang, and H. Wang, "A supervised approach for tag hierarchy construction in open source communities," in *Proceedings of the 7th Asia-Pacific Symposium on Internetware*, ACM, 2015, pp. 148–152.

[71] W. Wang, P. M. Barnaghi, and A. Bargiela, “Probabilistic topic models for learning terminological ontologies,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 7, pp. 1028–1040, 2009.

[72] K. Liu, B. Fang, and W. Zhang, “Ontology emergence from folksonomies,” in *Proceedings of the 19th ACM international conference on Information and knowledge management*, ACM, 2010, pp. 1109–1118.

[73] F. Almoqhim, D. E. Millard, and N. Shadbolt, “Improving on popularity as a proxy for generality when building tag hierarchies from folksonomies,” in *International Conference on Social Informatics*, Springer, 2014, pp. 95–111.

[74] N. Chinchor, “Muc-4 evaluation metrics,” in *Proceedings of the 4th conference on Message understanding*, Association for Computational Linguistics, 1992, pp. 22–29.

[75] O. T. N. D. B. N. K. P. B. T. D. R. D. W. N. E. v. Z. J. P. L. e. Roskov Y. Ower G., “Species 2000 & itis catalogue of life, 2019 annual checklist.,” 2019.

[76] M. Döring. “Gbif type specimen names.” (2017), [Online]. Available: https://doi.org/10.15468/sl9pyf (visited on 11/27/2019).

[77] G. A. Miller, “Wordnet: A lexical database for english,” *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[78] J. Euzenat *et al.*, “Results of the ontology alignment evaluation initiative 2010,” University of Trento, Tech. Rep., 2011.

[79] H. Paulheim and J. Fümkranz, “Unsupervised generation of data mining features from linked open data,” in *Proceedings of the 2nd international conference on web intelligence, mining and semantics*, ACM, 2012, p. 31.

[80] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, “Harp: Hierarchical representation learning for networks,” in *Proc. 32nd AAAI Conf.Artif. Intell.*, 2018, 2127––2134.

[81] J. Liang, S. Gurukar, and S. Parthasarathy, “Mile: A multi-level framework for scalable graph embedding,” *arXiv preprint arXiv:1802.09612*, 2018.

[82] D. Archdeacon, “Topological graph theory,” *A survey. Congressus Numerantium*, vol. 115(5-54):18, 1996.

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

[84] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, “Line: Large-scale information network embedding,” in *Proceedings of the 24th international conference on world wide web*, 2015.

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

[86] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, pp. 3111–3119, 2013.

[87] D. K. Duvenaud *et al.*, "Convolutional networks on graphs for learning molecular fingerprints," *Advances in neural information processing systems*, vol. 28, pp. 2224–2232, 2015.

[88] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[89] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[90] V. Bellini, A. Schiavone, T. Di Noia, A. Ragone, and E. Di Sciascio, "Knowledge-aware autoencoders for explainable recommender systems," in *Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems*, 2018.

[91] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," in *International Conference on Artificial Neural Networks*, Springer, 2018, pp. 412–422.

[92] T. A. Akyildiz, A. A. Aljundi, and K. Kaya, "Gosh: Embedding big graphs on small hardware," in *49th International Conference on Parallel Processing-ICPP*, 2020, pp. 1–11.

[93] P. Ristoski and H. Paulheim, "Rdf2vec: Rdf graph embeddings for data mining," in *International Semantic Web Conference*, Springer, 2016, pp. 498–514.

[94] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*, Springer, 2018, pp. 593–607.

[95] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2d knowledge graph embeddings," *arXiv preprint arXiv:1707.01476*, 2017.

[96] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," *Advances in neural information processing systems*, vol. 26, pp. 2787–2795, 2013.

[97] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," *arXiv preprint arXiv:1412.6575*, 2014.

[98] R. Das *et al.*, "Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning," *arXiv preprint arXiv:1711.05851*, 2017.

[99] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition and applications," *arXiv preprint arXiv:2002.00388*, 2020.

[100] B. Hendrickson and R. W. Leland, "A multi-level algorithm for partitioning graphs.," *SC*, vol. 95, no. 28, pp. 1–14, 1995.

[101] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[102] M. Pietrasik and M. Reformat, "Neural blockmodeling for multilayer networks," in *2021 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2021, pp. 1–8.

[103] K. Nowicki and T. A. B. Snijders, "Estimation and prediction for stochastic blockstructures," *Journal of the American statistical association*, vol. 96, no. 455, pp. 1077–1087, 2001.

[104] C. Kemp, J. B. Tenenbaum, T. L. Griffiths, T. Yamada, and N. Ueda, "Learning systems of concepts with an infinite relational model," in *AAAI*, vol. 3, 2006, p. 5.

[105] X. Fan, L. Cao, and R. Y. Da Xu, "Dynamic infinite mixed-membership stochastic blockmodel," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 9, pp. 2072–2085, 2014.

[106] M. Berlingerio, M. Coscia, and F. Giannotti, "Finding redundant and complementary communities in multidimensional networks," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, pp. 2181–2184.

[107] M. Barigozzi, G. Fagiolo, and G. Mangioni, "Identifying the community structure of the international-trade multi-network," *Physica A: statistical mechanics and its applications*, vol. 390, no. 11, pp. 2051–2066, 2011.

[108] S. Paul and Y. Chen, "Consistent community detection in multi-relational data through restricted multi-layer stochastic blockmodel," *Electronic Journal of Statistics*, vol. 10, no. 2, pp. 3807–3870, 2016.

[109] C. De Bacco, E. A. Power, D. B. Larremore, and C. Moore, "Community detection, link prediction, and layer interdependence in multilayer networks," *Physical Review E*, vol. 95, no. 4, p. 042 317, 2017.

[110] E. P. Xing, W. Fu, and L. Song, "A state-space mixed membership blockmodel for dynamic network tomography," *The Annals of Applied Statistics*, vol. 4, no. 2, pp. 535–566, 2010.

[111] Z. Yu, M. Pietrasik, and M. Reformat, "Deep dynamic mixed membership stochastic blockmodel," in *2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, IEEE, 2019, pp. 141–148.

[112] Z. Yu, X. Fan, M. Pietrasik, and M. Z. Reformat, "Fragmentation coagulation based mixed membership stochastic blockmodel," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 6704–6711.

[113] L. Elliott and Y. Teh, “Scalable imputation of genetic data with a discrete fragmentation-coagulation process,” *Advances in neural information processing systems*, vol. 25, 2012.

[114] C. Lee and D. J. Wilkinson, “A review of stochastic block models and extensions for graph clustering,” *Applied Network Science*, vol. 4, no. 1, pp. 1–50, 2019.

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

[116] H. Zhang, L. Qiu, L. Yi, and Y. Song, “Scalable multiplex network embedding.,” in *IJCAI*, vol. 18, 2018, pp. 3082–3088.

[117] H. Song and J. J. Thiagarajan, “Improved deep embeddings for inferencing with multi-layered graphs,” in *2019 IEEE International Conference on Big Data (Big Data)*, IEEE, 2019, pp. 5394–5400.

[118] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[119] S. Wasserman, K. Faust, *et al.*, “Social network analysis: Methods and applications,” 1994.

[120] M Vickers and S Chan, “Representing classroom social structure,” *Victoria Institute of Secondary Education, Melbourne*, 1981.

[121] E. Lazega *et al.*, *The collegial phenomenon: The social mechanisms of cooperation among peers in a corporate law partnership.* Oxford University Press on Demand, 2001.

[122] V Krebs, *Fortune 500 teams*, http://moreno.ss.uci.edu/data.html#krebs.

[123] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi, “The anatomy of a scientific rumor,” *Scientific reports*, vol. 3, no. 1, pp. 1–9, 2013.

[124] A. Sinclair and M. Jerrum, “Approximate counting, uniform generation and rapidly mixing markov chains,” *Information and Computation*, vol. 82, no. 1, pp. 93–133, 1989.

[125] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[126] A. Sanfeliu and K.-S. Fu, “A distance measure between attributed relational graphs for pattern recognition,” *IEEE transactions on systems, man, and cybernetics*, no. 3, pp. 353–362, 1983.

[127] F. A. Saad and V. K. Mansinghka, “Hierarchical infinite relational model,” in *Uncertainty in Artificial Intelligence*, PMLR, 2021, pp. 1067–1077.

[128] J. Paisley, C. Wang, D. M. Blei, and M. I. Jordan, “Nested hierarchical dirichlet processes,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 2, pp. 256–270, 2014.

[129] A. Bonifati, S. Dumbrava, and N. Mir, "Hierarchical clustering for property graph schema discovery.," in *EDBT*, 2022, pp. 2–449.

[130] K. Toutanova and D. Chen, "Observed versus latent features for knowledge base and text inference," in *Proceedings of the 3rd workshop on continuous vector space models and their compositionality*, 2015, pp. 57–66.

[131] N. Jain, J.-C. Kalo, W.-T. Balke, and R. Krestel, "Do embeddings actually capture knowledge graph semantics?" In *European Semantic Web Conference*, Springer, 2021, pp. 143–159.

[132] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

[133] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[134] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical association*, vol. 66, no. 336, pp. 846–850, 1971.

[135] P. Yao and D. Barbosa, "Typing errors in factual knowledge graphs: Severity and possible ways out," in *Proceedings of the Web Conference 2021*, 2021, pp. 3305–3313.

[136] T. Salimans, D. Kingma, and M. Welling, "Markov chain monte carlo and variational inference: Bridging the gap," in *International conference on machine learning*, PMLR, 2015, pp. 1218–1226.

[137] D. M. Blei and M. I. Jordan, "Variational inference for dirichlet process mixtures," *Bayesian analysis*, vol. 1, no. 1, pp. 121–143, 2006.

[138] C. Wang and D. Blei, "Variational inference for the nested chinese restaurant process," *Advances in Neural Information Processing Systems*, vol. 22, 2009.

[139] J. Chen, J. Zhu, J. Lu, and S. Liu, "Scalable inference for nested chinese restaurant process topic models," *arXiv preprint arXiv:1702.07083*, 2017.

[140] M. Zhou, "Infinite edge partition models for overlapping community detection and link prediction," in *Artificial intelligence and statistics*, PMLR, 2015, pp. 1135–1143.

[141] P. Rai, C. Hu, R. Henao, and L. Carin, "Large-scale bayesian multi-label learning via topic-based label embeddings," *Advances in neural information processing systems*, vol. 28, 2015.

[142] X. Fan, B. Li, C. Li, S. SIsson, and L. Chen, "Scalable deep generative relational model with high-order node dependence," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

Appendix A: Probability Mass and Density Functions

In this appendix, we provide the probability mass and density functions for the distributions used in our paper. Probability mass functions capture the probability of a discrete random variable realizing a value, denoted x:

$$\text{Bernoulli}(p, q) = p^x q^{(n-x)} \tag{A.1}$$

$$\text{Multinomial}(\mathbf{p}) = \frac{\Gamma(\sum_i x_i + 1))}{\prod_i \Gamma(x_i + 1))} \prod_i^L p_i^{x_i} \tag{A.2}$$

Where p, q, and $\mathbf{p}$ are the parameters of their respective distributions. Probability density functions capture the relative likelihood of a continuous random variable realizing the value x:

$$\text{Beta}(\alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{B}(\alpha, \beta)}, \mathrm{B}(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{A.3}$$

$$\text{Dirichlet}(\boldsymbol{\alpha}, L) = \frac{\Gamma(\sum_{l=1}^L \alpha_l)}{\prod_{l=1}^L \Gamma(\alpha_l)} \prod_{l=1}^L x_l^{\alpha_l - 1} \tag{A.4}$$

$$\tag{A.5}$$

Where α, β, $\boldsymbol{\alpha}$, and L are the parameters of their respective distributions.

Appendix B: Integral Form of the Beta Function

In this appendix, we provide the derivation to obtain the integral form of the Beta function. We do this by leveraging the definition of the Beta distribution. Specifically, we begin with the identity that the integral of a probability density function with respect to its support is equal to 1 and proceed with simple integral calculus:

$$\begin{aligned}
\int_0^1 \mathrm{Beta}(\alpha, \beta)\ \mathrm{d}x &= 1 \\
\int_0^1 \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{B}(\alpha, \beta)}\ \mathrm{d}x &= 1 \\
\frac{1}{\mathrm{B}(\alpha, \beta)} \int_0^1 x^{\alpha-1}(1-x)^{\beta-1}\ \mathrm{d}x &= 1 \\
\int_0^1 x^{\alpha-1}(1-x)^{\beta-1}\ \mathrm{d}x &= \mathrm{B}(\alpha, \beta) \qquad \text{(B.1)}
\end{aligned}$$

Appendix C: Marginalizing Finite Level Memberships

In order to marginalize finite level memberships, we begin with the definition of its posterior, $\mathbb{P}(\mathbf{a}_i \mid \mathbf{A}_{-i}, \mathbf{Z}, \boldsymbol{\alpha})$, which is defined analogously to Equation 6.7 with the exception that the Dirichlet prior is used in the this case. Formally, we obtain the following through Bayes' rule:

$$\mathbb{P}(\mathbf{a}_i \mid \mathbf{A}_{-i}, \mathbf{Z}, \boldsymbol{\alpha}) = \frac{\mathbb{P}(\mathbf{Z} \mid \mathbf{A}, \boldsymbol{\alpha})\mathbb{P}(\mathbf{a_i} \mid \mathbf{A_{-i}}, \boldsymbol{\alpha})}{\int_{\mathbf{a}_i} \mathbb{P}(\mathbf{Z} \mid \mathbf{A}, \boldsymbol{\alpha})\mathbb{P}(\mathbf{a_i} \mid \mathbf{A_{-i}}, \boldsymbol{\alpha})\mathrm{d}\mathbf{a_i}} \tag{C.1}$$

$$= \frac{\text{Multinomial}(\mathbf{a}_i)\text{Dirichlet}(\boldsymbol{\alpha}, L)}{\int_{\mathbf{a}_i} \text{Multinomial}(\mathbf{a}_i)\text{Dirichlet}(\boldsymbol{\alpha}, L)\ \mathrm{d}\mathbf{a}_i} \tag{C.2}$$

Where $\boldsymbol{\alpha}$ is a vector of L concentration parameters for each level in the distribution, namely $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_L]$ such that $\alpha_l > 0$ and L is the finite number of levels in the hierarchy. In our marginalization, we adopt the notation from Equation 6.8 to indicate the number of indicators in $\mathbf{z}_{i*}$. Furthermore, we define the following vector of concentration parameters to aid in readability: $\boldsymbol{\alpha'} = [\alpha_1 + \#^{\mathbf{z}_{i*}=1}, \alpha_2 + \#^{\mathbf{z}_{i*}=2}, ..., \alpha_L + \#^{\mathbf{z}_{i*}=L}]$. With these variables in place, we can derive the Dirichlet posterior for finite level indicators:

$$\mathbb{P}(\mathbf{a}_i \mid \mathbf{A}_{-i}, \mathbf{Z}_{i*}, \boldsymbol{\alpha})$$

$$= \frac{\text{Multinomial}\big(\mathbf{a}_i\big)\text{Dirichlet}(\boldsymbol{\alpha})}{\int_{\mathbf{a}_i} \text{Multinomial}\big(\mathbf{a}_i\big)\text{Dirichlet}(\boldsymbol{\alpha})\ \mathrm{d}\mathbf{a}_i}$$

$$\overset{(1)}{=} \frac{\left(\frac{\Gamma(\sum_{l=1}^{L} \#^{\mathbf{z}_{i*}=l} + 1)}{\prod_{l=1}^{L} \Gamma(\#^{\mathbf{z}_{i*}=l} + 1)} \prod_{l=1}^{L} (a_i^l)^{\#^{\mathbf{z}_{i*}=l}}\right)\left(\frac{\Gamma(\sum_{l=1}^{L} \alpha_l)}{\prod_{l=1}^{L} \Gamma(\alpha_l)} \prod_{l=1}^{L} (a_i^l)^{\alpha_l - 1}\right)}{\int_{\mathbf{a}_i} \left(\frac{\Gamma(\sum_{l=1}^{L} \#^{\mathbf{z}_{i*}=l} + 1)}{\prod_{l=1}^{L} \Gamma(\#^{\mathbf{z}_{i*}=l} + 1)} \prod_{l=1}^{L} (a_i^l)^{\#^{\mathbf{z}_{i*}=l}}\right)\left(\frac{\Gamma(\sum_{l=1}^{L} \alpha_l)}{\prod_{l=1}^{L} \Gamma(\alpha_l)} \prod_{l=1}^{L} (a_i^l)^{\alpha_l - 1}\right) \mathrm{d}\mathbf{a}_i}$$

$$
\begin{aligned}
&\overset{(2)}{=} \frac{\left(\frac{\Gamma(\sum_{l=1}^{L} \#^{\mathbf{z}_{i*}=l}+1)}{\prod_{l=1}^{L}\Gamma(\#^{\mathbf{z}_{i*}=l}+1)}\frac{\prod_{l=1}^{L}\Gamma(\alpha_l)}{\Gamma(\sum_{l=1}^{L}\alpha_l)}\right)\left(\frac{\prod_{l=1}^{L}\Gamma(\alpha'_l)}{\Gamma(\sum_{l=1}^{L}\alpha'_l)}\frac{\Gamma(\sum_{l=1}^{L}\alpha'_l)}{\prod_{l=1}^{L}\Gamma(\alpha'_l)}\right)\prod_{l=1}^{L}(a_i^l)^{\alpha'_l-1}}{\int_{\mathbf{a}_i}\left(\frac{\Gamma(\sum_{l=1}^{L} \#^{\mathbf{z}_{i*}=l}+1)}{\prod_{l=1}^{L}\Gamma(\#^{\mathbf{z}_{i*}=l}+1)}\frac{\prod_{l=1}^{L}\Gamma(\alpha_l)}{\Gamma(\sum_{l=1}^{L}\alpha_l)}\right)\left(\frac{\prod_{l=1}^{L}\Gamma(\alpha'_l)}{\Gamma(\sum_{l=1}^{L}\alpha'_l)}\frac{\Gamma(\sum_{l=1}^{L}\alpha'_l)}{\prod_{l=1}^{L}\Gamma(\alpha'_l)}\right)\prod_{l=1}^{L}(a_i^l)^{\alpha'_l-1}\,\mathrm{d}\mathbf{a}_i}\\
&\overset{(3)}{=} \frac{\left(\frac{\Gamma(\sum_{l=1}^{L} \#^{\mathbf{z}_{i*}=l}+1)}{\prod_{l=1}^{L}\Gamma(\#^{\mathbf{z}_{i*}=l}+1)}\frac{\prod_{l=1}^{L}\Gamma(\alpha_l)}{\Gamma(\sum_{l=1}^{L}\alpha_l)}\frac{\prod_{l=1}^{L}\Gamma(\alpha'_l)}{\Gamma(\sum_{l=1}^{L}\alpha'_l)}\right)\mathrm{Dirichlet}(\boldsymbol{\alpha'})}{\int_{\mathbf{a}_i}\left(\frac{\Gamma(\sum_{l=1}^{L} \#^{\mathbf{z}_{i*}=l}+1)}{\prod_{l=1}^{L}\Gamma(\#^{\mathbf{z}_{i*}=l}+1)}\frac{\prod_{l=1}^{L}\Gamma(\alpha_l)}{\Gamma(\sum_{l=1}^{L}\alpha_l)}\frac{\prod_{l=1}^{L}\Gamma(\alpha'_l)}{\Gamma(\sum_{l=1}^{L}\alpha'_l)}\right)\mathrm{Dirichlet}(\boldsymbol{\alpha'})\,\mathrm{d}\mathbf{a}_i}\\
&\overset{(4)}{=} \frac{\left(\frac{\Gamma(\sum_{l=1}^{L} \#^{\mathbf{z}_{i*}=l}+1)}{\prod_{l=1}^{L}\Gamma(\#^{\mathbf{z}_{i*}=l}+1)}\frac{\prod_{l=1}^{L}\Gamma(\alpha_l)}{\Gamma(\sum_{l=1}^{L}\alpha_l)}\frac{\prod_{l=1}^{L}\Gamma(\alpha'_l)}{\Gamma(\sum_{l=1}^{L}\alpha'_l)}\right)\mathrm{Dirichlet}(\boldsymbol{\alpha'})}{\left(\frac{\Gamma(\sum_{l=1}^{L} \#^{\mathbf{z}_{i*}=l}+1)}{\prod_{l=1}^{L}\Gamma(\#^{\mathbf{z}_{i*}=l}+1)}\frac{\prod_{l=1}^{L}\Gamma(\alpha_l)}{\Gamma(\sum_{l=1}^{L}\alpha_l)}\frac{\prod_{l=1}^{L}\Gamma(\alpha'_l)}{\Gamma(\sum_{l=1}^{L}\alpha'_l)}\right)\int_{\mathbf{a}_i}\mathrm{Dirichlet}(\boldsymbol{\alpha'})\,\mathrm{d}\mathbf{a}_i}\\
&\overset{(5)}{=} \mathrm{Dirichlet}(\boldsymbol{\alpha'}) \qquad \text{(C.3)}
\end{aligned}
$$

Where (1) is obtained by applying the definitions of the Multinomial and Dirichlet distributions as per Equations A.2 and A.4, respectively; (2) leverages the definition of $\boldsymbol{\alpha'}$ to group level memberships and introduces cancelling numerator and denominator terms using $\boldsymbol{\alpha'}$ to obtain a Dirichlet probability density function as shown by replacement in (3); (4) groups terms constant with respect to $\mathbf{a}_i$ in integration; and (5) leverages the law of total probability.

Appendix D: Simplifying Level Likelihood

In this appendix, we provide the simplification of level likelihood by eliminating the Gamma function for more efficient computation. Recall from Equation 6.19 that the level likelihood, $\mathbb{P}(\mathbf{g}_{ij*} \mid \mathbf{G}_{-(ij*)}, \mathbf{P}, \mathbf{Z}, \lambda, \eta)$, is expressed as follows:

$$\begin{aligned}
&\mathbb{P}(\mathbf{g}_{ij*} \mid \mathbf{G}_{-(ij*)}, \mathbf{P}, \mathbf{Z}, \lambda, \eta) \\
&= \prod_{g_{ijr} \in \mathbf{g}_{ij*}} \frac{\Gamma(\#_{-(ijr)}^{c_{pqr}=1} + g_{ijr} + \lambda)\Gamma(\#_{-(ijr)}^{c_{pqr}=0} + (1 - g_{ijr}) + \eta)\Gamma(\#_{-(ijr)}^{c_{pqr}=1} + \#_{-(ijr)}^{c_{pqr}=0} + \lambda + \eta)}{\Gamma(\#_{-(ijr)}^{c_{pqr}=1} + \#_{-(ijr)}^{c_{pqr}=0} + 1 + \lambda + \eta)\Gamma(\#_{-(ijr)}^{c_{pqr}=1} + \lambda)\Gamma(\#_{-(ijr)}^{c_{pqr}=0} + \eta)} \\
&= \prod_{g_{ijr} \in \mathbf{g}_{ij*}} \frac{1}{\#_{-(ijr)}^{c_{pqr}=1} + \#_{-(ijr)}^{c_{pqr}=0} + \lambda + \eta} \frac{\Gamma(\#_{-(ijr)}^{c_{pqr}=1} + g_{ijr} + \lambda)\Gamma(\#_{-(ijr)}^{c_{pqr}=0} + (1 - g_{ijr}) + \eta)}{\Gamma(\#_{-(ijr)}^{c_{pqr}=1} + \lambda)\Gamma(\#_{-(ijr)}^{c_{pqr}=0} + \eta)}
\end{aligned} \tag{D.1}$$

We can leverage the fact that $g_{ijr} \in \{0, 1\}$ to define the second term as a piecewise function with respect to the value of g_{ijr}. Doing so allows us to cancel out terms which appear in both the numerator and denominator after expanding the Gamma function. This process leads to the following simplification:

$$\frac{\Gamma(\#_{-(ijr)}^{c_{pqr}=1} + g_{ijr} + \lambda)\Gamma(\#_{-(ijr)}^{c_{pqr}=0} + (1 - g_{ijr}) + \eta)}{\Gamma(\#_{-(ijr)}^{c_{pqr}=1} + \lambda)\Gamma(\#_{-(ijr)}^{c_{pqr}=0} + \eta)} = \begin{cases} \#_{-(ijr)}^{c_{pqr}=1} + \lambda & g_{ijr} = 1 \\ \#_{-(ijr)}^{c_{pqr}=0} + \eta & g_{ijr} = 0 \end{cases} \tag{D.2}$$

This allows us to put together Equations D.1 and D.2 to get the following, as seen in Equation 6.19:

$$\mathbb{P}(\mathbf{g}_{ij*} \mid \mathbf{G}_{-(ij*)}, \mathbf{P}, \mathbf{Z}, \lambda, \eta) = \prod_{g_{ijr} \in \mathbf{g}_{ij*}} \frac{g_{ijr}(\#_{-(ijr)}^{c_{pqr}=1} + \lambda) + (1 - g_{ijr})(\#_{-(ijr)}^{c_{pqr}=0} + \eta)}{\#_{-(ijr)}^{c_{pqr}=1} + \#_{-(ijr)}^{c_{pqr}=0} + \lambda + \eta} \tag{D.3}$$