Learning and Inference over Relational Data

Ralph Abboud

Jesus College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2022

Acknowledgements

I would first like to thank my supervisors Dr. İsmail İlkan Ceylan and Prof. Thomas Lukasiewicz for their consistent support throughout my DPhil. Without them, this work would not be possible. I am also very grateful to Jesus College, the University of Oxford, and to DeepMind for funding my research, and allowing me to pursue my dreams over the last four years. I hope that this work repays the faith you have placed in me. I am also very grateful to my colleagues at the Computer Science department Eleonora Giunchiglia, Vid Kocijan, and Tommaso Salvatori, for the many wonderful moments and memories we have shared, even when apart during the pandemic. Your presence has made my DPhil experience more complete, and I will always value the bonds we have made.

On a more personal level, I would like to thank my partner Rayane for supporting me and motivating me over the last three years. Even from a distance, she made dealing with deadlines, papers and presentations so much easier, and helped me past very challenging moments during the pandemic and beyond. I am also grateful to my sister Rana, her husband Ewen, and my lovely niece Ella, whose smile and positive attitude to life continue to inspire me every day. Finally, I owe this thesis, and everything else, to my wonderful parents Layla and Edmond, who have made me the man I am today. You may not be as close to me as I would like, but you continue to motivate me and push me to improve, just by being there. Thank you for everything. I hope I can always make you proud.

Abstract

Relational data is ubiquitous in modern-day computing, and drives several key applications across multiple domains, such as information retrieval, question answering, recommendation systems, and drug discovery. As a result, a main research question in artificial intelligence (AI) is to build models that exploit relational data in an efficient and reliable way, while injecting the relevant inductive biases and being robust to input noise. In recent years, neural models such as graph neural networks (GNNs) and shallow node embedding models have enabled significant breakthroughs in learning representations over relational structures. However, the abilities and limitations of these systems are not completely understood, and several challenges remain to endow these models with reliability guarantees, enrich their relational inductive bias, and apply them in more challenging problem settings. In this thesis, we study learning and inference over relational data. More specifically, we analyse the properties and limitations of existing models both theoretically and empirically, and propose new approaches with improved relational inductive bias and representation power.

Publications

This thesis is based on my following publications [6, 3, 2, 4, 5]:

1. Ralph Abboud, İsmail İlkan Ceylan, and Thomas Lukasiewicz. Learning to Reason: Leveraging Neural Networks for Approximate DNF Counting. AAAI 2020.

2. Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. BoxE: A Box Embedding Model for Knowledge Base Completion. NeurIPS 2020.

3. Ralph Abboud, İsmail İlkan Ceylan, and Radoslav Dimitrov. On the Approximability of Weighted Model Integration over DNF Structures. KR 2020.

4. Ralph Abboud, İsmail İlkan Ceylan, and Radoslav Dimitrov. Approximate Weighted Model Integration on DNF Structures. AIJ 2022.

5. Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe and Thomas Lukasiewicz. The Surprising Power of Graph Neural Networks with Random Node Initialization. IJCAI 2021.

as well as the following paper [1], currently under review:

6. Ralph Abboud and İsmail İlkan Ceylan. Node Classification Meets Link Prediction on Knowledge Graphs. arXiv preprint 2106.07297

More recently, I have co-authored the following papers [7, 122], which are not included in my thesis:

7. Ralph Abboud, Radoslav Dimitrov, and İsmail İlkan Ceylan. Shortest Path Networks for Graph Property Prediction. arXiv preprint 2206.01003.

8. Johannes Messner, Ralph Abboud, and İsmail İlkan Ceylan. Temporal Knowledge Graph Completion using Box Embeddings. AAAI 2022.

Contents

List of Figures

Chapter 1

Introduction

Deep learning systems, powered by neural networks, have achieved breakthrough results on a variety of challenging tasks, such as computer vision [96] and machine translation [160]. Deep learning models learn patterns from data with minimal human intervention, and empirically generalise well beyond their training set. Therefore, there has been an increasing interest to apply deep learning systems across several domains. Along these lines, a prominent research frontier in recent years is to apply deep learning to *relational data*.

Fundamentally, relational data represents information as a set of *entities* connected by semantically meaningful *relations*. For example, one can represent products, sellers and users on an online marketplace as entities, and depict transactions as a ternary relation across the three aforementioned entity types, e.g., Alice purchases a ball from Charlie. A popular special case of relational data are *graph* structures, where relations are at most binary. In this case, the relations can be seen as defining (labelled) *edges* between the entities of the graph, which themselves constitute the graph *nodes*.

Relational representations are very general, and appear across various application domains. For instance, users in social networks, connected pairwise based on their interactions (friendship, following, likes), can be seen as a graph structure. The same applies to papers in citation networks [153, 154] and their citation connections, as well as molecules, where atoms can be seen as entities and their bonds can be represented as binary relations. In fact, relational data encapsulates several traditional data domains. For instance, images are a special case of a graph in the shape of a grid, where adjacent pixels are connected by an edge, and sequences are a series of entities with edges connecting consecutive entities.

Given the ubiquity of relational data and the prevalence of graph structures, building powerful relational machine learning models is an important research question,

with ramifications across several tasks, such as information retrieval [182], question answering [20], recommendation systems [173], and drug discovery [60]. Broadly speaking, machine learning tasks over graphs can be divided into three main categories:

1. **Node-level tasks.** Given an input graph with unlabelled or partly labelled nodes, node-level tasks aim to predict node properties, e.g., a class or a value, for nodes without a pre-labelled property. For instance, predicting the topic of a paper in a citation network, where papers (the entities in the input graph) have content features and are connected to other papers through binary citation relations, is a node classification task.

2. **Graph-level tasks.** Given an input graph, graph-level tasks seek to predict a *global* graph property, such as a class or value, based on node features, edges, and the overall input graph structure. These tasks are very prominent over, e.g., molecular graphs, and include several graph property prediction problems, such as toxicity classification and zero-point vibrational energy (ZPVE) regression [140].

3. **Edge-level tasks.** Given an input graph, edge-level tasks aim to predict unknown edge properties over existing edges or, more commonly, to predict *missing* edges in the graph based on existing edges and node features. In the latter case, the problem is known as *link prediction* when the input graph is single-relational, e.g., citation networks, and as *knowledge graph completion* (KGC), when the input graph is multi-relational, e.g., knowledge graphs.

Unlike other data domains (images, audio, etc.), graphs (and relational structures in general) do not necessarily follow a regular structure (pixel grid, sequence of samples, respectively). Indeed, distinct input graphs can have highly different characteristics in terms of number of nodes, relational structure, etc. Therefore, machine learning models over graphs require dedicated approaches that can exploit distinct relational configurations. Moreover, graphs naturally encode *dependencies* between nodes through their edges, which imposes additional learning constraints. Finally, graphs, as with all other data domains, require certain *inductive biases* to be captured by machine learning models so as to effectively fit the training data and generalise. For instance, machine learning models must be invariant to permutations of nodes in the graph, so as not to overfit on a specific arbitrary ordering of graph nodes and generalise poorly. Furthermore, machine learning models must effectively

exploit the relational structure of the graph, and build on this to make more principled predictions.

Initially, the first machine learning approaches over graphs extracted scalar *features* from input graphs and subsequently applied conventional models to learn a mapping to the output space. A simple example of this approach is extracting node *degrees*, i.e., the number of edges connected to every node, and using the distribution of node degrees as a set of features for a standard machine learning model. Common choices of features include graph statistics such as centrality measures, as well as several graph *kernels* [177, 22], commonly used for solving the *graph isomorphism* problem, i.e., checking whether two graphs are identical modulo node permutations, or *isomorphic*. However, this approach is limited in many ways. First, graph features have limited *expressive* power, as they cannot distinguish between different, i.e., non-isomorphic, input graphs, and thus erroneously map distinct graphs to identical outputs. For example, node degree extraction is problematic, as several highly distinct graphs can yield identical degree distributions. Second, feature extraction does not provide the machine learning model access to the graph structure, leading to a loss of information and poor inductive bias. Finally, the set of features extracted is manually selected and not learned, which limits the applicability of this approach.

In light of these challenges, an increasing amount of research has been dedicated towards developing machine learning approaches that directly apply to graph structures (and relational data more generally) and autonomously learn the features to be extracted. In particular, the current main paradigm for machine learning over graphs is to automatically learn (latent) vector representations, or *embeddings*, for graph nodes and edges that compactly represent essential information across the graph. More specifically, a learnable function, called the *encoder*, is applied to the input graph to map it to representative node and edge embeddings, and another learnable function, the *decoder*, applies on these learned embeddings and maps them to the target output. This framework is trained end-to-end using gradient descent [91], and offers a compelling means to overcome the manual feature selection of traditional methods. Moreover, it explicitly computes representations for graph components, which offers a strong inductive bias, and learns latent features that ideally describe these components. In particular, entities and relations with similar embeddings ideally must perform similar roles in the graph [73].

The encoder-decoder setup is a promising line of research for overcoming the limitations of standard feature extraction techniques. However, the same challenges, namely expressive power and inductive bias, also arise when it comes to designing

and selecting the corresponding encoder and decoder components. In particular, when building machine learning models to learn embeddings over graphs, a key problem is to define encoders and decoders that (i) are sufficiently powerful so as to distinguish any non-isomorphic pair of graphs and map them to distinct outputs, and (ii) can fully exploit the graph structure and explicitly build on it so as to generalise effectively. As a result, several families of graph machine learning models have been developed, each addressing these challenges to varying degrees of success - for a broader coverage on graph representation learning, we refer the reader to the standard literature [73].

Among machine learning approaches over graphs, the most successful methods currently in the literature are (i) shallow *node embedding models* [21, 166], which explicitly learn latent representations for entities (corresponding to graph nodes) and relations over graphs, and use these representations to make predictions, and (ii) *graph neural networks (GNNs)* [149, 67], which apply a series of permutation-invariant (respectively, equivariant) computations on input graphs to learn functions explicitly respecting the graph structure. Both classes of models have enabled breakthrough results over graph-structured data across multiple graph-level, node-level and edge-level tasks.

Shallow node embedding models explicitly assign embedding representations to the entities and relations of an input graph, and optimise these representations with respect to a given *scoring function* (which can be seen as decoder in our terminology). These models are typically applied to multi-relational graphs, such as knowledge graphs (KGs), e.g., YAGO [113], Freebase [19], where entities are connected through multiple different relation types.

In shallow node embedding models, entity and relation representations are typically passed through a scoring function to predict the likelihood of edges existing in the graph. More specifically, entity and relation embeddings are *trained* on the graph structure, such that existing edges are mapped to better scores than randomly sampled unknown edges. The explicit representation of entities and relations offers node embedding models a strong inductive bias, and yields invariance with respect to node orderings in the graph. Furthermore, the scoring function offers a principled means to include and/or learn *relational patterns* to improve generalisation. For instance, one can naively impose a *relational hierarchy*, e.g., any two *friends* necessarily *know* one another.

Though shallow node embedding models offer several advantages, they also suffer from major limitations. First, many embedding models have limited expressive power, and thus fail to represent arbitrary input graphs, i.e., there exist graphs which no

parametrisation of a given embedding model can accurately represent. Second, most shallow node embedding models have insufficient inductive capacity relative to jointly capturing combinations of relational patterns, e.g., hierarchies, symmetries, which hinders their generalisation in real-world data settings. In particular, models which fail to learn parameters that satisfy prominent inference patterns will fail to effectively apply these patterns on new, unseen data. Third, embedding models are *transductive*, in that they can only make predictions regarding an entity or a relation if it has been *previously observed* in training, and thus must explicitly retrain on unseen entities before making predictions. Finally, most embedding models are primarily designed for graphs, and cannot naturally apply to more general relational data, e.g., higher-arity knowledge bases, which further limits their applicability.

Graph neural networks (GNNs) [149, 67] are neural networks designed to operate on graph-structured data, and naturally encode key structural biases, e.g., node permutation invariance. Among GNNs, the most popular family of models is message passing neural networks (MPNNs) [62, 15], which perform multiple iterations of message passing between a node and its *direct neighbourhood*, i.e., the set of adjacent nodes directly connected by an edge, to learn and refine representations in the input graph and subsequently learn functions over graphs.

Unlike shallow node embedding models, MPNNs learn a transformation over the input graph, and can apply to unseen nodes, i.e., they can be *inductive*. Moreover, MPNNs explicitly use all existing edges in the input graph to learn representations, and thus directly build on the overall graph structure. However, MPNNs also struggle with multiple limitations. Indeed, MPNNs have limited expressive power [127, 183] and fail to distinguish even simple pairs of non-isomorphic graphs. Therefore, they cannot learn any arbitrary function over graph-structured data. Second, MPNNs have limited inductive biases in the multi-relational setting, which limits their ability to capture prominent relational properties and generalise in rich relational contexts, e.g., knowledge graphs. Third, the message passing procedure in MPNNs suffers from *oversquashing* when applied repeatedly in deeper models, as it excessively squashes information arriving from long-range dependencies in the graph [8], and ultimately returns less informative node representations with more iterations [100], a problem known as *oversmoothing*. This procedure is also very specialised to graphs, and is very challenging to generalise to more general relational structures. Finally, MPNNs, by construction, only use existing edges during message passing, which can lead to poor learning when the input data is noisy or incomplete.

In this thesis, we study learning and inference over relational data, both over graph structures and more general relational data, and propose several models and frameworks, supported with theoretical analyses and results, to improve on the relational inductive bias and representation power of models in the field. More specifically, we systematically study existing models, prove theoretical properties and results about them, and propose extensions, as well as novel models, to (i) provably capture and/or impose rich relational inductive biases, (ii) develop a better understanding of the expressiveness and representational limitations of existing models, and (iii) extend existing models and approaches to novel, challenging application domains pertaining to reasoning and inference. The contributions of this thesis can be summarised as follows:

- In Chapter 2, we discuss key preliminaries that are used throughout the thesis. More specifically, we formally describe relational data and knowledge graphs, logic, shallow node embedding models, and graph neural networks.

- In Chapter 3, we propose a new shallow node embedding model, BoxE, that uses region-based box embeddings to represent relations in a knowledge graph (resp., knowledge base). We show that BoxE is a fully expressive model, and thus can represent arbitrary configurations for a given knowledge graph (resp., knowledge base). Moreover, we show that BoxE can provably capture and inject a rich language of relational *inference patterns*, and thus offers a very strong relational inductive bias.

- In Chapter 4, we propose a unifying perspective for node classification and link prediction, and train MPNNs and shallow embedding models (extended with feature processing) in this joint setting on increasingly incomplete relational data using a novel dataset, WikiAlumni. Through this study, we observe that shallow embedding models are more robust to incompleteness, and ultimately outperform MPNNs.

- In Chapter 5, we consider a simple, but surprisingly powerful, extension to MPNNs, namely using partially randomised initial node features. We then show that this seemingly trivial change allows MPNNs to become universal approximators of invariant functions over graphs of bounded size. We then conduct a empirical study using carefully designed synthetic datasets EXP and CEXP, and validate this theoretical result in practice.

- In Chapter 6, we design an MPNN for a special class of the weighted model counting (WMC) task, an important task in probabilistic inference, and train this model using an approximate solver. We then empirically show that this MPNN is highly accurate, scalable, and generalises strongly, even on weighted model counting instances larger than those encountered in training.
- In Chapter 7, we study a generalisation of weighted model counting to the hybrid discrete-continuous domain, known as weighted model integration (WMI), and propose a polynomial-time approximation algorithm, ApproxWMI, for an important sub-class of WMI problems. ApproxWMI therefore enables large-scale probabilistic inference, and thus paves the way for more comprehensive neural approaches for WMI.
- In Chapter 8, we summarise the main findings of this thesis, discuss future perspectives and interesting directions, and conclude.

Chapter 2

Background

In this chapter, we present preliminaries used throughout this thesis, namely relational data and knowledge graphs, propositional and first-order logic, shallow node embedding models, and graph neural networks (GNNs).

2.1 Relational Data and Knowledge Graphs

Graphs. We denote a graph by $G(V, E)$, where V denotes the set of nodes (or alternatively, vertices) in the graph, and $E \subseteq V \times V$ is the set of edges connecting pairs of nodes. A graph is *undirected* when all edges are bidirectional, i.e., if $(v_1, v_2) \in E$ holds, then $(v_2, v_1) \in E$ also holds, and is *directed* otherwise. A graph is *simple* if it does not include any self-loops. Two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are *isomorphic* if there exists a bijection $f : V_1 \to V_2$ such that an edge $(u, v) \in E_1$ exists if and only if $(f(u), f(v)) \in E_2$ also exists.

When studying functions over graphs, we are primarily interested in *invariance* and *equivariance* properties. We say that a function f is *invariant* if for any two isomorphic graphs G, H, it holds that $f(G) = f(H)$. Furthermore, we say that a node-level function f mapping a graph G with vertices V to vectors $\boldsymbol{x} \in \mathbb{R}^{|V|}$ is *equivariant* if, for every permutation π of V, it holds that $f(G^\pi) = f(G)^\pi$.

Knowledge graphs. We also consider *multi-relational* graphs, which include multiple *edge types*. More specifically, a multi-relational graph can be seen as a tuple $(V, E_1, \ldots, E_r)$, where $E_1, \ldots, E_r$ correspond to r distinct relation types. Multi-relational graphs are common in practice. For instance, the different bonds in a molecular graph, as well as different interactions in a social network, are usually represented with typed edges. Moreover, *knowledge graphs* (KG), a very prominent data source in modern-day computing, correspond to multi-relational directed graphs.

In knowledge graphs, domain knowledge about a finite set of entities is represented through a series of typed edges indicating semantic connections between pairs of entities. For instance, one can construct a geographical knowledge graph, in which entities represent nations, cities, towns, etc., and describe connections between them through relations, e.g., a nation has a capital city, a nation borders another nation. Nowadays, many large-scale KGs, such as YAGO [113], NELL [124], Knowledge Vault [51], and Freebase [19], are publicly available and include hundreds of millions of edges. These KGs are heavily used in a wide range of applications, such as query answering [76], information retrieval [182] and recommender systems [173], owing to their structured knowledge representation [175].

Relational data. A *relational vocabulary* consists of a finite set $\mathbf{E}$ of *entities* and a finite set $\mathbf{R}$ of *relations*. A *fact* (also called a *ground atom*) is of the form $r(e_1, \ldots, e_n)$, where $r \in \mathbf{R}$ is an n-ary relation, and $e_i \in \mathbf{E}$ are entities. When all relations in $\mathbf{R}$ are restricted to an arity no greater than 2, facts induce a graph over the entities in $\mathbf{E}$. Therefore, one can alternatively view knowledge graphs as a finite set of facts defined using unary and binary relations, such that each binary fact $r(e_1, e_2)$ corresponds to a directed typed edge, and each unary fact $c(e_1)$ corresponds to a node type. Finally, we define a *knowledge base* (KB) as a generalisation of knowledge graphs, where the finite set of facts and their underlying relations can have an arbitrary arity.

2.2 Logic

In this section, we briefly present both propositional logic and first-order logic.

2.2.1 Propositional Logic

Let S be a finite set of propositional variables. Given $p \in S$, a *literal* is of the form p or $\neg p$. A *conjunctive clause* is a conjunction of literals, e.g. $p_1 \wedge \neg p_2$, and a *disjunctive clause* is a disjunction of literals, e.g., $p_1 \vee \neg p_2$. A clause has *width* k if it has exactly k literals. For instance, the earlier examples of a conjunctive and a disjunctive clause both have width 2. A formula φ is in *conjunctive normal form* (CNF) if it is a conjunction of disjunctive clauses, and it is in *disjunctive normal form* (DNF) if it is a disjunction of conjunctive clauses. To illustrate, the formula $\varphi = (p_1 \wedge \neg p_3) \vee (p_4 \wedge p_1)$ is a DNF. We say that a DNF (resp., CNF) has width k if it contains clauses of width at most k. Concretely, the earlier DNF formula has width 2, as both its clauses themselves have width 2. Finally, an *assignment* $\nu : S \mapsto \{0, 1\}$ maps every variable to either 0 (False), or 1 (True). An assignment

ν *satisfies* a propositional formula φ, denoted $\nu \models \varphi$, in the usual sense, where $\models$ is the propositional entailment relation. The *satisfiability problem* (SAT) is the task of deciding whether a given propositional formula φ has a satisfying assignment.

2.2.2 First-Order Logic

Syntax. We consider a relational vocabulary σ which consists of (potentially infinite) pairwise disjoint sets of relations $\mathbf{R}$, constants $\mathbf{K}$, and variable names $\mathbf{V}$. A *term*, denoted s_i, is either a constant or a variable. An *atom* is of the form $r(s_1, \ldots, s_n)$, where r is an n-ary relation, and $s_1, \ldots, s_n$ are terms. A *ground* atom is an atom without variables, i.e., an atom exclusively using constants.

First-order logic defines the logical connectives of negation, conjunction, and disjunction, denoted by $\neg, \wedge$ and $\vee$, respectively, as well as the existential quantifier ($\exists$) and the universal quantifier ($\forall$). Using these connectives and quantifiers, *formulas* in first-order logic can be inductively defined based on atoms. In particular, formulas in first-order logic can be constructed using the following grammar:

$$\varphi = r(s_1, \ldots, s_n) \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \exists x.\ \varphi \mid \forall x.\ \varphi,$$

where $r \in \mathbf{R}$ is an n-ary relation, $s_1, \ldots, s_n$ are terms, and $x \in \mathbf{V}$ is a variable.

Within a first-order formula, we say that a variable is *quantified* if it is in the scope of a quantifier ($\exists, \forall$). Otherwise, the variable is *free*. A *sentence* is a formula where all variables are quantified. For ease of notation, we denote sentences as φ, and denote formulas with free variables $x_1, \ldots, x_k$ as $\varphi(x_1, \ldots, x_k)$.

In what follows, we only consider the constructors $\neg, \exists, \wedge$ to define the semantics of first-order logic, as the other constructors ($\vee, \forall$) can be written in terms of the first three. More formally:

$$\forall x.\varphi \equiv \neg(\exists x.(\neg\varphi)), \varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi).$$

Furthermore, *logical implication*, denoted by $\rightarrow$, can equivalently be written as $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$. We therefore use $\rightarrow$ for ease of notation.

Semantics. A first-order interpretation is a pair $I = (\Delta^I, \cdot^I)$, where Δ^I is a non-empty domain of elements, and $\cdot^I$ is an *interpretation function*.

Given an interpretation I over a domain, the interpretation function maps every constant to an element in Δ^I, and maps every n-ary relation r to a subset of $(\Delta^I)^n$, i.e., the n^{th} power set of Δ^I consisting of n-element subsets of Δ^I. Furthermore, a *variable assignment* τ maps variables to domain elements. We can therefore define mappings for constants, variables and relations using I and τ as follows:

- $\kappa^{I,\tau} = \kappa^I$ for all constants $\kappa \in \mathbf{K}$.
- $x^{I,\tau} = \tau(x)$ for all variables $x \in \mathbf{V}$.
- $r^{I,\tau} = r^I$ for all relations $r \in \mathbf{R}$.

The first-order entailment relation is defined inductively, as follows:

- $I, \tau \models r(s_1, \ldots, s_n)$ if $(s_1^{I,\tau}, \ldots, s_n^{I,\tau}) \in r^{I,\tau}$,
- $I, \tau \models \neg\varphi(s_1, \ldots, s_n)$ if $I, \tau \not\models \varphi(s_1, \ldots, s_n)$,
- $I, \tau \models \varphi(s_1, \ldots, s_n) \wedge \psi(s_1, \ldots, s_n)$ if $I, \tau \models \varphi(s_1, \ldots, s_n)$ and $I, \tau \models \psi(s_1, \ldots, s_n)$,
- $I, \tau \models \exists.y\ \varphi(s_1, \ldots, s_n)$ if there exists $e \in \Delta^I$ such that $I, \tau' \models \varphi(s_1, \ldots, s_n)$, where τ' is defined analogously to τ, except that $\tau'(y) = e$.

For sentences φ (where the variable assignment τ does not apply), we say that the interpretation I is a *model* of φ if $I \models \varphi$. If the interpretation domain is finite, we are in the context of finite structures [103], which is the main focus of this work.

2.3 Shallow Node Embedding Models

2.3.1 Overview

At a high level, shallow node embedding models learn *representations* for entities and relations in a given knowledge graph. These representations are then combined to compute *scores* for facts in the KG, which indicate the *plausibility* of a fact, i.e., its likelihood of being true. More concretely, given a KG with a relation r and entities e_1, e_2, shallow node embedding models define embeddings $\mathbf{r}$, $\mathbf{e_1}$ and $\mathbf{e_2}$ respectively in an embedding space, e.g., $\mathbb{R}^d$, and then use these embeddings to compute fact scores, denoted $\mathsf{score}(r(e_1, e_2))$, where score denotes an embedding model's *scoring function*.

Within these models, embedding values are learned by training on existing facts, such that score maps known facts to high plausibility values, and maps *negatively sampled* facts, assumed to be false, to low plausibility values. The latter step is essential, as otherwise embedding models would not receive any negative signal and can easily learn to map the universe of all possible facts to "True".

Negative sampling. KGs typically only include known true facts, and thus do not include false facts to use in training. Hence, it is necessary to make assumptions about fact falsehood in the input KG, and to use such "false" facts to achieve a balanced

training procedure. A common assumption used to approximate false facts is to use the *local closed-world assumption* (LCWA), which assumes that all facts of the form $r(e_1, e_2)$ not currently in the KG are false if a true fact of the form $r(e_1, e')$ or $r(e', e_2)$ currently exists in the KG [51]. Therefore, one can produce *corrupted* facts for a given true fact $r(e_1, e_2)$ by sampling a random entity e' and using it instead of e_1 (resp., e_2), yielding the corrupted fact $r(e', e_2)$ (resp., $r(e_1, e')$).

Using this negative sampling, embedding models train by (i) maximising the plausibility of known facts $r(e_1, e_2)$ and (ii) minimising the plausibility of a set $C^{r(e_1,e_2)}$ of $k \in \mathbb{N}^+$ negatively sampled corrupted facts (of the form $r(e', e_2)$ or $r(e_1, e')$). Finally, negative sampling is typically restricted so as to only produce corrupted facts which currently do not exist in the KG. In other words, a resulting corrupted fact $r(e', e_2)$ (resp., $r(e_1, e')$) is only considered if it is not currently present as a true fact in the KG training set.

Loss function. Given a true fact $r(e_1, e_2)$, its corresponding k negative samples (i.e., corrupted facts) and the scores for all aforementioned facts, a *loss function* is used to compute gradients for all embeddings and map each fact score to its target value. One popular example is *negative sampling* loss [159], which uses a *margin* hyper-parameter $\gamma > 0$ to separate true and corrupted facts, such that (i) true facts are assigned positive scores lower than γ and (ii) corrupted facts are assigned positive scores higher than γ. More formally, negative sampling loss is defined as follows:

$$\begin{aligned}\mathsf{loss}_{\mathrm{NS}}(r(e_1, e_2)) = &- \log\Big(\sigma\big(\gamma - \mathsf{score}(r(e_1, e_2))\big)\Big) \\ &- \sum_{r(h,t)\in C^{r(e_1,e_2)}} \Pr(r(h, t)) \log\Big(\sigma\big(\mathsf{score}(r(h, t)) - \gamma)\big)\Big),\end{aligned}$$

where σ denotes the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, and $\Pr(r(h, t))$ is a probability for each negative sample computed using a softmax function coupled with a *temperature* hyper-parameter α. More formally:

$$\Pr(r(h, t)) = \frac{e^{-\alpha\ \mathsf{score}(r(h,t))}}{\sum_{r(h',t')\in C^{r(e_1,e_2)}} e^{-\alpha\ \mathsf{score}(r(h',t'))}}.$$

The α hyper-parameter allows models to perform *self-adversarial* negative sampling, and assign more weight to negative samples with high plausibility in the current model. Note that, when $\alpha = 0$, negative sampling uses *uniform* weights across negative samples, and thus the term $\Pr(r(h, t))$ can simply be replaced with $\frac{1}{k}$.

Another popular loss function is *cross-entropy*, which maps the scores of $r(e_1, e_2)$ and all its negative sampled facts to a probability distribution using the softmax

function and then trains this distribution to assign probability 1 for $r(e_1, e_2)$ and 0 everywhere else. As before, we assume positive scores returned by score, and consider that lower values indicate higher plausibility. Cross-entropy loss is defined as:

$$\mathsf{loss}_{\mathrm{CE}}(r(e_1, e_2)) = -\log\big(\mathrm{softmax}(r(e_1, e_2))\big),$$

where:

$$\mathrm{softmax}(r(e_1, e_2)) = \frac{e^{-\mathsf{score}(r(e_1,e_2))}}{e^{-\mathsf{score}(r(h,t))} + \sum_{r(h',t') \in C^{r(e_1,e_2)}} e^{-\mathsf{score}(r(h',t'))}}.$$

All in all, shallow node embedding models operate using the same fundamental training pipeline, based on assigning representations to KG entities and relations, and optimising the scores of facts with respect to a loss function. Hence, the main differences between different shallow node embedding models lie in how these models represent entities and relations, and in their underlying scoring function. In fact, these choices play a key role in determining the *properties* of the underlying model.

Finally, we note that node embedding models learn explicit representations for entities and relations, and thus must train on every entity (resp., relation) so as to learn their representations. Therefore, node embedding models are restricted to the *transductive* evaluation setting. In the transductive setting, all nodes (labelled and unlabelled) in the input graph are seen during training, and evaluation only occurs on an unlabelled, already observed sub-graph within the training graph.

Empirical evaluation. To evaluate KGC (resp., KBC) models empirically, a set of *true facts*, typically from the test set of a benchmark knowledge graph (resp., knowledge base), are compared against the set of all possible corrupted facts, obtained analogously to negative sampling by corrupting a random position with a uniformly randomly selected entity. Hence, a binary fact (resp., an n-ary fact) is compared against up to $2(|\mathbf{E}| - 1)$ (resp., $n(|\mathbf{E}| - 1)$) corrupted facts to estimate prediction quality, e.g., through ranking.

Similarly to negative sampling, corrupted facts computed from an n-ary true fact are filtered by considering, for each arity position i, all possible distinct entities $e' \neq e_i$, such that the resulting fact $r(e_1, \ldots, e_{i'}, \ldots, e_n)$ does not exist in the KB. However, filtering for evaluation includes not only the training facts, but also validation and test facts. Alternatively, *raw* evaluation only samples a distinct $e_{i'}$, irrespective of the status of the resulting fact in the KB.

Following the computation of all corrupted facts, both true and corrupted facts for a given arity position $i \in \{1, \ldots, n\}$ are scored using the trained embedding model

and sorted in decreasing order of plausibility. From this sorting, the *ranking* of a true fact F relative to its generated corrupted facts at arity position i, denoted $\text{rank}_i(F)$, is extracted. Hence, an n-ary fact $r(e_1, \ldots, e_n)$ yields n separate ranks (1 per arity position) $\text{rank}_i(F), i \in \{1, \ldots, n\}$, and these ranks are used to assess the overall performance of the model across the test set using the following metrics, averaged across all arity positions:

- **Mean Rank (MR).** The average rank of a true fact against its corrupted counterparts, averaged across all arity positions:

$$\text{MR}(F) = \frac{1}{n} \sum_{i=1}^{n} \text{rank}_i(F)$$

- **Mean Reciprocal Rank (MRR).** The average inverse rank, i.e., 1/rank, of a true fact.

$$\text{MRR}(F) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{\text{rank}_i(F)}$$

- **Hits@x.** The proportion of true fact ranks not exceeding x.

$$\text{Hits@}x(F) = \frac{1}{n} \sum_{i=1}^{n} [\![\text{rank}_i(F) \leq x]\!],$$

  where $[\![]\!]$ denotes the 0/1 indicator function for the inequality comparing evaluation ranks to the target x.

2.3.2 Model Properties

Models for knowledge graph completion (resp., knowledge base completion) are conceptually studied based on their *expressive power*, as well as their ability to capture *inference patterns*. We discuss both properties next.

Model expressiveness. Fundamentally, the *expressiveness* of a KGC (resp., KBC) model indicates its ability to perfectly fit different KG (resp., KB) fact configurations. More formally, given the set $\mathcal{S}$ of all possible facts over a finite relational vocabulary, a shallow node embedding model $\mathcal{M}$ is said to be *fully expressive* if for any two disjoint sets $\mathcal{T} \subseteq \mathcal{S}$ of true facts and $\mathcal{F} \subseteq \mathcal{S}$ of false facts, there exists a (finite-dimensional) model configuration for $\mathcal{M}$ that maps all facts in $\mathcal{T}$ to True and all facts in $\mathcal{F}$ to False.

Intuitively, the expressiveness of a model describes its capacity to represent known facts. In particular, a fully expressive model can fit any arbitrary knowledge graph

(resp., knowledge base) configuration, whereas less expressive models can underfit the training set. In the latter case, this underfitting can yield poor generalisation to unknown facts, as the model can fail to learn key information from the training data.

Expressiveness is an important requirement for developing powerful embedding models. However, expressiveness alone is not sufficient for strong performance. Indeed, even full expressiveness only guarantees that all known facts are correctly fit, but gives no such guarantees for *unknown* facts. Hence, full expressiveness offers no indication as to the *inductive capacity* of model, namely its ability to extrapolate from known facts to make accurate predictions beyond its training set. Therefore, the theoretical study of embedding models must also consider their inductive capacity, primarily by studying their ability to jointly represent the training data and capture common inference patterns.

Inference patterns. Inference patterns are commonly used to formally analyse the generalisation ability of embedding models. Briefly, an *inference pattern* is a specification of a logical property that may exist in a KG (resp., KB), which, if learned, enables further principled inferences from existing facts. For instance, a popular inference pattern is *symmetry*, which applies for binary relations. More specifically, if a binary relation r is symmetric, then whenever $r(e_1, e_2)$ holds, $r(e_2, e_1)$ necessarily also holds. Therefore, any embedding model capturing the symmetry of r can automatically predict the symmetric closure of r, and thus has a strong inductive capacity. Another common inference pattern is *relational hierarchy*, where every fact holding with a sub-relation r automatically holds for a super-relation s.

Capturing inference patterns is *complementary* to model expressiveness, as it provides models with potential representations that improve their prediction quality beyond the training set and serves as an indication of model inductive capacity. However, the expressiveness of a model can directly impact its ability to capture inference patterns. For example, a model assigning identical scores to any pair of facts $r(e_1, e_2)$ and $r(e_2, e_1)$ by construction is clearly not fully expressive, as it fails to represent anti-symmetric r configurations. This expressiveness limitation directly affects inference patterns, as this same model clearly cannot capture the anti-symmetry inference pattern for relation r, which imposes mutual exclusion between $r(e_1, e_2)$ and $r(e_2, e_1)$. Nonetheless, the converse does not hold. More specifically, failing to capture an inference pattern does not necessarily imply a limitation on model expressiveness. For instance, a model that cannot capture the symmetry inference pattern can still be fully expressive and accurately represent all symmetric pairs for a symmetric relation

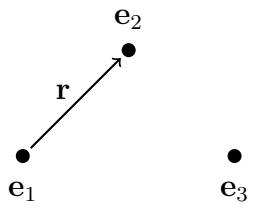


Figure 2.1: A sample 2-dimensional TransE model over a KG with $\mathbf{E} = \{e_1, e_2, e_3\}$ and $\mathbf{R} = \{r\}$. In this configuration, the binary relation r, encoded via a vector $\mathbf{r}$, exactly translates between the embeddings of e_1 and e_2. However, the translation $\mathbf{r}$ does not hold between any other pair of entities, and thus $r(e_1, e_2)$ is the only true fact encoded in this configuration.

r over the finitely sized $\mathbf{E} \times \mathbf{E}$. Hence, inference patterns, as well as model expressiveness, offer different perspectives through which to study the power and viability of embedding models.

2.3.3 Prominent Models

Currently, shallow embedding models can broadly be classified into three main categories: translational models, bilinear models, and neural models. We present a brief overview of the main models in each category next.

Translational models. Fundamentally, translational models apply exclusively on knowledge graphs and model binary relations as translations (or other geometric operations) in a latent embedding space, and represent entities as points in this latent space. The seminal translational model is TransE [21]. In TransE, every relation r is modelled by a d-dimensional translation vector $\boldsymbol{r} \in \mathbb{R}^d$, and every entity e_i is represented by a point vector $\boldsymbol{e_i} \in \mathbb{R}^d$. Given a binary fact $r(e_1, e_2)$, TransE computes a fact score as:

$$\mathsf{score}(r(e_1, e_2)) = \|\boldsymbol{e_1} + \boldsymbol{r} - \boldsymbol{e_2}\| .$$

Therefore, a correct fact in TransE has a score of 0, and corresponds to the head entity point translating exactly to the tail entity point via the relevant relation translation. To illustrate, an example TransE configuration is shown in Figure 2.1.

TransE has been widely studied and applied for KGC, but suffers from many limitations. First, it has limited expressiveness. In particular, it cannot represent *one-to-many*, *many-to-one*, and *many-to-many* relations and facts. For instance, perfectly representing the facts $r(e_1, e_2)$ and $r(e_1, e_3)$ in TransE implies that $\boldsymbol{e_1} + \boldsymbol{r} = \boldsymbol{e_2}$ and that $\boldsymbol{e_1} + \boldsymbol{r} = \boldsymbol{e_3}$, and thus that $\boldsymbol{e_2} = \boldsymbol{e_3}$. This then falsely implies that any

facts holding for e_2 necessarily hold for e_3. Moreover, TransE cannot represent multiple relations holding between two entities: To represent both $r(e_1, e_2)$ and $s(e_1, e_2)$, TransE must set $\boldsymbol{r} = \boldsymbol{s}$, which implies that the relations are semantically identical. Furthermore, TransE cannot represent symmetric relations, as a symmetric fact pair $r(e_1, e_2)$ and $r(e_2, e_1)$ can only simultaneously hold if $\boldsymbol{r} = 0$. Finally, TransE, by design, can only handle binary facts, and thus cannot naturally apply to general knowledge bases[1].

To alleviate these limitations, several extensions to TransE have been proposed. For example, TransH [176] introduces relation-specific hyperplanes, such that, for a fact $r(e_1, e_2)$, a relation r projects entity points for e_1 and e_2 into this hyperplane prior to applying the TransE scoring function. Furthermore, TransR [104] introduces relation-specific spaces to which entity points are mapped using a bilinear product, and TranSparse [82] sets sparsity requirements on this bilinear mapping. By mapping entities into relation-specific representations, all aforementioned models can then model multiple relations holding among the same entities, but still suffer from the remaining TransE limitations. Finally, TransF [58] relaxes the TransE scoring function so that only the *direction* between head and tail aligns with the relation translation, as opposed to translation holding exactly. This allows TransF to model, e.g., one-to-many relations, by setting tail entities at varying distances along the same direction from the origin as the head entity. However, TransF still cannot represent symmetric relations.

Beyond translations, RotatE [159] uses *rotations* to model relations in a *complex* embedding space. More specifically, RotatE represents an entity $e_i \in \mathbf{E}$ using a complex point vector $\boldsymbol{e_i} \in \mathbb{C}^d$, and represents relations as a complex rotation vector $\boldsymbol{r} \in \mathbb{C}^d$, which applies element-wise to entity representations. These rotations are norm-preserving, and thus for every $j \in \{1, \ldots, d\}$, $|\boldsymbol{r}(j)| = 1$, where (j) denotes the j^{th} element of $\boldsymbol{r}$. Formally, the RotatE scoring function is defined as follows:

$$\mathsf{score}(r(e_1, e_2)) = \|\boldsymbol{e_1} \circ \boldsymbol{r} - \boldsymbol{e_2}\|,$$

where $\circ$ denotes the element-wise rotation operation.

By representing relations as rotations, RotatE can represent relational symmetry by setting all rotation phases at all dimensions to multiples of π, i.e., $\forall j \in \{1, \ldots, d\}$,

[1] In practice, TransE is not trained to represent correct facts with a score of 0, but rather with a *margin* γ allowing some error. This margin offers the model some flexibility and enables it to alleviate the aforementioned limitations empirically, e.g., by making entity representations in one-to-many relations close in the latent space, rather than exactly identical. However, the earlier limitations remain valid and will still arise given sufficiently dense data, irrespective of margin.

$\boldsymbol{r}(j) = \pm 1$. However, RotatE otherwise remains as limited as TransE, in that it also cannot represent one-to-many, many-to-many and many-to-one relations.

At a high level, translational models are interpretable and can capture various inference patterns. Indeed, relational symmetry can be read off in RotatE by checking the rotation angles of every relation representation, and inverse relations r, s can be inferred in TransE (resp., RotatE) by verifying $\boldsymbol{r} = \boldsymbol{s}$ (resp., $\boldsymbol{r} = \bar{\boldsymbol{s}}$, i.e., $\boldsymbol{s}$ is the complex conjugate of $\boldsymbol{r}$). However, translational models have limited expressiveness. In fact, no known translational model is fully expressive.

Bilinear models. Bilinear models are based on *tensor factorisation* methods, and learn representations for entities and relations from which a *truth tensor* for a given KG (resp., KB) can be reconstructed. More specifically, every KG (resp., n-ary KB) can be represented by a 0/1 truth tensor $\mathbf{T}$ of order 3 (resp, $n+1$). Concretely, given a KG and the fact $r_i(e_j, e_k), i \in \{1, \ldots, |\mathbf{R}|\}, j, k \in \{1, \ldots, |\mathbf{E}|\}$, the following equivalence holds:

$$r_i(e_j, e_k) \Leftrightarrow \mathbf{T}_{ijk} = 1$$

Bilinear models learn entity and relation embeddings which can be combined using a bilinear product to reconstruct the overall KG truth tensor. For example, RESCAL [131] represents a relation r as a full-rank $d \times d$ matrix $\mathbf{M}_r$, and entities as d-dimensional vectors $\boldsymbol{e}$, and computes the score for a fact $r(e_1, e_2)$ as:

$$\mathsf{score}(r(e_1, e_2)) = \boldsymbol{e_1}^{\mathrm{T}} \mathbf{M}_r \boldsymbol{e_2},$$

such that higher scores indicate a higher likelihood of fact correctness, and where $^{\mathrm{T}}$ indicates the matrix transpose operation. RESCAL is fully expressive, but introduces d^2 parameters per relation, which makes the model susceptible to overfitting. Therefore, DistMult [185] is proposed as a simplification to RESCAL, where the matrix $\mathbf{M}_r$ is made diagonal. This reduces the number of parameters per relation to d, but creates a limitation in expressiveness. Indeed, given diagonal $\mathbf{M}_r$, the scoring function of DistMult is inherently symmetric, and therefore the scores of facts $r(e_1, e_2)$ and $r(e_2, e_1)$ are identical by construction. Hence, DistMult cannot represent non-symmetric relations, and is not fully expressive.

To overcome the limitations of DistMult while preserving compact model parametrisation, ComplEx [166] applies the DistMult setup in a complex space. More concretely, an entity $e_i \in \mathbf{E}$ is represented by a vector in $\mathbb{C}^d$, and the diagonal matrix

$\mathbf{M}_r$ is defined using complex numbers. The ComplEx scoring function for $r(e_1, e_2)$ is then defined in the complex space as:

$$\mathsf{score}(r(e_1, e_2)) = \mathrm{Re}(\boldsymbol{e_1}^{\mathrm{T}} M_r \bar{\boldsymbol{e_2}}),$$

where Re denotes the real component of a complex number and $\bar{\cdot}$ denotes the complex conjugate, i.e., $\overline{a + bi} = a - bi$. Note that ComplEx can represent anti-symmetry by using imaginary values. In fact, ComplEx is fully expressive.

SimplE [86] is based on canonical polyadic (CP) [75] tensor factorisation. It represents every entity e_i with two vectors $\boldsymbol{h_i}$ and $\boldsymbol{t_i}$, which are used based on the position (head and tail, respectively) e_i occupies in a binary fact, i.e., $\boldsymbol{h_i}$ is used when e_i is the head entity, and $\boldsymbol{t_i}$ is used when it is the tail entity. Moreover, SimplE represents relations using two vectors $\boldsymbol{r}$ and $\boldsymbol{r^{-1}}$, where the latter represents the inverse relation of r, i.e., when $r(e_1, e_2)$ holds, $r^{-1}(e_2, e_1)$ also holds. Then, the SimplE scoring function for $r(e_1, e_2)$ is defined as:

$$\frac{1}{2}(\boldsymbol{h_1}^{\mathrm{T}} \boldsymbol{r} \boldsymbol{t_2} + \boldsymbol{h_2}^{\mathrm{T}} \boldsymbol{r} \boldsymbol{t_1}).$$

Note that SimplE implicitly applies two copies of DistMult on the different head and tail embeddings of entities, and does so with the help of inverse relations. This added parametrisation allows SimplE to represent anti-symmetry and to be fully expressive.

Finally, TuckER [10] is based on Tucker decomposition [167], and introduces a shared learnable tensor through which scores are computed. More concretely, given a fixed dimensionality d for entity and relation embeddings, TuckER introduces a shared tensor $\mathcal{W} \in \mathbb{R}^{d \times d \times d}$, and computes the score of $r(e_1, e_2)$ as:

$$\mathcal{W} \times_1 \boldsymbol{e_1} \times_2 \boldsymbol{r} \times_3 \boldsymbol{e_2},$$

where $\times_i$ indicates tensor multiplication along the i^{th} order. Observe that all fact scores are computed using $\mathcal{W}$, and thus $\mathcal{W}$ enables significant parameter sharing, and in fact allows the model to be fully expressive. With special configurations for $\mathcal{W}$, as well as entity and relation embeddings, TuckER is shown to subsume RESCAL, its adaptations, and SimplE [10].

In general, bilinear models (with the exception of DistMult) are fully expressive. This can intuitively be understood, as, given sufficient dimensionality, a factorisation of the KB truth tensor can eventually fully reconstruct this tensor. However, bilinear models are less interpretable than their translational counterparts, given the rich interactions between embeddings and the products involved in score computation.

Neural models. Neural models represent entities and relations in a KB with latent embeddings, but use a *neural component* to process embeddings and/or return a score for a given fact.

A seminal example is neural tensor networks (NTN) [155], where entities are represented as vectors and relations are represented using (i) a third-order tensor $\mathbf{M}_r \in \mathbb{R}^{d \times d \times k}$, and (ii) two matrices $\mathbf{U}_r, \mathbf{V}_r \in \mathbb{R}^{d \times k}$ to compute relation-specific scores. More concretely, scores for a fact $r(e_1, e_2)$ within NTN are given by:

$$\mathsf{score}(r(e_1, e_2)) = \mathbf{w}^{\mathrm{T}} \tanh(\mathbf{e_1}^{\mathrm{T}} \mathbf{M}_r \mathbf{e_2} + \mathbf{U}_r \mathbf{e}_1 + \mathbf{V}_r \mathbf{e}_2 + \mathbf{b}_r),$$

where $\mathbf{w} \in \mathbb{R}^k$ is a shared transformation vector, $\mathbf{b}_r \in \mathbb{R}^k$ is a relation-specific bias vector, and tanh denotes the hyperbolic tangent activation function. Therefore, NTN closely resembles a single-layer perceptron, but with an additional bilinear component similar to that of RESCAL.

NTN has been succeeded by several models leveraging more sophisticated neural network architectures. For instance, ConvE [48] applies a convolutional neural network (CNN) on entity and relation embeddings to compute fact scores, and HittER [39] uses a transformer-based model.

Neural models offer very strong representation power owing to the universality of neural networks [43, 77]. However, this power comes at the cost of interpretability. Indeed, neural networks are largely considered as black-box components, which cannot be analysed due to their non-linearities. Therefore, it is highly challenging to theoretically study the behaviour of neural embedding models from an expressiveness or inductive capacity perspective.

2.4 Graph Neural Networks

Graph neural networks (GNNs) [67, 149] have become the de facto standard approach for learning functions on graph-structured data, owing to their strong inductive bias on graph structures. In particular, GNNs capture the structure of the graph in a principled manner. Given an input graph G, GNNs act on initial vector representations for every node in G (e.g., features, embeddings), and update these representations iteratively using *invariant* or *equivariant* GNN layers. GNNs have enabled breakthrough results on several graph-based tasks, such as graph classification [27] and node classification [78].

2.4.1 Message Passing Neural Networks

Among GNNs, a popular approach in the literature is message passing neural networks (MPNNs) [62], which perform *message passing* between graph nodes to compute representation updates. Fundamentally, an MPNN operates on an undirected graph G, and consists of T message passing iterations such that, at each iteration, nodes exchange messages with their neighbours and subsequently update their vector representations. Hence, a node $u \in G$ can potentially have $T+1$ different representations across iterations, and we denote these by $\mathbf{h}_u^{(t)} \in \mathbb{R}^d, t \in \{0, \ldots, T\}$, where d is the embedding dimensionality. More formally, given node $u \in G$, iteration $t \in \{0, \ldots, T-1\}$, and the *neighbourhood* of u, given by $N(u) = \{v \in V |\ (u, v) \in E\}$, the standard MPNN update equation can be written as:

$$\mathbf{h}_u^{(t+1)} = \mathsf{COM}\Big(\mathbf{h}_u^{(t)}, \mathsf{AGG}\big(\{\mathbf{h}_v^{(t)} |\ v \in N(u)\}\big)\Big), \tag{2.1}$$

where COM and AGG are functions, and AGG is permutation-invariant.

In MPNNs, the node representations $\mathbf{h}_u^{(T)}, u \in V$, are used to compute an output for the task at hand. For node-level tasks, this is done through a node-level differentiable function $f(\mathbf{h}_u^{(T)})$, e.g., a multi-layer perceptron (MLP) $f : \mathbb{R}^d \to \{0, 1\}$ for binary node classification. For edge-level functions, the analogous function is of the form $f(\mathbf{h}_u^{(T)}, \mathbf{h}_v^{(T)}), u, v \in V$.

Pooling. For graph-level tasks, node representations $\mathbf{h}_u^{(T)}, u \in V$, must be combined into a single representation to predict graph properties. This operation is referred to as "pooling" [129], and can be done using standard functions such as mean, maximum, or sum, following which a mapping to the output space, e.g., via a function f, can be made. However, pooling can also be done using more sophisticated methods such as linear maps and, more generally, neural networks. Moreover, pooling can also involve node representations from intermediate layers in certain cases [56].

Properties. MPNNs, by design, offer strong inductive biases for graph-structured data. Indeed, the permutation invariance of the AGG function makes MPNNs invariant to the order of nodes in neighbourhoods. Furthermore, MPNNs explicitly leverage edges and the overall graph structure when computing node representations, and can naturally apply to nodes/edges not previously seen during training, i.e., they can be *inductive*. Moreover, MPNNs perform computations at the *local node* level, and learn increasingly global information with repeated message passing iterations, which is helpful for tasks such as node classification. Finally, MPNN can autonomously learn relationships between nodes and identify key features, and thus offer a strong alternative to classic feature engineering [85].

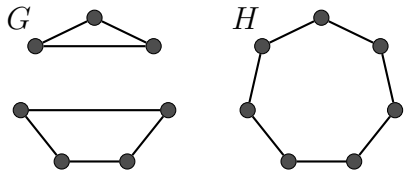


Figure 2.2: G and H are indistinguishable by 1-WL.

Limitations. Despite their strengths, MPNNs also present multiple limitations impeding their applicability. First, it has been shown that standard MPNNs can be at most as powerful as the 1-dimensional Weisfeiler-Leman algorithm (1-WL) [183, 127].

Fundamentally, the 1-WL algorithm is a heuristic for graph isomorphism checking, in which node representations are iteratively refined by (i) aggregating messages from neighbour nodes at every node and (ii) computing a combination of the current node representation and those of its neighbours using an injective hash function. These refinement steps are repeated until the number of distinct node representations converges, which requires at most $O(n)$ iterations for a graph with n nodes.

Notice that 1-WL refinement steps are highly similar to MPNN updates, with the main difference lying in the explicit use of a hash function, rather than learnable parameters, as well AGG and COM. Indeed, this is the main intuition underlying the expressiveness bound proof: A 1-WL refinement iteration can be emulated by an MPNN with maximally injective choices for AGG and COM, so as to best emulate the hash function, which in turn yields the 1-WL expressiveness upper bound.

As the expressive power of MPNNs is upper-bounded by 1-WL, graphs (or alternatively, nodes) cannot be distinguished by MPNNs if 1-WL does not distinguish them. For example, 1-WL cannot distinguish between the graphs G and H, shown in Figure 2.2, despite them being clearly non-isomorphic. Therefore, MPNNs cannot learn functions with different outputs for G and H.

Another somewhat trivial limitation in the expressiveness of MPNNs is that information is only propagated along edges, and hence can never be shared between distinct connected components of a graph [13, 183]. However, an easy way to overcome this is to include a *global readout*. More specifically, a global readout computes a graph-level aggregate representation, similarly to pooling, which is used *during* message passing to help compute representation updates. With global readout, the MPNN update equation becomes:

$$\mathbf{h}_u^{(t+1)} = \mathsf{COM}\Big(\mathbf{h}_u^{(t)}, \mathsf{AGG}\big(\{\mathbf{h}_v^{(t)} |\ v \in N(u)\}, \mathsf{READOUT}(\{\mathbf{h}_v^{(t)} |\ v \in V\})\big)\Big), \qquad (2.2)$$

where $\mathsf{READOUT}$ is a permutation-invariant function.

Beyond expressiveness, it has been shown that MPNNs suffer from an *oversmoothing* [100] problem with more message passing iterations, as node representations converge to identical values when T increases. This phenomenon is in many ways inherent to MPNNs, as their update equation shares a strong conceptual connection with graph filters [73], and the composition of these filters ultimately leads to a very selective low-pass filtering operation which causes severe information loss. This implies that using deeper MPNNs to compute more global graph representations eventually produces undesirable similarities across node representations and ultimately loses key information. Hence, MPNNs cannot currently be built with large depths, i.e., message passing iterations, as has been the case within deep learning models more generally.

MPNNs also struggle with transferring long-range information across the graph, a problem known as *oversquashing* [8]. More concretely, MPNNs, by design, produce an exponentially large receptive field for every node relative to the number of message passing layers T. However, information from this field must fit into a constant-sized embedding representation. Hence, this setting leads to a natural *bottleneck* impeding long-range information flow in MPNNs. Note that oversquashing and oversmoothing, though both brought on by increased message passing, occur due to different underlying phenomena: Oversmoothing stems from the filtering nature of MPNN updates, whereas oversquashing arises from communication bottlenecks and exponential information compression due to direct neighbourhood message passing.

2.4.2 Prominent Models

We now introduce several popular MPNN models, both for standard single-relational graphs and for multi-relational graphs. We start with the Graph Convolutional Network (GCN) [92], which extends the convolution operation, which has found success in convolutional neural networks [96], to graphs. Concretely, GCN performs message passing based on node representations, e.g., initial features, and the normalised adjacency matrix of the input graph, and jointly considers the updating node along with its neighbours by means of a self-loop. More formally, the GCN message passing update equation is given by:

$$\mathbf{h}_u^{(t+1)} = \sigma\Big(\mathbf{W}^{(t+1)} \sum_{v \in N(u) \cup \{u\}} \frac{\mathbf{h}_v^{(t)}}{\sqrt{|N(u)| + |N(v)|}}\Big),$$

where $\mathbf{W}^{(t+1)} \in \mathbb{R}^{d \times d}, t \in \{0, \ldots, T-1\}$ denotes iteration-specific learnable linear transformations, and σ is a non-linearity, e.g., sigmoid, hyperbolic tangent (tanh), rectified linear unit (ReLU).

Intuitively, GCNs perform the following operations at every iteration:

1. They multiply all node representations at time t by $\mathbf{W}^{(t+1)}$ to yield transformed representations.

2. They aggregate node representations from all node neighbourhoods based on normalised, neighbourhood-based, weights. Aggregation and combination are unified in GCNs, as the updating node is included via a self-loop.

GCNs explicitly account for the graph structure, namely neighbourhood sizes and adjacency, to convolve over the graph, and therefore introduce a strong convolutional inductive bias. Furthermore, GCN iterations can be efficiently performed using matrix multiplications, which makes this model very computationally efficient in practice. However, GCNs, as initially proposed by Kipf and Welling, are transductive, as they require the complete graph adjacency matrix to compute normalisation weights for message passing. Indeed, adding new nodes changes the adjacency matrix, leading to different and incompatible message passing protocols during evaluation relative to training time [73]. Nonetheless, later versions of GCNs have been made inductive by dropping node degree normalisation from the update equation.

In addition to the convolutional perspective, message passing can also be interpreted through the perspective of *sequence modeling* over graph nodes. Indeed, one can consider the evolution of node representations across message passing iterations as a sequence, and thus apply a sequence model to predict the next representation for a given node based on its prior representations. This is the main motivation behind Gated Graph Neural Networks (GGNNs) [102]. GGNNs use a gated recurrent unit (GRU) to update node representations (i.e., as the COM function), and perform aggregation by summing transformed neighbour representations. More formally:

$$\mathbf{h}_u^{(t+1)} = \mathsf{GRU}(\mathbf{h_u}^{(t)}, \sum_{v \in N(u)} \mathbf{W}^{(t+1)} \mathbf{h}_v^{(t)}).$$

In addition to GCN and GGNN, more sophisticated aggregation functions have also been explored. For example, Graph Isomorphism Networks (GIN) [183] use a multilayer perceptron (MLP) as the COM function, and sum aggregation to perform maximally injective node updates. In other words, GIN is designed to compute a maximum number of distinct representations given distinct node neighbourhoods. The GIN update operation is given by:

$$\mathbf{h}_u^{(t+1)} = \mathsf{MLP}((1+\epsilon)\mathbf{h}_u^{(t)}, \sum_{v \in N(u)} \mathbf{h}_v^{(t)}),$$

where $\epsilon \geq 0$ is a tunable hyper-parameter typically set to 0.

Graph Attention Networks (GAT) use an attention mechanism for aggregation, such that the summation of neighbour representations is weighted by attention scores. At the node level, GAT computes a pairwise attention score between edge-connected nodes u_1, u_2 (as well as self-loops u, u), denoted α_{u_1,u_2} (resp., $\alpha_{u,u}$ for self-loops) using an MLP and both nodes' vector representations, and subsequently normalises all scores such that $\Sigma_{u' \in G} \alpha_{u,u'} = 1$. More formally:

$$\mathbf{h}_u^{(t+1)} = \sigma(\mathbf{W}^{(t+1)} \sum_{v \in N(u) \cup u} \alpha_{u,v} \mathbf{h}_v^{(t)}).$$

In this equation, the attention weights are computed as:

$$\alpha_{u,v} = \frac{\mathbf{e}_{u,v}}{\sum_{v' \in N(u) \cup u} \mathbf{e}_{u,v'}},$$

such that $\mathbf{e}_{u,v} = \mathbf{a}^T(\mathbf{W}\mathbf{h}_u || \mathbf{W}\mathbf{h}_v)$, where $\mathbf{a} \in \mathbb{R}^{2d}$, $\mathbf{W} \in \mathbb{R}^{d \times d}$ are shared learnable weights, and $||$ denotes the concatenation operation.

MPNNs for multi-relational graphs. So far, all presented MPNNs aggregate across all node neighbourhoods without considering the relations connecting nodes, i.e., the same messages are passed along edges irrespective of their potentially different underlying relations. Therefore, these MPNNs are only applicable to single-relational graphs, and cannot incorporate relational information during message passing.

To alleviate this, a simple relational extension to GCN, named rGCN [150], has been proposed, so as to (i) partition node neighbourhoods based on the relation of the connecting edges and to (ii) compute separate aggregation functions for every relation in the relational vocabulary $\mathbf{R}$. In what follows, we denote the neighbourhood of node u with respect to relation r as $N^r(u) = \{v \in V |\ r(u,v) \in E_r\}$, where E_r denotes the set of all r relation edges in G. The rGCN update equation can be written as:

$$\mathbf{h}_u^{(t+1)} = \sigma(\mathbf{W}_{\text{self}}^{(t+1)} \mathbf{h}_u^{(t)} + \sum_{r \in \mathbf{R}} \sum_{v \in N^r(u)} \frac{1}{c_{r,u}} \mathbf{W}_r^{(t+1)} \mathbf{h}_v^{(t)}),$$

where $\mathbf{W}_{\text{self}}^{(t+1)}$ and $\mathbf{W}_{\text{r}}^{(t+1)}$ are learnable self-loop and relation-specific linear maps respectively, and $c_{r,u}$ is a normalisation constant analogous to that of GCN. Note that rGCN naturally handles multi-relational graphs, and thus can apply on, e.g., knowledge graphs. In fact, rGCN has been applied towards KGC, by using the rGCN model to compute entity representations and subsequently applying the DistMult scoring function on these representations to predict missing edges.

Since the proposal of rGCN, several more sophisticated multi-relational MPNNs have been introduced. For instance, KBGAT [130] builds on rGCN and uses a variant of graph attention networks (GAT), in which entities can attend differently to their neighbours during message passing. Furthermore, Graph Transformer Networks (GTNs) use transformer-based operations [189] to perform message passing on multi-relational graphs. However, KBGAT, GTN, and current multi-relational MPNNs in general are all largely based on the same underlying treatment of relations, i.e., separate aggregation per relation. Therefore, we only give a high-level overview of these models, and refer interested readers to the original papers for more technical details.

2.4.3 Higher-order Graph Neural Networks

Recently, substantial research has been dedicated to proposing higher-order GNN models to improve on the limited expressiveness of MPNNs. In particular, a main goal in developing higher-order MPNNs is to emulate the generalisation of 1-WL, namely the k-WL graph isomorphism heuristic, which operates over k-tuples of nodes.

In what follows, we use folklore k-WL [29, 68], $k \in \mathbb{N}^+$, which is equivalent to the standard $k+1$-WL used in the literature [127, 114], as this is the standard in graph isomorphism literature. When $k \geq 2$, k-WL is strictly more powerful than 1-WL. Moreover, for any consecutive values k, $k+1$ ($k \geq 1$), $(k+1)$-WL is strictly more powerful than k-WL [29], which implies a continual improvement in expressive power with larger values of k that creates a hierarchy among the different k-WL algorithms.

A key higher-order model directly emulating the k-WL hierarchy is k-GNN, which operates on k-tuples of nodes. More specifically, k-GNN learns embeddings for k-tuples of nodes, and performs message passing between them, as opposed to individual nodes [127] in standard MPNNs. The k-GNN model has $(k-1)$-WL expressive power, but requires $O(|V|^k)$ memory to run, leading to *excessive memory requirements*.

Invariant (resp., equivariant) graph networks (IGNs) [115, 116, 88] implicitly emulate $(k-1)$-WL using k^{th} order tensors. At a high level, k-IGNs can be seen as a composition of T equivariant layers $L_1, \ldots, L_T$ (with intermediate non-linearities) followed by an invariant layer H and a multi-layer perceptron. The intermediate equivariant layers each return k^{th} order tensors, and the overall model is invariant by design, as it is a composition of equivariant layers followed by the invariant layer H.

Unlike standard MPNNs, invariant networks [116] have been shown to be *universal*. However, this result is not relevant in practice, as it requires intermediate

tensor orders of $O(|V|^2)$, where $|V|$ denotes the number of graph nodes. Furthermore, higher-order models like k-IGNs require *intractably-sized intermediate tensors* in practice, preventing their widespread use.

To alleviate the heavy computation requirements of k-IGNs, a special class of model, provably powerful graph networks (PPGNs)[114], has been proposed, so as to achieve 2-WL expressive only using 2^{nd} order tensors and matrix multiplication. More specifically, PPGN is based on "blocks" of (equivariant) multi-layer perceptrons (MLPs), whose outputs are now combined through matrix multiplication. PPGN theoretically offers 2-WL expressive power, and only requires memory $O(|V|^2)$ (compared to $O(|V|^3)$ for 3-GNNs and 3-IGNs). All in all, higher-order GNNs are more powerful than MPNNs in theory, but suffer from many practical limitations that limit their widespread practical use.

Chapter 3

BoxE: A Box Embedding Model for Knowledge Base Completion

3.1 Introduction

Knowledge bases (KBs) are fundamental means for representing, storing, and processing structured information, and are commonly used to enhance the *reasoning* and *learning* capabilities of modern information systems. KBs can be seen as a collection of facts of the form $r(e_1, \ldots, e_n)$, which represent a relation r between the entities $e_1, \ldots, e_n$, and knowledge graphs (KGs) are a special case, where all relations are binary, i.e., connecting only two entities. KBs such as YAGO [113], NELL [124], Knowledge Vault [51], and Freebase [19] contain hundreds of millions of facts, and are increasingly important both in academia and industry. Concretely, they are key components in multiple applications, such as question answering [20], recommender systems [173], information retrieval [182], and natural language processing [184].

Despite their ubiquity and large size, modern knowledge bases are highly *incomplete*, which makes their downstream use more challenging. For instance, 71% of individuals in Freebase lack a connection to a place of birth [180], and this missing information could lead to, e.g., missing potential query answers based on birthplace, or reaching wrong decisions due to noisy/incomplete data. *Knowledge base completion (KBC)*, aiming at automatically inferring missing facts in a knowledge base based on existing facts, has thus become a focal point of research in recent years. One prominent approach for KBC is to learn *embeddings* for entities and relations in a latent space, such that these embeddings, once learned from known facts, can be used to *score* the plausibility of unknown facts.

Currently, the main embedding approaches for KBC are translational models [21, 159], which score facts based on distances in the embedding space, bilinear models

[166, 185, 10], which learn embeddings that factorise the truth tensor of a knowledge base, and neural models [48, 155, 130], which score facts using dedicated neural architectures. Each of these models suffer from limitations, most of which are well-known. Translational models, for instance, are theoretically inexpressive, i.e., they cannot provably fit an arbitrary knowledge graph. Furthermore, embedding models fail to capture simple sets of *logical rules*: Even capturing a simple hierarchy rule, e.g., $\mathsf{friends}(x, y) \Rightarrow \mathsf{knows}(x, y)$, goes beyond the capability of most existing models [72]. This also makes it difficult to inject background knowledge, i.e., schematic knowledge, in the form of logical rules, into the model to improve KBC performance. Additionally, existing KBC models are primarily designed for knowledge graphs, and thus do not naturally extend to KBs with *higher-arity* relations involving 3 or more entities, e.g., $\mathsf{degreeFrom}(\mathsf{Turing}, \mathsf{PhD}, \mathsf{Princeton})$ [57]. Higher-arity relations are prevalent in modern KBs such as Freebase [179], and cannot always be reduced to a KG without loss of information [57]. Despite the rich landscape of approaches for knowledge base completion, no existing model currently offers a solution to all these limitations.

In this chapter, we simultaneously address all these problems by encoding relations as explicit *regions* in the embedding space, where logical properties such as relation subsumption and disjointness can naturally be analysed and inferred. Specifically, we present *BoxE*, a *spatio-translational* box embedding model, which models relations as sets of d-dimensional boxes naturally corresponding to classes in the latent space, and entities as d-dimensional points. Facts are then scored based on the final positions of entity embeddings relative to relation boxes.

Fundamentally, the box abstraction for relations in BoxE allows the model to represent facts of arbitrary arity. Therefore, BoxE naturally applies to general knowledge bases, and can represent entity classes, i.e., unary relations. Moreover, BoxE relies on translations not at the relation level, like standard translational models, but at the *entity* level, such that entities translate one another when appearing jointly in a fact. This distinction in fact makes BoxE fully expressive, which, to our knowledge, is a first for a translation-based embedding model. Most importantly, the box representations for relations in BoxE endow the model with strong inductive capacity, capturing *generalised* inference patterns and rule languages from data, and even allowing the *injection* of a rich language of logical rules, which can significantly improve KBC performance.

The rest of the chapter is organised as follows. Section 3.2 discusses related work on higher-arity knowledge base completion and region-based embedding models, and Section 3.3 presents the BoxE model. We then analyse the properties of this model

in Section 3.4, and conduct an extensive empirical evaluation in Section 3.5. Finally, we conclude with a summary and discussions in Section 3.6.

3.2 Related Work

In this section, we discuss related works addressing knowledge base completion over higher-arity knowledge bases as well as region-based embedding models, as these share strong connections with BoxE.

Higher-arity KBC. The vast majority of research in the literature has targeted knowledge graph completion (KGC) and developed models for predicting binary facts (cf. Chapter 2 for a detailed discussion). However, relatively little attention has been given to the general knowledge base completion setting. Though this may initially seem like a minor technicality, this difference leads to a substantial loss of information, as some higher-arity facts cannot be reduced to sets of binary facts [57]. Hence, the KBC task involves distinct relational structures than KGC which require dedicated approaches, and naively reducing knowledge bases to KGC is not a sound solution. Furthermore, current KGC models do not naturally extend to KBs, e.g., translational models. Finally, higher-arity relations are a rich source of information and are common in large-scale KB: Over one third of entities in Freebase appear in a higher-arity fact [179]. In light of this landscape, an increasing body of work has recently been developed toward building higher-arity KBC models.

A main research direction is to extend existing models for KGC to the higher-arity setting. For instance, HSimplE [57] generalises SimplE, and computes an element-wise product between entity and relation embeddings, such that entity embeddings are shifted based on the arity position at which they appear in a fact. More specifically, HSimplE uses a unique embedding per entity, and this embedding is shifted, i.e., dimensions are shifted within the vector, differently depending on the arity position where the entity appears. The authors of HSimplE additionally propose HypE, which performs a more sophisticated operation to yield position-specific entity representations. Concretely, HypE applies a shared learnable convolution operation to entity embeddings, and subsequently concatenates convolution outputs and projects them into the latent embedding space to obtain position-specific entity embeddings. At this point, the same scoring mechanism as HSimplE then applies.

In addition to these models, m-TransH [179] generalises TransH and introduces arity-specific computations and projections, and m-DistMult and m-CP [57] generalise DistMult and CP respectively by considering the factorisations of higher-order truth

tensors. Finally, m-TuckER and Generalized Tensor Decompositions (GETD) [106] extend TuckER into the higher-arity setting, where m-TuckER is a direct generalisation of the underlying factorisation, and the latter introduces further factorisation to avoid intractable parametrisation requirements.

Generally speaking, the currently proposed higher-arity KGC models directly apply to KBs, and thus can circumvent the information loss stemming from fact binarisation. However, these models face several conceptual and practical challenges. For instance, m-TuckER and GETD cannot apply to KBs with distinct-arity relations simultaneously appearing, and generalisations to translational models, by design, inherit the representational limitations of their base models. Furthermore, naive generalisations of bilinear models, e.g., m-DistMult, m-TuckER, are not scalable with respect to the maximum relation arity in the input KB.

Region-based models. Region-based models explicitly define regions in the embedding space where an output property , e.g., membership to a class, holds, and thus offer an interpretable framework through which learned entity and relation representations can be analysed. As a result, region-based embedding models have received increasing research attention, particularly for classification tasks. For example, unbounded cones [170, 171] are applied to the entity classification task as an abstraction for possible classes, and these cones are learned jointly with entity positions in a latent embedding space. However, unbounded cones suffer from a strong limitation, in that any two unbounded cones have positive correlation, and thus any conditional probability based on a known class can only increase the likelihood of other classes, i.e., for any entity $e_i \in \mathbf{E}$, and for any two classes c_1, c_2, $\Pr(x \in c_1 | c_2) > \Pr(x \in c_1)$. Hence, cones have more recently been replaced with bounded axis-aligned hyper-rectangles (boxes) [157, 101]. Boxes eliminate the positive correlation limitation inherent with unbounded cones, and naturally capture class hierarchies and intersections.

Given the advantages of boxes for representing sets of objects, they have also been used in settings beyond classification. For example, boxes have been used to represent answer sets in the Query2Box embedding-based query answering system [76]. More specifically, Query2Box addresses conjunctive queries, and by extension queries in disjunctive normal form (DNF), by computing a query answer set, represented as a box, via multiple projection and intersection operations based on the target query structure. Note that Query2Box can be applied to KBC. However, in this setting, the model ultimately reduces to a translational model, very similar to TransE, with a box correctness region applying on tail entities.

Region-based models are widely popular for class and set representation, but cannot naturally apply to binary and higher-arity facts. Indeed, naively representing tuples of entities as points, and n-ary relations as regions in the space leads to a substantial computational overhead ($|\mathbf{E}|^n$ potential points, and loss of parameter sharing). Furthermore, modelling interactions between multiple entities in the current models in a relational context is not trivial. Therefore, existing work on region-based embeddings cannot currently be reasonably applied to the KBC/KGC setting.

3.3 The BoxE Model

In this section, we introduce *BoxE*, an embedding model for knowledge base completion. BoxE encodes relations as axis-aligned *hyper-rectangles* (or boxes) and entities as *points* in the d-dimensional Euclidian space.

3.3.1 Representation

In BoxE, every entity $e_i \in \mathbf{E}$ is represented by two vectors $\boldsymbol{e_i}, \boldsymbol{b_i} \in \mathbb{R}^d$, where $\boldsymbol{e_i}$ defines the *base position* of the entity, and $\boldsymbol{b_i}$ defines its *translational bump*, which translates all the entities co-occurring in a fact with e_i from their base positions to their final embeddings by "bumping" them. The *final embedding* of an entity e_i relative to a fact $r(e_1, \ldots, e_n)$ is hence given by:

$$\boldsymbol{e_i^{r(e_1,\ldots,e_n)}} = (\boldsymbol{e_i} - \boldsymbol{b_i}) + \sum_{1 \leq j \leq n} \boldsymbol{b_j}. \tag{3.1}$$

Essentially, entity representations in BoxE are *dynamic* and *fact-dependent* for facts with arity of 2 or higher. Indeed, given a unary fact $c(e_1)$, the final embedding of e_1 relative to this fact is simply $\boldsymbol{e_1}$, its base position vector. However, this final embedding varies for, e.g., binary facts. For example, the final embedding of e_1 relative to $r(e_1, e_2)$ is given by $\boldsymbol{e_1} + \boldsymbol{b_2}$, whereas this embedding relative to $r(e_1, e_3)$ is given by $\boldsymbol{e_1} + \boldsymbol{b_3}$. Furthermore, representations for higher-arity facts are also distinct from binary representations, even when using the same entities. To illustrate, the final embedding of e_1 relative to $s(e_1, e_2, e_3)$ is now $\boldsymbol{e_1} + \boldsymbol{b_2} + \boldsymbol{b_3}$. Therefore, translational bumps in BoxE induce potentially distinct final entity embeddings for the same entity, based on the different facts being considered and the other entities co-appearing in this fact. In particular, for an n-ary fact $r(e_1, \ldots, e_i, \ldots, e_n)$, BoxE induces $|\mathbf{E}|^{n-1}$ potential distinct final embeddings for e_i, where these distinct embeddings correspond to all possible distinct entity tuples co-appearing with e_i.

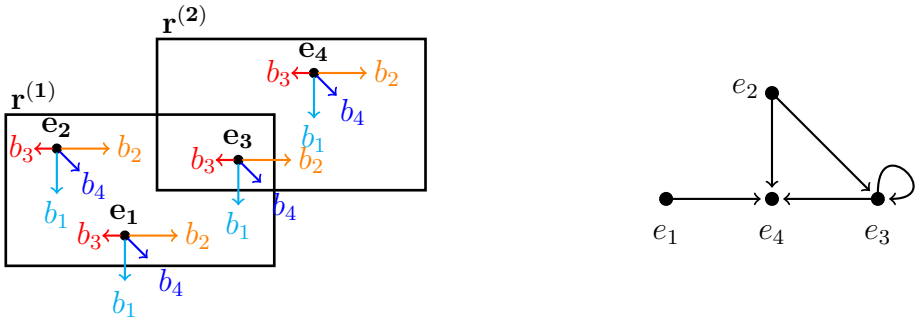


Figure 3.1: A sample BoxE model is shown on the left for $d = 2$. The binary relation r is encoded via the box embeddings $\mathbf{r^{(1)}}$ and $\mathbf{r^{(2)}}$. Every entity e_i has an embedding $\mathbf{e_i}$, and defines a bump on other entities, as shown with distinct colours. This model induces the KG on r, shown on the right.

To represent relations, BoxE uses hyper-rectangles (boxes). More concretely, BoxE represents an n-ary relation $r \in \mathbf{R}$ using n hyper-rectangles, i.e., boxes, $\boldsymbol{r^{(1)}}, \ldots, \boldsymbol{r^{(n)}} \in \mathbb{R}^{2d}$. Therefore, BoxE intuitively defines n *regions* in $\mathbb{R}^d$, each corresponding to a fixed arity position, such that a fact $r(e_1, ..., e_n)$ holds when the final embeddings of $e_1, ..., e_n$ each appear in their corresponding relation position box $\boldsymbol{r^{(1)}}, \ldots, \boldsymbol{r^{(n)}}$. This creates a *class* abstraction, where every relation box intuitively represents the set of all entities appearing at every arity position. This abstraction is best shown for the special case of unary relations (i.e., classes). There, a class is represented by a unique box, and a unary fact $c(e_i)$, i.e., a class membership fact, holds if the base position $\boldsymbol{e_i}$ (the final embedding of e_i in the unary setting) is inside $\mathbf{c}^{(1)}$.

We now illustrate the operation of BoxE with an example on a knowledge graph.

Example 3.3.1. Consider an example over a single binary relation r and entities e_1, e_2, e_3, e_4. A BoxE model configuration is given on the left in Figure 3.1, for $d = 2$, where every entity is represented as a point, and the binary relation r is represented with two boxes $\boldsymbol{r^{(1)}}$ and $\boldsymbol{r^{(2)}}$. Every entity is translated by the bump vectors of all other entities. For example, $r(e_1, e_4)$ is a true fact in the model, e.g., to be ranked high, since (i) $\boldsymbol{e_1^{r(e_1,e_4)}} = \boldsymbol{e_1} + \boldsymbol{b_4}$ is a point in $\boldsymbol{r^{(1)}}$ ($\boldsymbol{e_1}$ appears in the head box), and (ii) $\boldsymbol{e_4^{r(e_1,e_4)}} = \boldsymbol{e_4} + \boldsymbol{b_1}$ is a point in $\boldsymbol{r^{(2)}}$ ($\boldsymbol{e_4}$ appears in the tail box). Similarly, $r(e_3, e_3)$ is a true fact in the model, as $\boldsymbol{e_3^{r(e_3,e_3)}} = \boldsymbol{e_3} + \boldsymbol{b_3}$ is a point in $\boldsymbol{r^{(1)}}$ and $\boldsymbol{r^{(2)}}$, i.e., the entity is reflexive in r. The model encodes all (and only) the facts from the KG, shown on the right in Figure 3.1.

BoxE is a *spatio-translational* model: It uses static *spatial regions*, defined by relation boxes, to encode fact correctness and relational semantics, and uses *translations*,

via translational bumps, to model entity interactions in the embedding space. Note that BoxE semantics, though involving translation, differ significantly from existing translational models. Indeed, in BoxE, fact correctness is based on *absolute, static* regions in the embedding space, and entity interactions are modelled by translational bumps, yielding distinct fact-based entity representations. More specifically, entity representations in BoxE, following translational bumps, must appear in a specific, absolute region of the embedding space, defined by the relevant relation boxes. By contrast, translational models, e.g. TransE, RotatE, represent correctness in a *relative* fashion, i.e., a fact holds when the difference between entities is a fixed relation vector, and translation is performed in a *relation-based* fashion. In such models, final entity representations in the latent space don't matter in isolation, but their *relative difference* ultimately dictates fact correctness.

By using translations in this absolute way, BoxE avoids problems with representing one-to-many, many-to-one, and many-to-many facts that are seen in earlier translational models, as BoxE can simply induce distinct representations for the different entities using translational bumps. Moreover, BoxE can compactly represent interactions between multiple relations, and thus represent multiple relations holding between the same entities, simply by enforcing intersections (resp., mutual exclusion) between relation boxes. Finally, the use of absolute relation regions allows BoxE to naturally apply to higher-arity facts and classes, whereas the relative design of translational models limits them to the binary setting.

Translational bumps are very powerful, as they allow us to model complex interactions across entities in an effective manner. Observe that for the sample KG, there are 4^2 potential facts that can hold, and therefore 2^{16} possible KG configurations. Nonetheless, they can all be compactly captured by setting appropriate translational bumps to translate final entity embeddings into or out of the respective relation boxes as needed. Indeed, we later formally show that a BoxE configuration exactly fitting all facts can always be found for any KG, given sufficiently many dimensions, thus proving full expressiveness of the model.

3.3.2 Scoring Function

In the earlier example, we identified correct facts that need to be ranked higher by our scoring function, and also showed that BoxE captures all and only these facts, and thus sets all remaining facts to be false. Therefore, we need a scoring function that produces scores aligning with the desired fact correctness, and that adequately reflects the semantics and properties of BoxE. To this end, we first define a distance function

for evaluating entity positions relative to the box positions. This distance function intuitively grows slowly, relative to the centre of the target box, if a point is in this box, but grows rapidly if the point is outside of the box. The motivation behind this design is two-fold. First, lowering score ranges inside the box and increasing them outside it provides a membership advantage for entity representations which aligns with BoxE semantics. Second, this design is well-suited for model training, as points outside their target box can effectively be pushed towards it, whereas points inside are provided a minimal gradient to remain inside (or, alternatively, to move out for negative sampling facts).

Formally, let us denote by $\boldsymbol{l^{(i)}}, \boldsymbol{u^{(i)}} \in \mathbb{R}^d$ the *lower* and *upper* boundaries of a relation box $\boldsymbol{r^{(i)}}$, $i \in \{1, \ldots, n\}$, where n is the arity of relation r. Furthermore, let $\boldsymbol{c}^{(i)} = 0.5(\boldsymbol{l^{(i)}} + \boldsymbol{u^{(i)}})$ denote the centre of $\boldsymbol{r^{(i)}}$, and let $\boldsymbol{w}^{(i)} = \boldsymbol{u^{(i)}} - \boldsymbol{l^{(i)}} + 1$ be the width of $\boldsymbol{r^{(i)}}$ in all dimensions incremented by 1. For convenience, we now consider an n-ary fact $r(e_1, \ldots, e_i, \ldots, e_n)$ to align indices. Namely, entity e_i corresponds exactly to arity position i, and thus its final embedding should appear inside $\boldsymbol{r^{(i)}}$. We now define the membership of the final embedding of e_i to box $\boldsymbol{r^{(i)}}$ as:

$$\boldsymbol{e_i^{r(e_1,\ldots,e_i,\ldots,e_n)}} \in \boldsymbol{r^{(i)}} \Leftrightarrow \boldsymbol{l^{(i)}} \leq \boldsymbol{e_i^{r(e_1,\ldots,e_i,\ldots,e_n)}} \leq \boldsymbol{u^{(i)}}.$$

Furthermore, we denote element-wise multiplication, division, and inversion by $\odot, \oslash$ and $^{\odot -1}$ respectively. Then, the *distance function* between a final entity embedding and a target relation box is defined piece-wise over two cases, as follows:

$$\mathsf{dist}(\boldsymbol{e_i^{r(e_1,\ldots,e_n)}}, \boldsymbol{r^{(i)}}) = \begin{cases} \mid \boldsymbol{e_i^{r(e_1,\ldots,e_n)}} - \boldsymbol{c^{(i)}} \mid \oslash \boldsymbol{w^{(i)}} & \text{if } \boldsymbol{e_i} \in \boldsymbol{r^{(i)}}, \\ \mid \boldsymbol{e_i^{r(e_1,\ldots,e_n)}} - \boldsymbol{c^{(i)}} \mid \odot \boldsymbol{w^{(i)}} - \kappa & \text{otherwise}, \end{cases}$$

where $\kappa = 0.5 \odot (\boldsymbol{w^{(i)}} - 1) \odot (\boldsymbol{w^{(i)}} - \boldsymbol{w^{(i)^{\odot -1}}})$, is a width-dependent factor introduced to maintain function continuity.

We now study the dist function for both cases. In the first case, the final embedding of e_i is in the target box $\boldsymbol{r^{(i)}}$. Therefore, dist divides the distance to the box centre by the incremented width $\boldsymbol{w^{(i)}}$ of $\boldsymbol{r^{(i)}}$. As $\boldsymbol{w^{(i)}}$ by definition is greater than or equal to 1, this yields a lower score, which correlates inversely with the size of the target box. In particular, dividing by $\boldsymbol{w^{(i)}}$ sets a maximum value of $\frac{1}{2} - \frac{1}{2\boldsymbol{w^{(i)}}}$ at the box boundaries, and this maximum asymptotically approaches $\frac{1}{2}$ as the box grows. Hence, scores inside a box are always upper-bounded by $\frac{1}{2}$ per dimension, imposing a low absolute score for box membership irrespective of box size. Furthermore, the $\boldsymbol{w^{(i)}}$ quotient yields smaller (but non-zero) gradients, reflecting the higher similarity between close points when both are studied relative to larger boxes.

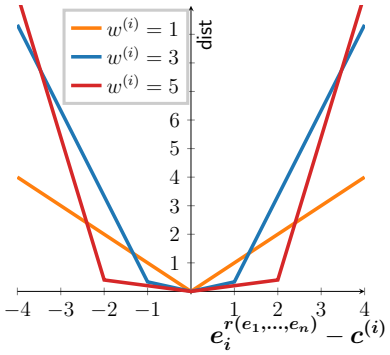


Figure 3.2: The dist function for width $\boldsymbol{w^{(i)}} = 1, 3, 5$.

In the second case, the final embedding of e_i falls outside $\boldsymbol{r^{(i)}}$, and thus dist *multiplies* the distance to the box centre by $\boldsymbol{w^{(i)}}$, increasing the score. Notice that this multiplication yields the opposite result as the first case: As the target box $\boldsymbol{r^{(i)}}$ grows, scores increase more rapidly, and thus a higher penalty is imposed on points falling outside of larger boxes. Therefore, entities whose final embeddings fail to appear in a larger box have larger gradients to help move them inside the target box during training, and negative samples exiting such boxes are also supported in the opposite direction.

Plots for dist for one-dimensional $\boldsymbol{w^{(i)}}$ for values 1, 3, and 5 are shown in Figure 3.2. Observe that, in the extreme case $\boldsymbol{w^{(i)}} = 1$, $\boldsymbol{r^{(i)}}$ has a true width of zero and is therefore point-shaped. In this scenario, dist reduces to standard $L1$ distance between the final embedding and the box centre, in this case the point-box itself. Conversely, as $\boldsymbol{w^{(i)}}$ increases, dist gives lower values (upper bounded by $\frac{1}{2}$) to the region inside the box and imposes more rapidly increasing scores for points outside this box. In fact, if we consider the opposite extreme where $\boldsymbol{w^{(i)}} \to +\infty$, then the dist decision boundary becomes very similar to a step function, which exactly represents the truth semantics of BoxE. The dist function thus achieves three objectives. First, it treats points inside the box preferentially to points outside the box. Second, it ensures that outside points receive high gradient through which they can more easily reach their target box, or escape it for negative samples. Third, it gives weight to boxes, and in particular their size, in distance computation, so as to offer a comprehensive scoring mechanism that is compatible both with BoxE truth modelling and with the smoothness requirements for a distance function.

Finally, we now define the overall BoxE scoring function as the sum of the L$-x$

norms of dist across all n entities and relation boxes, i.e.:

$$\mathsf{score}(r(e_1, ..., e_n)) = \sum_{i=1}^{n} \left\| \mathsf{dist}(\boldsymbol{e_i^{r(e_1,...,e_n)}}, \boldsymbol{r^{(i)}}) \right\|_x .$$

Generalisation. In addition to the earlier standard definition, where BoxE represents relations using one relation box per arity position, we also consider a more general definition, where *multiple* relation boxes are defined per arity position, to comprehensively capture entity *permutations* across arity positions. This generalisation is mainly of theoretical interest, and is instrumental to capture all permutations of entities for higher-arity facts (in Theorem 3.4.1). Therefore, we exclusively focus on the standard BoxE model unless explicitly stated otherwise.

3.4 Model Properties

In this section, we analyse the representation power and inductive capacity of BoxE. More specifically, we show that BoxE is fully expressive, and captures a rich rule language combining multiple inference patterns. We additionally show that BoxE can lucidly *incorporate* a given set of logical rules from a sub-language of this language, i.e., rule injection, and can naturally support and represent class information. Finally, we discuss the time and space complexity of BoxE.

3.4.1 Full Expressiveness

We prove that BoxE is fully expressive with a worst-case $d = |\mathbf{E}|^{n-1}|\mathbf{R}|$ dimensions for a knowledge base with relational vocabulary $(\mathbf{E}, \mathbf{R})$ and maximum fact arity n. For KGs, i.e., $n = 2$, this result reduces to $d = |\mathbf{E}||\mathbf{R}|$. Therefore, BoxE is fully expressive over KGs with dimensionality *linear* in $|\mathbf{E}|$. The proof uses translational bumps to make an arbitrary true fact F false, while preserving the correctness of other facts. This result requires a careful technical construction, which (i) pushes a single entity representation within F outside its corresponding relation box at a specific dimension, and (ii) modifies all other model embeddings to prevent a change in the truth value of any other fact. We note that this result makes BoxE the first translation-based model that is fully expressive. We now formally state and prove the full expressiveness theorem for knowledge graphs. Following this proof, we also include a BoxE generalisation to produce an expressiveness result for KBs.

Theorem 3.4.1. *BoxE is a fully expressive model with the embedding dimensionality d of entities, bumps, and relations set to $d = |\mathbf{E}||\mathbf{R}|$, where $\mathbf{R}$ consists of at most binary relations.*

Proof. We prove the result for knowledge graphs ($n = 2$), and then show how this can be lifted to arbitrary knowledge bases with higher-arity relations. The result is shown by induction: We start with a base case, with a KG G where all possible facts stemming from the relational vocabulary $(\mathbf{E}, \mathbf{R})$ are set to true, and show that BoxE can trivially represent this KG. Then, in the induction step, we show that a BoxE model with $d = |\mathbf{E}||\mathbf{R}|$ can make any arbitrary fact in G false without affecting other facts, i.e. previously true (resp., false) facts remain true (resp., false). In this step, F is made false by pushing the final embedding of its tail entity outside its corresponding relation box at a fact-specific dimension, dictated by the entities and relations appearing in F. Then, the BoxE configuration is adjusted so as to prevent a change in the truth value of any other fact.

Let us assume without loss of generality that all relations and entities are indexed. Specifically, we consider relations $r_i \in \mathbf{R}$, and entities $e_j \in \mathbf{E}$, where $i, j \in \mathbb{N}$, $0 \leq i \leq |\mathbf{R}| - 1$, and $0 \leq j \leq |\mathbf{E}| - 1$. Accordingly, we denote the relation boxes of r_i as $\boldsymbol{r_i^{(1)}}$ and $\boldsymbol{r_i^{(2)}}$, and denote their lower and upper bounds by $\boldsymbol{l_i^{(1)}}, \boldsymbol{u_i^{(1)}}$ and $\boldsymbol{l_i^{(2)}}, \boldsymbol{u_i^{(2)}}$ respectively. We consider d-dimensional embedding vectors $\boldsymbol{v}$ with $d = |\mathbf{E}||\mathbf{R}|$, and write $\boldsymbol{v}(i, j)$ to refer to the vector index $i|\mathbf{E}| + j$. Intuitively, in our construction, the sequence of indices $\boldsymbol{v}(i, 0), \ldots, \boldsymbol{v}(i, |\mathbf{E}| - 1)$ corresponds to a "chunk" of $\boldsymbol{v}$ reserved for the relation r_i.

Base case. We initialise the KG G as the whole universe of possible facts over $(\mathbf{E}, \mathbf{R})$. BoxE can trivially represent G, by setting all entity and bump vectors to 0, and all relation boxes at all arity positions as unit boxes centered at 0.

Induction step. In this step, we consider a true fact $r_i(e_j, e_k)$, and make this fact false without affecting the truth value of any fact in G. This is done as follows:

Step 1. Increment $\boldsymbol{b_j}(i, k)$, the bump of e_j at dimension (i, k), by a value C, such that:

$$\boldsymbol{e_k}(i, k) + (\boldsymbol{b_j}(i, k) + C) > \boldsymbol{u_i^{(2)}}(i, k).$$

Step 2. Decrement all entity base position embeddings except that of e_k by C at dimension (i, k):

$$\forall\ k' \neq k, \boldsymbol{e_{k'}}(i, k) := \boldsymbol{e_{k'}}(i, k) - C.$$

Step 3. For relation r_i, grow the head box $\boldsymbol{r_i^{(1)}}$ by C at dimension (i,k) both upwards and downwards, and grow the tail box $\boldsymbol{r_i^{(2)}}$ downwards by C in this dimension:

$$\boldsymbol{l_i^{(1)}}(i,k) = \boldsymbol{l_i^{(1)}}(i,k) - C,$$
$$\boldsymbol{u_i^{(1)}}(i,k) = \boldsymbol{u_i^{(1)}}(i,k) + C,$$
$$\boldsymbol{l_i^{(2)}}(i,k) = \boldsymbol{l_i^{(2)}}(i,k) - C.$$

Step 4. For all other relations $r_x \in \mathbf{R}, x \neq i$, grow all boxes by C at dimension (i,k) in both directions, that is, for $\beta \in \{1,2\}$:

$$\boldsymbol{l_x^{(\beta)}}(i,k) = \boldsymbol{l_x^{(\beta)}}(i,k) - C,$$
$$\boldsymbol{u_x^{(\beta)}}(i,k) = \boldsymbol{u_x^{(\beta)}}(i,k) + C.$$

Observe that Step 1 makes $r_i(e_j, e_k)$ false, by pushing $\boldsymbol{e_k^{r_i(e_j,e_k)}}$ outside of $\boldsymbol{r_i^{(2)}}$ at dimension (i,k) from above. This flips the truth value of $r_i(e_j, e_k)$, as required. We now show that the results of Steps 1 and 2, combined with the changes to relation boxes made in Steps 3 and 4, which affect facts involving r_i and other relations respectively, preserve the correctness/falsehood of all facts other than $r_i(e_j, e_k)$. To this end, we consider any possible fact $F = r_{i'}(e_{j'}, e_{k'})$ from the KG, and analyse the effect of the induction step at the head and tail of the fact. We consider the following cases:

Case 1. **The fact F is true:** To verify that $F = r_{i'}(e_{j'}, e_{k'})$ remains true after the inductive step, we analyse both the final head entity embedding $\boldsymbol{e_{j'}^F}$ and the final tail entity embedding $\boldsymbol{e_{k'}^F}$.

(a) **Head entity:** Observe that (i) $\boldsymbol{e_{j'}^F}$ can change by at most C following Steps 1 and 2, and (ii) all relation head boxes are grown by C in both directions in Steps 3 and 4. These together imply that $\boldsymbol{e_{j'}^F} \in \boldsymbol{r^{(1)}}$ is guaranteed to hold provided that it was true before the induction step.

(b) **Tail entity:** If $e_{k'} \neq e_k$, then $\boldsymbol{e_{k'}^F}$ is not changed if $e_{j'} = e_j$, and decremented by C at dimension (i,k) if $e_{j'} \neq e_j$. Hence, the changes to both $\boldsymbol{r_i^{(2)}}$ and $\boldsymbol{r_x^{(2)}}, x \neq i$ in Steps 3 and 4 are sufficient to maintain $\boldsymbol{e_{k'}} \in \boldsymbol{r^{(2)}}$. Conversely, if $e_{k'} = e_k$, then $\boldsymbol{e_t^F}$ is unchanged when $e_{j'} \neq e_j$, and thus $\boldsymbol{e_{k'}} \in \boldsymbol{r^{(2)}}$ still holds. Otherwise, when $e_{j'} = e_j$, $\boldsymbol{e_{k'}^F}$ is incremented by C, which, for $r = r_i$, makes F false, as required, and for $r \neq r_i$, still keeps $\boldsymbol{e_{k'}} \in \boldsymbol{r^{(2)}}$, as all other tail boxes are grown upwards by C.
Hence, for any true fact in G, except the fact $r_i(e_j, e_k)$, we conclude that $\boldsymbol{e_{j'}} \in \boldsymbol{r^{(1)}}$ and $\boldsymbol{e_{k'}} \in \boldsymbol{r^{(2)}}$ continues to hold after the induction step, as required.

Case 2. **The fact F is false:** To verify that $F = r_{i'}(e_{j'}, e_{k'})$ remains false after the inductive step, we again consider the final head and tail entity embeddings.

(a) **Head entity:** By construction, all false facts $F = r_{i'}(e_{j'}, e_{k'})$ satisfy the inequality:

$$\boldsymbol{e}^{\boldsymbol{F}}_{\boldsymbol{k'}}(i', k') > \boldsymbol{u}^{(\mathbf{2})}_{\boldsymbol{i'}}(i', k'),$$

and any changes to $\boldsymbol{e}^{\boldsymbol{F}}_{\boldsymbol{j'}}$ do not affect this inequality.

(b) **Tail entity:** If $e_{k'} \neq e_k$, then F verifies $\boldsymbol{e}^{\boldsymbol{F}}(i', k') > \boldsymbol{u}^{(\mathbf{2})}_{\boldsymbol{i'}}(i', k')$, where $k' \neq k$. This inequality continues to hold regardless of the changes to $\boldsymbol{e}^{\boldsymbol{F}}_{\boldsymbol{k'}}(i, k)$. Otherwise, if $e_{k'} = e_k$, and $r_{i'} = r_i$, then $e_{j'} \neq e_j$, as F is initially false, and $r_i(e_j, e_k)$ is initially true. Furthermore, in this setting, since $e_{j'} \neq e_j$, $\boldsymbol{e}^{\boldsymbol{F}}_{\boldsymbol{k'}}$ is unchanged as $k = k'$, and this therefore maintains the falsehood inequality. Finally, if $r_{i'} \neq r_i$, then the falsehood inequality for F holds at a dimension different than (i, k). Therefore, none of the changes in the induction step affect this fact falsehood inequality.
Hence, all false facts in G remain false after the induction step, as required.

Thus, the induction step can make any true fact $r_i(e_j, e_k)$ in knowledge graph G false in a BoxE model with $d = |\mathbf{E}||\mathbf{R}|$ without affecting the remainder of the facts in G. Hence, all fact configurations are possible and expressible by such a BoxE model, and this model is fully expressive, as required.

This proof can be generalised from knowledge graphs to higher-arity knowledge bases using the relation box extension mentioned at the end of Section 3.3. Indeed, in a higher-arity setting ($n > 2$), the number of possible entity permutations in an n-tuple is exponential ($n!$). Therefore, we propose an extended BoxE whereby we define $(n-1)!$ relation boxes per arity position, and thus $n!$ relation boxes in total, namely $\boldsymbol{r}^{(\boldsymbol{a},\boldsymbol{\pi})}$, where $a \in \{1, ..., n\}$ represents arity position and $\pi \in \{1, ..., n-1!\}$ is injective to a given ordering of the other $n-1$ entities in the fact. Using these boxes, BoxE can then capture the truth configuration with respect to *any* entity order given an entity n-tuple in higher-arity.

For a maximum arity of n, a dimensionality $d = |\mathbf{E}|^{n-1}|\mathbf{R}|$ is needed for full expressiveness with this generalised model. The proof then proceeds as follows:

1. We define a higher-arity indexing function $(\theta_1, \theta_2, ..., \theta_n)$, which refers to vector index $\sum_{a=1}^{n} |\mathbf{E}|^{n-a}\theta_a$, such that θ_1 is a relation index analogous to i in our earlier proof. From here, we now seek to make facts false at the *last* arity position, analogously to the tail in the original proof. As per the extended relation box definition, we assume an ordering over the set of entities $\mathbf{E}$, and use this ordering to induce relation boxes $\mathbf{r}^{(\mathbf{i},\pi)}$ at every arity position i, based on the permutation π observed across entities in all other $n-1$ arity positions.

2. Given an n-ary fact $r_i(e_{j_1}, \ldots, e_{j_n})$, we also increment the bump of e_{j_1} by C, and decrement the entity representations of all entities except e_{j_n} by C.

3. Given $r_i(e_{j_1}, \ldots, e_{j_n})$, we consider the $(n-1)$-tuple $e_{j_1}, \ldots, e_{j_{n-1}}$ and follow its unique ordering π to infer the relation box used at arity position n, $\boldsymbol{r^{(n,\pi)}}$. Then, we grow $\boldsymbol{r^{(n,\pi)}}$ downwards by C only, analogously to $\boldsymbol{r^{(2)}}$ in Step 3, and grow $\boldsymbol{r^{(a',\pi')}}$ upwards and downwards by C, for any $a' \in \{1, \ldots, n-1\}$ (as well as n for $\pi' \neq \pi$) and $\pi' \neq \pi$.

4. Grow all other boxes as in Step 4.

This proof can be trivially extended to knowledge bases with non-uniform arities (i.e., KBs containing relations with different arities) by introducing extra parameters to relations of lower arity, and setting the correctness of the n-arity facts solely based on the original facts.

□

Remark. Despite theoretically introducing exponential reliance on n, the generalised BoxE setup does not cause substantial over-parametrisation, as (i) the maximum arity n is typically *fixed* and *small*, i.e., typically not exceeding 5-6, (ii) the number of relations $|\mathbf{R}|$ is also small, i.e. in the hundreds. Furthermore, in the binary setting ($n = 2$), this setup reduces exactly to standard BoxE. However, this extension introduces an explicit reliance on entity orderings and restricts parameter sharing in relation representations. Moreover, the expressiveness gain from this extension is very limited in practice, as entity tuples with highly variable truth configurations are extremely rare. Hence, we use the standard BoxE model defined in Section 3.3 throughout the remainder of this chapter, including in the higher-arity KB experiments in Section 3.5.4.

3.4.2 Inference Patterns and Generalisations

We now study the inductive capacity of BoxE in terms of common inference patterns over KGs appearing in the KGC literature, and compare it with earlier models. First, we recall the formal definitions for some key inference patterns over knowledge graphs:

- A *symmetry* pattern is a rule of the form $r_1(x, y) \Rightarrow r_1(y, x)$, where $r_1 \in \mathbf{R}$. This pattern encodes that a relation between two entities holds in both directions. For example, the friends relation in social networks is typically symmetric.

- An *anti-symmetry* pattern is a rule of the form $r_1(x, y) \Rightarrow \neg r_1(y, x)$, where $r_1 \in \mathbf{R}$. Anti-symmetry is the opposite of symmetry, in that an anti-symmetric relation between two entities can only hold in one direction. For instance, the supervises relation between a professor and their student is anti-symmetric.

- An *inversion* pattern is a rule of the form $r_1(x, y) \Leftrightarrow r_2(y, x)$, where $r_1 \neq r_2 \in \mathbf{R}$. An example of two relations following the pattern is supervises and supervisedBy.

- A *composition* pattern is a rule of the form $r_1(x, y) \wedge r_2(y, z) \Rightarrow r_3(x, z)$, where $r_1 \neq r_2 \neq r_3 \in \mathbf{R}$. This pattern is analogous to relation composition, e.g., $\mathsf{aunt}(x, z)$ is a composition of $\mathsf{sister}(x, y)$ and $\mathsf{mother}(y, z)$.

- A *hierarchy* pattern is a rule of the form $r_1(x, y) \Rightarrow r_2(x, y)$,where $r_1 \neq r_2 \in \mathbf{R}$. Hierarchy implies that when r_1 holds for any given pair of entities, then r_2 necessarily holds. For example, $\mathsf{friends}(x, y)$ and $\mathsf{knows}(x, y)$ are clearly connected by a hierarchy rule.

- An *intersection* pattern is a rule of the form $r_1(x, y) \wedge r_2(x, y) \Rightarrow r_3(x, y)$, where $r_1 \neq r_2 \neq r_3 \in \mathbf{R}$. Intersection is a generalisation of hierarchy, in that two relations r_1, r_2 (as opposed to just r_1 for hierarchy) simultaneously holding between two entities imply a third relation r_3 holding. For instance, $\mathsf{worksFor}(x, y)$ and $\mathsf{hasSharesIn}(x, y)$ implies $\mathsf{partnerOf}(x, y)$.

- A *mutual exclusion rule* is of the form $r_1(x, y) \wedge r_2(x, y) \Rightarrow \bot$, where $r_1 \neq r_2 \in \mathbf{R}$. Intuitively, mutual exclusion implies that r_1 and r_2 cannot simultaneously hold over the same entity pairs, e.g., $\mathsf{livesIn}(x, y)$ and $\mathsf{spouse}(x, y)$.

In what follows, we say that a model *captures* an inference pattern if it admits a set of parameters *exactly* and *exclusively* satisfying the pattern, in keeping with the literature [159]. More formally, a model $\mathcal{M}$ captures an inference pattern σ if there exists a parametrisation of the relation representations of $\mathcal{M}$ such that:

1. (Exactness)$\mathcal{M}$ satisfies σ.

2. (Exclusiveness) For any other rule σ', $\mathcal{M}$ satisfies σ' if and only if $\sigma \models \sigma'$, i.e., σ entails σ'.

To illustrate, consider the TransE model: TransE can capture composition [21, 159] between r_1, r_2, r_3 by setting $\boldsymbol{r_1} + \boldsymbol{r_2} = \boldsymbol{r_3}$, but cannot capture hierarchy, as in this

Table 3.1: Inference patterns/generalised inference patterns captured by selected KBC models. TuckER coincides with ComplEx, so is omitted from the table.

Inference pattern	BoxE	TransE	RotatE	DistMult	ComplEx
Symmetry: $r_1(x,y) \Rightarrow r_1(y,x)$	✓/✓	✗/✗	✓/✓	✓/✓	✓/✓
Anti-symmetry: $r_1(x,y) \Rightarrow \neg r_1(y,x)$	✓/✓	✓/✓	✓/✓	✗/✗	✓/✓
Inversion: $r_1(x,y) \Leftrightarrow r_2(y,x)$	✓/✓	✓/✗	✓/✓	✗/✗	✓/✓
Composition: $r_1(x,y) \wedge r_2(y,z) \Rightarrow r_3(x,z)$	✗/✗	✓/✗	✓/✗	✗/✗	✗/✗
Hierarchy: $r_1(x,y) \Rightarrow r_2(x,y)$	✓/✓	✗/✗	✗/✗	✓/✗	✓/✗
Intersection: $r_1(x,y) \wedge r_2(x,y) \Rightarrow r_3(x,y)$	✓/✓	✓/✗	✓/✗	✗/✗	✗/✗
Mutual exclusion: $r_1(x,y) \wedge r_2(x,y) \Rightarrow \bot$	✓/✓	✓/✓	✓/✓	✓/✗	✓/✗

model, $r_1(x,y) \Rightarrow r_2(x,y)$ holds only if $\boldsymbol{r_1} = \boldsymbol{r_2}$, and thus $r_2(x,y) \Rightarrow r_1(x,y)$, leading to loss of generality.

Remark. The requirement for setting distinct relations in pattern definitions, e.g., composition, is in keeping with conventions in the literature. This may appear counter-intuitive, as it limits the generality of the studied patterns, but it is necessary for the study of single inference patterns, as special cases with identical relations can go beyond the scope of what current models can capture. To illustrate, the composition pattern, captured by TransE (and RotatE), can only be captured when $r_1 \neq r_2 \neq r_3$. Indeed, if we set $r_1 = r_3$, then composition reduces to transitivity: $r_1(x,y) \wedge r_2(y,z) \Rightarrow r_1(x,z)$. Such a rule, though similar in shape to composition, cannot be captured by TransE, as setting $\boldsymbol{r_1} + \boldsymbol{r_2} = \boldsymbol{r_1}$ implies $\boldsymbol{r_2} = 0$, and thus that $r_2(x,x)$ holds for any interpretation of x. Note, however, that using finitely many such rules, i.e., generalised inference patterns and rule languages (discussed later in this section) recovers full generality. For instance, the rules $r_1(x,y) \wedge r_2(y,z) \Rightarrow r_3(x,z)$, and $r_3(x,y) \Rightarrow r_1(x,y)$ together simulate the transitivity rule given above.

First, we compare the inductive capacity of BoxE in terms of capturing individual inference patterns against prominent models in the KGC literature, namely TransE, RotatE, DistMult and ComplEx. This comparison is shown in Table 3.1, with single pattern capturing indicated by a tick or cross on the left-hand side of every cell entry. From this table, we observe that BoxE captures all patterns except composition, indicating a strong inductive bias. Conversely, bilinear models (DistMult, ComplEx) fail to capture both composition and intersection patterns, whereas translational models (TransE, RotatE) fail to capture hierarchy, with TransE also unable to capture relation symmetry.

Studying individual inference patterns offers some insights as to the inductive capacity of shallow node embedding models, and is the main approach currently

adopted in the literature. However, this perspective for studying inductive capacity is unrealistic and incomplete, as knowledge graphs with only a single inference pattern are highly unlikely in practice. In fact, it is clear upon closer inspection that simply considering single inference patterns is not sufficient, and multiple inference patterns can interact in a KG. Furthermore, combinations of patterns can lead to certain fact configurations that are beyond what certain node embedding models can represent. Therefore, we pose the following question: Can KGC models capture multiple, distinct instances of the same inference pattern jointly?

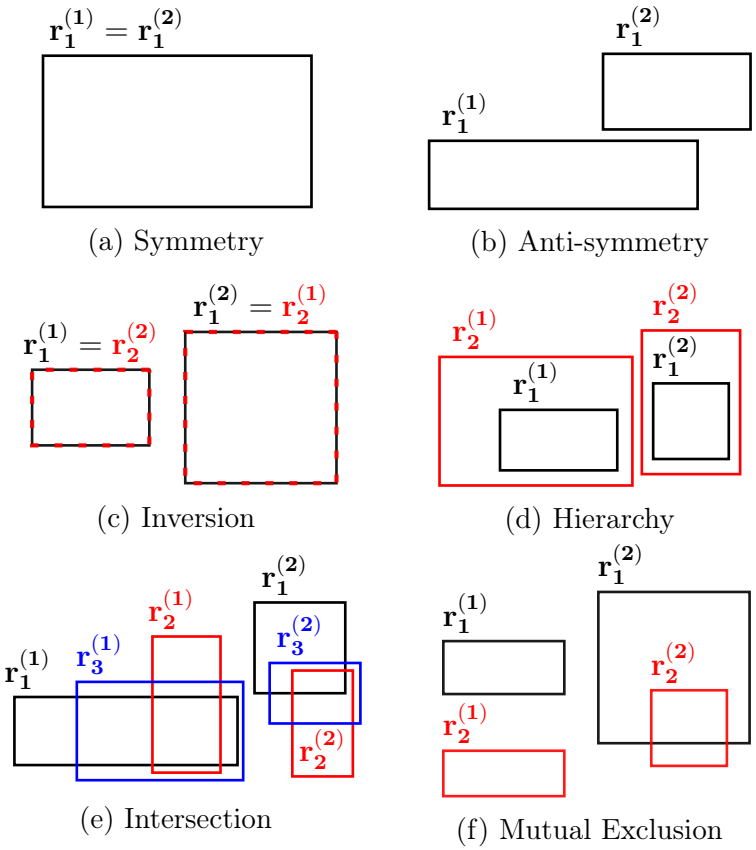
Immediately, one can observe that capturing multiple inference patterns jointly is significantly more challenging. For instance, we know that TransE can capture the composition rules $r_1(x, y) \wedge r_2(y, z) \Rightarrow r_3(x, z)$ and $r_1(x, y) \wedge r_4(y, z) \Rightarrow r_3(x, z)$ separately, by setting $\boldsymbol{r_1} + \boldsymbol{r_2} = \boldsymbol{r_3}$ and $\boldsymbol{r_1} + \boldsymbol{r_4} = \boldsymbol{r_3}$, respectively. However, jointly capturing both composition rules imposes that both earlier equations hold, and together these equations incorrectly imply $\boldsymbol{r_2} = \boldsymbol{r_4}$, which leads to a loss of generality. Hence, TransE can only capture single instances of composition rules. Similarly, bilinear models can capture the hierarchy rules $r_1(x, y) \Rightarrow r_3(x, y)$ and $r_2(x, y) \Rightarrow r_3(x, y)$ separately, but jointly capturing them incorrectly imposes a third hierarchy, either $r_1(x, y) \Rightarrow r_2(x, y)$ or $r_2(x, y) \Rightarrow r_1(x, y)$ [72]. These examples are clearly not edge cases, and highlight severe limitations in how the inductive capacity of KBC models is analysed. Therefore, we propose *generalised inference patterns* as follows:

Definition 3.4.1. *A* rule *is in one of the forms given in Table 3.1. To distinguish between types of rules, we write* ρ-rule*, where* $\rho \in \{$*symmetry, . . . , mutual exclusion*$\}$*. A* generalised ρ*-pattern is a finite set of* ρ*-rules over* $\mathbf{R}$*.*

Intuitively, generalised inference patterns characterise model inductive capacity relative to a finite *set* of rules of a same pattern. Indeed, every generalised inference pattern defines a trivial *rule language*, consisting of a single type of rule, which a model must then exactly and exclusively capture. Hence, the study of generalised inference patterns includes the examples presented above, and provides a unifying framework for qualifying model inductive capacity.

Using generalised inference patterns, we now study the inductive capacity of BoxE, as well as the aforementioned representative KGC models TransE, RotatE, DistMult, ComplEx, and TuckER. We report generalised inference patterns captured by all these models in Table 3.1, and prove all results, both for single and generalised inference patterns, in the following theorem.

Theorem 3.4.2. *All the results given in Table 3.1 for BoxE and other models hold.*



(a) Symmetry (b) Anti-symmetry

(c) Inversion (d) Hierarchy

(e) Intersection (f) Mutual Exclusion

Figure 3.3: Sample box configurations ($d = 2$) for the (generalised) inference patterns captured by BoxE. Relation indices align with the rule definitions in Table 3.1. Relation boxes for r_1, r_2, and r_3 are shown in black, red, and blue, respectively.

Intuitively, BoxE captures single and generalised inference patterns through *box configurations*. For instance, BoxE captures (generalised) symmetry by setting the 2 boxes for a relation r to be equal, and captures (generalised) inverse relations r_1 and r_2 by setting $\boldsymbol{r_1^{(1)}} = \boldsymbol{r_2^{(2)}}$ and $\boldsymbol{r_1^{(2)}} = \boldsymbol{r_2^{(1)}}$. Hierarchies are captured through box subsumption, i.e., $\boldsymbol{r_1^{(1)}}$ and $\boldsymbol{r_1^{(2)}}$ contained in $\boldsymbol{r_2^{(1)}}$ and $\boldsymbol{r_2^{(2)}}$ respectively, and this extends to intersection in the usual sense. Finally, anti-symmetry and mutual exclusion, are captured through disjointness between relation boxes. We highlight this with box configurations for all the aforementioned patterns in Figure 3.3.

We prove Theorem 3.4.2 in three separate lemmas. First, we study BoxE and prove its inference pattern results. Second, we show results for translational models TransE and RotatE. Third, we prove results for bilinear models DistMult, ComplEx, and TuckER.

Lemma 3.4.1. *BoxE captures generalised symmetry, anti-symmetry, inversion, hierarchy, intersection, and mutual exclusion rules.*

Proof. We show that each generalised inference pattern can be captured by BoxE except for composition (and argue why BoxE cannot capture this explicitly).

Generalised intersection. We first introduce the concept of *boxicity*. Let $G(V, E)$ be a graph, where V is the set of nodes, and E is the set of edges. The *boxicity* of G is the minimum embedding dimension in which G can be represented as an intersection of axis-aligned boxes, such that (i) every box corresponds to a specific node, (ii) boxes intersect iff an edge connects their respective nodes [142]. It has been shown that the boxicity of a graph with p edges is $O(\sqrt{p \cdot \log(p)})$ [35]. This implies that, given a graph G, where every relation $r \in \mathbf{R}$ is represented as a node in V, and every edge between them represents an intersection, any finite combination of intersections between relations can be represented in a finite-dimensional vector space of worst-case dimensionality $O(|\mathbf{R}|\sqrt{\log(|\mathbf{R}|)})$, corresponding to the fully connected graph with $|\mathbf{R}|$ nodes and $O(|\mathbf{R}|^2)$ edges.

For a given knowledge graph G, we define a *relation intersection graph*. That is, for every relation $r \in \mathbf{R}$, we define two nodes, corresponding to its head and tail boxes, and then set edges in the graph based on desired intersections between relation boxes, which are dictated by intersection rules.

Prior to encoding rules into relation intersection graph edges, we first compute the *deductive closure* of the set of intersection rules. In other words, we check whether any rule of the form $r_i(x, y) \Rightarrow r_j(x, y)$, or $r_i(x, y) \wedge r_j(x, y) \Rightarrow r_k(x, y)$ for any i, j, k can be entailed from the given set of rules, and keep adding new rules to this initial set until no more rules can be deduced. That is, we compute the logical closure of the initial set. This allows us to make all possible relation intersections explicit.

Then, we map all rules in the computed deductive closure to edges as follows:

- For every intersection rule $r_1(x, y) \wedge r_2(x, y) \Rightarrow r_3(x, y)$, we set edges between the node corresponding to the head of r_3 and those of r_1 and r_2, with the same done for tail nodes.

- For every deduced hierarchy rule $r_1(x, y) \Rightarrow r_2(x, y)$, we set edges between the head nodes of r_2 and r_1, with the same done for tail nodes.

With the resulting relation intersection graph G, we have encoded necessary conditions for all rules to hold, namely that relations whose intersections are contained in other relations intersect with these relations. We now leverage the boxicity argument, and show that there exists a box configuration of finite dimensionality capturing all the intersections encoded in G. This box configuration captures all intersections needed between the respective boxes for the rules to hold, but is not necessarily sufficient to

capture hierarchies and box containment. Hence, we modify the aforementioned box configuration using a procedure, which we apply iteratively over every intersection rule, such that the final configuration provably captures all rules, without capturing additional undesired rules.

Our box reconfiguration procedure is as follows:

1. Iterate over every intersection rule $r_1(x, y) \wedge r_2(x, y) \Rightarrow r_3(x, y)$:

   (a) If the r_3 head and tail boxes do not contain the head and tail box intersections $r_1 \cap r_2$, then we grow these r_3 boxes by the minimum possible amount to make this condition hold and establish the rule. In other words, we grow the r_3 boxes at every position to equal the boundary of either the r_1 boxes or the r_2 boxes at the dimensions where the rule does not hold due to r_1 or r_2. This growth operation preserves all existing edges in G, and does not force new intersections, as all forced intersections due to rule capturing are already encoded by the existing edges.

   (b) Following Part (a), the growth of r_3 can violate another rule in the set, in particular if r_3 is in the body of this rule. Hence, when any r_3 boxes are grown in Part (a), check all other intersection rules in the rule set: If the change in r_3 makes a rule no longer hold (i.e., the rule was captured prior to growing $\boldsymbol{r_3}$ and no longer is), then recursively call this procedure for this rule.

We now show that this procedure is correct, and then prove that it terminates, particularly with respect to the number of recursive calls made. First, we note that, following a successful iteration on a given rule, a rule is successfully captured (Part (a)), and no other rules are violated in the process (Part (b)). Thus, the final configuration returned by this procedure over the initial boxicity-given configuration is a valid BoxE configuration. In particular, this configuration captures all and only the provided patterns within the deductive closure, which includes the original intersection rules. Furthermore, growing r_3 boxes in the configuration to satisfy an intersection rule does not induce any rules outside their deductive closure. Indeed, when r_3 boxes are grown, they are only grown in dimensions where they fail to capture $\boldsymbol{r_1} \cap \boldsymbol{r_2}$. Thus, the procedure can only make $\boldsymbol{r_3}$ intersect with boxes that intersect with $\boldsymbol{r_1}$ or $\boldsymbol{r_2}$. As a result, the procedure can only force intersections between boxes within the deductive closure of the rule set.

It now remains to show that this procedure terminates, and thus that a configuration of this kind indeed can be found. In particular, we study the maximal number of recursive calls needed. Consider a rule $\rho : r_1(x, y) \wedge r_2(x, y) \Rightarrow r_3(x, y)$, where r_3 boxes are grown. For simplicity, we only consider a single box for r_3, i.e., a unique arity position, as the analysis is analogous at every arity position. We define *boundaries* as being the lower and upper limits of a box at every dimension. Thus, a d-dimensional box has $2d$ boundaries. Therefore, in our binary BoxE configuration with $|\mathbf{R}|$ relations and $d = O(|\mathbf{R}|\sqrt{\log |\mathbf{R}|})$, there are $O(|\mathbf{R}|^2\sqrt{\log |\mathbf{R}|})$ boundaries. For our analysis, we are interested in the number of *distinct* boundaries in our configuration.

We now consider the effect of an application of a call to Part (a) of the procedure on the number of distinct boundaries. If ρ is already captured, then no action is needed. Otherwise, $\boldsymbol{r_3}$ needs to be grown. Hence, in this scenario, there exists at least one dimension in which the lower (resp., upper) boundary of $\boldsymbol{r_3}$ is strictly higher (resp., lower) than the maximum (resp., minimum) lower (resp., upper) bound of either $\boldsymbol{r_1}$ or $\boldsymbol{r_2}$. Therefore, when $\boldsymbol{r_3}$ is grown, the value of the problematic bound(s) at this dimension is made equal to the corresponding bound(s) of $\boldsymbol{r_1}$ or $\boldsymbol{r_2}$. As a result, the number of distinct boundaries is guaranteed to strictly drop by at least 1 following any growth operation.

Furthermore, we consider the recursive calls made in Part (b), after any growth to $\boldsymbol{r_3}$. Observe that recursion is only called when the change to $\boldsymbol{r_3}$ *exclusively* makes the checked rule false. This condition ensures that all recursive calls are made *only* when the growing of the rule head boxes, in this case $\boldsymbol{r_3}$, is the *exclusive* cause for rule violation, and so eliminates all other possible causes of violation such that they are handled only when the outer loop reaches the corresponding rule, which greatly simplifies the analysis. Finally, we observe that box growth can only be triggered when distinct boundaries exist. Hence, when the number of distinct boundaries drops to its (highly pessimistic and loose) minimum possible value of 1, no more recursive calls can be made. This observation, combined with the earlier finding that every box growth strictly reduces the number of distinct boundaries by at least 1, implies that the number of recursive calls in this procedure is upper bounded by $O(|\mathbf{R}|^2\sqrt{\log |\mathbf{R}|})$. Hence, this procedure terminates, and a BoxE configuration capturing generalised intersections exists.

Generalised hierarchy. The proof for generalised intersection immediately applies to generalised hierarchies, as hierarchy is a special case of intersection where $r_1 = r_2$.

Generalised symmetry. The symmetry inference pattern is a single-relation pattern, and can appear at most once per relation. Symmetry can be easily captured for

a relation r by setting $\boldsymbol{r^{(1)}}$ and $\boldsymbol{r^{(2)}}$ to be identical boxes. This can be independently done for any relation, and thus BoxE captures generalised symmetry.

Generalised anti-symmetry. Analogously to generalised symmetry, anti-symmetry is a single-relation pattern. This pattern is captured by setting $\boldsymbol{r^{(1)}}$ and $\boldsymbol{r^{(2)}}$ to be disjoint for every anti-symmetric r. Therefore, BoxE captures generalised anti-symmetry.

Generalised inversion. An inversion pattern $r_1(x, y) \Leftrightarrow r_2(y, x)$ can be captured by setting $\boldsymbol{r_1^{(1)}}$ and $\boldsymbol{r_2^{(2)}}$, as well as $\boldsymbol{r_1^{(2)}}$ and $\boldsymbol{r_2^{(1)}}$, to be identical boxes. This box sharing between inverse relations can easily be extended to any arbitrary set of inversion rules.

Generalised mutual exclusion. It is sufficient to observe that there exists a BoxE configuration for any arbitrary set of mutual exclusion rules due to the boxicity argument: Simply consider a graph G with no edges connecting mutually exclusive relations. A simpler argument can be given directly: Generalised mutual exclusion can be achieved by making one of the relation boxes (head, or tail) disjoint in a fixed-dimensional space.

(Generalised) composition. Consider the composition pattern $r_1(x, y) \wedge r_2(y, z) \Rightarrow r_3(x, z)$. In this pattern, we see that the entity that will appear in lieu of variable x will be bumped differently in every atom, as it appears with different entities. More concretely, if we replace variables x, y, z with entities e_1, e_2, e_3 respectively, then $\boldsymbol{e_1^{r_1}} = \boldsymbol{e_1} + \boldsymbol{b_2}$ and $\boldsymbol{e_1^{r_3}} = \boldsymbol{e_1} + \boldsymbol{b_3}$. We can also view bumps as equivalently applying to boxes, i.e., instead of $\boldsymbol{e_1} + \boldsymbol{b_2} \in \boldsymbol{r_1^{(1)}}$, we write $\boldsymbol{e_1} \in \boldsymbol{r_1^{(1)}} - \boldsymbol{b_2}$. Hence, it is equivalent to view BoxE as bumping relation boxes in the opposite direction.

Now, we can see that $\boldsymbol{r_1^{(1)}}$ is bumped by $-\boldsymbol{b_2}$, whereas $\boldsymbol{r_3^{(1)}}$ is bumped by $-\boldsymbol{b_3}$. Therefore, since bumps are entity-specific and unknown a priori since the bump stems from an abstract variable, one cannot analyse the relative positions of $\boldsymbol{r_1^{(1)}}$ and $\boldsymbol{r_3^{(1)}}$ and draw conclusions. By contrast, all other captured rules in BoxE are such that relation boxes corresponding to the same variable are bumped identically, which neutralises the effect of bumping and enables the capturing of the patterns. Hence, translational bumps, which allow BoxE to be fully expressive, prevent BoxE from capturing compositions.

□

Lemma 3.4.2. *All the results given in Table 3.1 for TransE, RotatE hold.*

Proof. We note that some results stated below are taken from the literature, but we include them for completeness. The novel results are given for the generalised inference patterns.

For convenience, we first recall the scoring functions of TransE and RotatE. Given a fact $r(e_1, e_2)$, TransE computes the score as:

$$\mathsf{score}(r(e_1, e_2)) = \|\boldsymbol{e_1} + \boldsymbol{r} - \boldsymbol{e_2}\|,$$

whereas for RotatE,

$$\mathsf{score}(r(e_1, e_2)) = \|\boldsymbol{e_1} \circ \boldsymbol{r} - \boldsymbol{e_2}\|,$$

where $\circ$ denotes the element-wise rotation operator. We now show results for TransE and RotatE relative to single and generalised inference patterns.

Hierarchy. Let $\mathcal{M}_r = \mathsf{score}_r^{-1}([0, \epsilon])$, where score_r^{-1} is the inverse map of score with respect to a fixed relation r, which computes the subset of embedding vector pairs $(\boldsymbol{v}, \boldsymbol{w}) \in \mathbb{R}^d \times \mathbb{R}^d$ yielding scores of at most ϵ via r. In other words, $\mathcal{M}_r$ indicates the decision region of the relation r with margin ϵ in the latent embedding space. As a result, $r_1(x, y) \Rightarrow r_2(x, y)$ holds iff $\mathcal{M}_1 \subset \mathcal{M}_2$. In TransE (resp., RotatE), $(\boldsymbol{e_1}, \boldsymbol{e_2}) \in \mathcal{M}_r$ if $\boldsymbol{e_1} + \boldsymbol{r} - \boldsymbol{e_2} \in D_\epsilon(0)$ (resp., $\|\boldsymbol{e_1} \circ \boldsymbol{r} - \boldsymbol{e_2}\| \in D_\epsilon(0)$), where $D_\epsilon(0)$ is the disk of centre 0 and radius ϵ. Since it is necessary that $M_1 \subset M_2$, we require that the disk $D_{1,\epsilon}(\boldsymbol{e_1} + \boldsymbol{r_1})$ (resp., $D_{1,\epsilon}(\boldsymbol{e_1} \circ \boldsymbol{r_1})$) and radius ϵ is contained in the corresponding disk D_2, defined analogously using r_2. Since D_1 and D_2 have the same margin-induced radius, this is only possible if $r_1 = r_2$, effectively enforcing relation equivalence. Thus, neither translational model can capture (neither single nor generalised) hierarchies.

Intersection. A model can represent the intersection pattern $r_1(x, y) \wedge r_2(x, y) \Rightarrow r_3(x, y)$ if $\mathcal{M}_1 \cap \mathcal{M}_2 \subset \mathcal{M}_3$. In TransE and RotatE, this is satisfied (for $\epsilon > 0$) if r_3 lies in the centre of the disk intersection of $D_\epsilon(r_1)$ and $D_\epsilon(r_2)$, thus both models capture a single intersection pattern. Notice however that this construction relies on ϵ. Indeed, if both models are allowed no error ($\epsilon = 0$, i.e., they must fit all facts with score 0), then they cannot capture intersection, as this would then only hold iff $\boldsymbol{r_1} = \boldsymbol{r_2} = \boldsymbol{r_3}$. Furthermore, even with $\epsilon > 0$, both TransE and RotatE fail to capture generalised intersection. In particular, if we consider rules $r_1(x, y) \wedge r_2(x, y) \Rightarrow r_3(x, y)$ and $r_3(x, y) \wedge r_2(x, y) \Rightarrow r_1(x, y)$, the rule $r_2(x, y) \Rightarrow r_1(x, y)$ is logically implied. However, this is a hierarchy rule which cannot be captured by either model, as described earlier. Hence, TransE and RotatE cannot capture generalised intersections.

Notice that our findings align with the earlier remark about the use of distinct relations in the inference pattern definitions: Despite hierarchy being a special case of intersection (corresponding to $r_1 = r_2$), both TransE and RotatE are unable to fit it, even with arbitrary ϵ, whereas single intersection can be captured with $\epsilon > 0$.

This stems from the greater flexibility afforded by having three distinct relation embeddings, which enable disks to not share an identical centre for single intersection. Hence, the requirement for distinct relations is justified, as it highlights subtle inferential nuances within knowledge graph completion models.

Symmetry. In TransE, $r(x, y) \Rightarrow r(y, x)$ holds iff $\boldsymbol{r} = 0$, which implies that r is reflexive. Thus, TransE does not capture symmetry. By contrast, in RotatE, symmetry is captured iff $r = \{\pm k\pi\}^d$, $k \in \mathbb{N}$, i.e., a rotation vector consisting exclusively of multiples of π. Symmetry is a single-relation pattern, and thus multiple rules, affecting different relations, can be captured independently. Hence, RotatE captures generalised symmetry.

Anti-symmetry. In TransE, a relation r is anti-symmetric iff $\|\boldsymbol{r}\| \geq \epsilon$. The result for RotatE is proven in the original work [159]. As anti-symmetry is a single-relation pattern, it can be applied independently across all relations. Thus, both TransE and RotatE capture generalised anti-symmetry.

Inversion. For both TransE and RotatE, inversion holds iff $\boldsymbol{r_1} = -\boldsymbol{r_2}$ (resp., $\boldsymbol{r_2} = \bar{\boldsymbol{r_1}}$, where the bar denotes the complex conjugate). However, whereas RotatE can capture generalised inversion through repeated application of the earlier equation across all inversion rules, since it can handle any deduced symmetry results, TransE cannot. More concretely, consider the rule set $r_1(x, y) \Leftrightarrow r_2(y, x)$, $r_2(x, y) \Leftrightarrow r_3(y, x)$, $r_3(x, y) \Leftrightarrow r_1(y, x)$. This rule set implies $r_1(x, y) \Leftrightarrow r_1(y, x)$, which RotatE can capture, but which TransE cannot. More generally, generalised inversion rules can yield symmetry rules which TransE cannot capture, and thus only RotatE can capture generalised inversion.

Mutual exclusion. To capture mutual exclusion between relations r_1 and r_2, TransE (resp., RotatE) must satisfy $\mathcal{M}_1 \cap \mathcal{M}_2 = \varnothing$. In TransE, this holds iff $\|\boldsymbol{r_1} - \boldsymbol{r_2}\| \geq 2\epsilon$. Analogously, for RotatE, this holds if $|\boldsymbol{r_i} - \boldsymbol{r_j}| \geq \arcsin(2\epsilon)$ at every dimension and all node embeddings have a norm of at least 1. Such constructions can be set up for arbitrarily many mutual exclusion pairs, through decreasing ϵ or increasing the magnitude of embeddings. Thus, both TransE and RotatE can capture generalised mutual exclusions.

Composition. For TransE (resp., RotatE), two relations r_1 and r_2 compose a third relation r_3 iff $\boldsymbol{r_1} + \boldsymbol{r_2} = \boldsymbol{r_3}$ (resp., $\boldsymbol{r_1} \circ \boldsymbol{r_2} = \boldsymbol{r_3}$). On the other hand, both fail to capture generalised compositions. In particular, for the rules $r_1(x, y) \wedge r_2(y, z) \Rightarrow r_3(x, z)$ and $r_1(x, y) \wedge r_4(y, z) \Rightarrow r_3(x, z)$, both models force identical representations for r_2 and r_4 (In RotatE, the equivalence is modulo 2π). □

Lemma 3.4.3. *All the results given in Table 3.1 for bilinear models DistMult, ComplEx and TuckER hold.*

Proof. TuckER is shown to subsume DistMult and ComplEx [10], so all positive results for either ComplEx and DistMult automatically follow for TuckER. Hence, these positive results for TuckER are omitted from the presentation. Analogously, when negative results are shown for TuckER, they automatically propagate to DistMult and ComplEx.

We now recall the TuckER scoring function for a fact $r(e_1, e_2)$, given by:

$$\mathsf{score}(r(e_1, e_2)) = \mathcal{W} \cdot \boldsymbol{e}_1 \cdot \boldsymbol{r} \cdot \boldsymbol{e}_2,$$

where $\mathcal{W}$ is a shared third-order tensor. For simplicity, we define $\mathbf{v}_{r,1} = \mathcal{W} \times \boldsymbol{e_1} \times \boldsymbol{r}$, i.e., the left part of the scoring function. The scoring function is then written as:

$$\mathsf{score}(r(e_1, e_2)) = \mathbf{v}_{r,1} \cdot \boldsymbol{e}_2.$$

Given a head entity e_1 and a relation r, we also define the space

$$\mathbf{A}_{r,1} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \mathbf{v}_{r,1} \cdot \boldsymbol{x} \geq \epsilon\}.$$

Hierarchy. For bilinear models, it has been shown that individual hierarchies can be captured, but not generalised hierarchies [72]. In particular, to satisfy the rules $r_1(x, y) \Rightarrow r_3(x, y)$ and $r_2(x, y) \Rightarrow r_3(x, y)$ simultaneously, bilinear models must set either $r_1(x, y) \Rightarrow r_2(x, y)$ or $r_2(x, y) \Rightarrow r_1(x, y)$.

Intersection. We show that TuckER cannot capture intersections. In TuckER, a rule of the form:

$$r_1(x, y) \wedge r_2(x, y) \Rightarrow r_3(x, y)$$

holds iff $\mathbf{A}_{r_2,1} \cap \mathbf{A}_{r_1,1} \subset \mathbf{A}_{r_3,1}$, $\forall \boldsymbol{x} \in \mathbb{R}^d$. This is true iff $\mathbf{v}_{r_1,1}, \mathbf{v}_{r_2,1}, \mathbf{v}_{r_3,1}$ are colinear, and thus that r_1, r_2, and r_3 are colinear. However, this also implies that either $r_1(x, y) \Rightarrow r_2(x, y)$, or $r_2(x, y) \Rightarrow r_1(x, y)$. Hence, TuckER fails to capture intersections.

Symmetry. ComplEx captures symmetry patterns by having real-only embedding matrices for its relations. DistMult is inherently symmetric by construction. Since symmetry is a single-relation pattern, multiple symmetries can be independently captured, and thus all three models can capture generalised symmetry.

Anti-symmetry. DistMult cannot capture anti-symmetry, as it is inherently a symmetric model. ComplEx captures anti-symmetry by having imaginary-only embedding matrices for its relations. Analogously to symmetry, anti-symmetry is also a

single-relation pattern, and thus ComplEx (and TuckER) can capture generalised anti-symmetry.

Inversion. It is known that DistMult cannot capture inversions, while ComplEx can [159]. Generalised inversion can also be captured in ComplEx, as symmetry, the only other type of rule deducible from multiple inversions, is also captured by ComplEx.

Mutual exclusion. In TuckER, two relations r_1 and r_2 are mutually exclusive iff $r_1 = -r_2$. This implies TuckER can capture mutual exclusion, but cannot capture generalised mutual exclusions. In particular, to satisfy $r_1(x, y) \wedge r_2(x, y) \Rightarrow \bot$ and $r_1(x, y) \wedge r_3(x, y) \Rightarrow \bot$, TuckER forces $r_2 = r_3$.

Composition. It is shown that both ComplEx and DistMult cannot capture composition patterns [159, 72]. Furthermore, it is also known that relation maps must be bijective to be able to represent composition [159]. This is not the case in TuckER, as relations are surjective maps from $\mathbb{R}^{d \times d}$ to $\mathbb{R}^d$, and linear bijections between vector spaces are only possible with the same dimensionality. Hence, TuckER also cannot capture compositions. □

Generalised inference patterns are necessary to establish a more complete understanding of model inductive capacity, and, in this respect, our results show that BoxE goes well beyond any other model. However, generalised patterns alone are not sufficient. Indeed, different types of inference rules can appear *jointly* in practical applications, so KBC models must be able to jointly capture them. For instance, a relation can participate in a hierarchy pattern as well as a composition pattern, e.g., $\mathsf{parent}(x, y) \wedge \mathsf{brother}(y, z) \Rightarrow \mathsf{uncle}(x, z)$ and $\mathsf{brother}(x, y) \Rightarrow \mathsf{sibling}(x, y)$. As with generalised inference pattern, this scenario is not studied with existing models, and in fact these models also struggle with such combinations of inference patterns. For instance, RotatE can capture composition and generalised symmetry, but to jointly capture a single composition rule such as $\mathsf{cousins}(x, y) \wedge \mathsf{hasChild}(y, z) \Rightarrow \mathsf{relatives}(x, z)$, along with the symmetry pattern for relations $\mathsf{relatives}$ and $\mathsf{cousins}$, forces RotatE to make $\mathsf{hasChild}$ symmetric as well, i.e., $\mathsf{hasChild}(x, y) \Rightarrow \mathsf{hasChild}(y, x)$, which is clearly absurd. Therefore, we also evaluate model inductive capacity relative to more general *rule languages* [72]. We define a rule language as the *union* of different types of rules. Thus, generalised inference patterns are trivial rule languages allowing only one type of rule. BoxE can capture rules from a rich language, as stated next.

Theorem 3.4.3. *Let $\mathcal{L}$ be the rule language that is the union of inverse, symmetry, hierarchy, intersection, mutual exclusion, and anti-symmetry rules. BoxE can capture any finite set of consistent rules from the rule language $\mathcal{L}$.*

Proof. We show that a BoxE model of dimensionality $d = O(|\mathbf{R}|^2)$ captures $\mathcal{L}$. To this end, we leverage constructs from the generalised inference patterns proof, namely the boxicity argument and the box reconfiguration procedure.

Let S be a *consistent* set of rules, i.e., the rules in S do not yield a contradiction. Further, let S_p, S_a, and S_m be subsets of S, where S_p consists of hierarchy, symmetry, inversion, and intersection rules (the "positive" rules not including a negation operator), S_a consists of anti-symmetry rules, and S_m consists of mutual exclusion rules. We first show that rules from $S_p \cup S_a$ can be captured, then extend this to additionally capture S_m.

Step 1: Defining the relation intersection graph. We define a set of $2|\mathbf{R}|$ nodes, where every relation is encoded with 2 nodes for its head and tail boxes. We now constrain this graph to eventually capture all rules in S_p, analogously to the generalised intersection proof. First, we capture all symmetry and inversion rules as follows:

1. **Symmetry:** For every symmetry rule, we combine the corresponding head and tail nodes of a relation r into a single node. In other words, a single relation r is made symmetric by encoding both $\boldsymbol{r^{(1)}}$ and $\boldsymbol{r^{(2)}}$ with one same node. This encoding enforces that the head and tail boxes of r are identical, and thus that r is indeed symmetric, as required.

2. **Inversion:** For every inversion rule $r_1(x, y) \Rightarrow r_2(y, x)$, we combine the respective head and tail nodes of r_1 and r_2 such that $\boldsymbol{r_1^{(1)}}$ and $\boldsymbol{r_2^{(2)}}$, as well as $\boldsymbol{r_1^{(2)}}$ and $\boldsymbol{r_2^{(1)}}$, are each represented by one node. This makes that their corresponding boxes are equal, effectively capturing inversion patterns.

Following this step, G now consists of at most $2|\mathbf{R}|$ nodes, and captures symmetry and inversion rules jointly. It now remains to define edges in G, as needed to later capture intersection and hierarchy rules. This is done analogously to the proof for generalised intersections. First, the deductive closure of all intersection and hierarchy rules is computed, and the corresponding edges are encoded in G. Note that the resulting graph G continues to capture inversion and symmetry, as these rules are encoded through nodes, and also encodes the deductive closure of all rules in S_p. Indeed, any box intersection imposed by the deductive closure of intersection and hierarchy rules with a node capturing a symmetry or inversion rule automatically implies a box intersection with the multiple boxes that the node represents. Hence, G enables capturing symmetry and inversion rules a priori, and jointly sets up the necessary edges for hierarchy and inversion rules. Finally, we leverage the boxicity

argument, and our final graph G, to obtain a box configuration where all the box intersections needed to later capture hierarchy and intersection rules are present (but not necessarily capturing hierarchy and intersection patterns at this stage), and which also successfully captures inversion and symmetry rules.

Step 2: Anti-symmetry (S_a). Anti-symmetry rules are captured by adding additional dimensions to the box configuration resulting from Step 1 to distinguish between the head and tail boxes of an anti-symmetric relation. S is consistent, therefore only anti-symmetry rules not contradicting the set of rules $S_p \cup S_m$ can be given. For example, if symmetry rule $r(x, y) \Leftrightarrow r(y, x) \in S_p$, then $r(x, y) \Rightarrow \neg r(y, x) \notin S_a$. This is important, as it implies that no combination of hierarchy, inversion, intersection symmetry, and mutual exclusion rule (or their deductive closure) can force an intersection between $\boldsymbol{r^{(1)}}$ and $\boldsymbol{r^{(2)}}$, for any anti-symmetric r, and thus, that subsequent steps in this proof preserve the anti-symmetry captured in this step.

We now capture anti-symmetry rules by dedicating a new "disjointness" dimension for all boxes, such that, for an anti-symmetric relation r, the box ranges for head and tail boxes are made disjoint in this dimension, i.e., $[l^{(1)}, u^{(1)}] \cap [l^{(2)}, u^{(2)}] = \emptyset$. For all other relations, we set random ranges for this dimension such that, for all rules in S_p and its deductive closure, if an anti-symmetric r is the head of a hierarchy rule $r_1 \Rightarrow r$, then the ranges of $\boldsymbol{r_1}$ boxes in this dimension are also disjoint, i.e., anti-symmetric, and respect the hierarchy. Furthermore, for an intersection rule $r_1 \wedge r_2 \Rightarrow r$, then $\boldsymbol{r_1} \cap \boldsymbol{r_2} \subset \boldsymbol{r}$. In other words, we set initial relation values that are consistent with all known rules. This initialisation exists, as S is consistent, so cannot create conflicting interval requirements for relations in rules. Moreover, one can produce such an initialisation by considering a recursive pass through the rule sets affected by the anti-symmetric relations, through which all other uninitialised relations in the deductive closure can be accordingly set, with all other relations initialised completely at random.

Therefore, this new dimension captures $\boldsymbol{r^{(1)}} \cap \boldsymbol{r^{(2)}} = \emptyset$, so correctly captures anti-symmetry, and cannot be broken by subsequent rule-based box growth due to the consistency of S. It also is compatible with all symmetry and inversion rules, as box sharing is maintained. Hence, our current BoxE configuration captures any consistent set of anti-symmetry, symmetry, and inversion rules. Given S_a, at most $|S_a|$ additional dimensions are needed, and since at most $|\mathbf{R}|$ anti-symmetry rules can exist, the worst-case dimensionality of our configuration remains $O(|\mathbf{R}|\sqrt{\log(|\mathbf{R}|)})$. We now build on this result and show that the current configuration can be modified to additionally capture intersection and hierarchy rules.

Step 3: Hierarchies and intersections. Given the box configuration at the end of Step 2, we now apply the box reconfiguration procedure presented in the generalised intersections proof to capture all hierarchy and intersection rules in S. Since S is consistent, no hierarchy and intersection rules force any inconsistency with the already captured symmetry, anti-symmetry and inversion rules, e.g., if $r_1(x,y) \Rightarrow r_1(y,x), r_2(x,y) \Rightarrow \neg r_2(y,x) \in S$, then $r_1(x,y) \Rightarrow r_2(x,y) \notin S$. Thus all symmetry, anti-symmetry, and inversion patterns, whose capture is based on structural concepts (box sharing and dedicated dimensions respectively), are preserved. In particular, box sharing is unaffected, and no box growth from this step can break the disjointness of anti-symmetric relation boxes, as S is consistent.

Step 4: Mutual exclusion. Given the BoxE configuration from Step 3, capturing rules from $S_p \cup S_a$, we also capture rules from S_m with additional dimensions. Indeed, we show that this can be done using a BoxE configuration with $d = O(|\mathbf{R}|^2)$ dimensions. Starting from the configuration after the completion of Step 3, we now dedicate a single dimension per mutual exclusion rule, and capture this pattern as follows: For every mutual exclusion rule, we set a dimension, where r_1 and r_2 have disjoint range intervals $z_1, z_2 \subset [0,1]$, such that, without loss of generality, $z_1 = [z_{1,\min}, z_{1,\max}]$, $z_2 = [z_{2,\min}, z_{2,\max}]$ and $z_{2,\min} > z_{1,\max}$. Then, we set initial ranges for every other box in the configuration at this new dimension analogously to Step 2 (i.e., arbitrarily, but in a rule-aware fashion), and then repeat the box reconfiguration procedure in Step 3 for capturing hierarchy and intersection rules starting from the current configuration to maintain the enforcement of these rules.

Note that anti-symmetry, symmetry, and inversion rules play no part in this step, as anti-symmetry rules are captured with dedicated dimensions as shown earlier, whereas symmetry and inversion rules are already enforced, and thus captured, through box sharing and equality.

Intuitively, this step first makes r_1 and r_2 mutually exclusive in one dimension, then recursively traverses the set of hierarchy and intersection rules, as in Step 3, to preserve the capturing of these rules in this new dimension specifically. Clearly, anti-symmetry remains true, since its dedicated dimension is not affected by the repetition of Step 3. Furthermore, since S is consistent, all mutual exclusion rules in S can be captured without causing inconsistency. In other words, rule sets such as $r_1(x,y) \Rightarrow r_2(x,y), r_1(x,y) \Rightarrow r_3(x,y)$, and $r_2(x,y) \wedge r_3(x,y) \Rightarrow \bot$ are not possible.

Hence, since $|S_m| \leq 0.5|\mathbf{R}|(|\mathbf{R}| - 1)$, the number of distinct pairs that can be selected from $\mathbf{R}$, a BoxE model with $d = 0.5|\mathbf{R}|(|\mathbf{R}|-1)+|\mathbf{R}|+|\mathbf{R}|\sqrt{\log|\mathbf{R}|} = O(|\mathbf{R}|^2)$

dimensions can capture any consistent set of rules S from the language of intersection, hierarchy, symmetry, anti-symmetry, mutual exclusion, and inversion rules.

Remark. We finally highlight one subtle but important detail: Whereas the inference pattern language just described can be captured by a BoxE model having $d = O(|\mathbf{R}|^2)$ dimensions, some individual generalised patterns (inversion, hierarchy, symmetry, anti-symmetry, mutual exclusion) can be captured with even constant number of dimensions, and generalised intersection can be captured with $O(|\mathbf{R}|\sqrt{\log(|\mathbf{R}|)})$ dimensions. Hence, an interesting contrast in dimensionality requirements arises between capturing individual generalised inference patterns and capturing the rule language of Theorem 3.4.3, which highlights the significantly larger requirements that capturing joint generalised patterns, and the potential existence of cycles, can impose on any embedding model.

□

Observe that Theorem 3.4.3 captures generalised inference patterns for BoxE as a special case. Moreover, we note that a similar result is implausible for other KBC models, given their limitations in capturing generalised inference patterns, and we are unaware of any analogous result in KBC. The only related result is for ontology embeddings, and for quasi-chained rules [72], but this result merely offers region structures enabling capturing a set of rules, without providing any viable model or means of doing so.

3.4.3 Rule Injection

We now pose a complementary question to capturing inference patterns: Can a KBC model be injected with a *given* set of rules such that it provably enforces them, thereby improving its prediction performance? Formally, we say that a rule $\varphi \Rightarrow \psi$ (resp., $\psi \Leftrightarrow \varphi$) can be *injected* to a model, if the model can be configured to force ψ to hold whenever φ holds (resp., φ holds whenever ψ holds and vice versa).

There is a subtle difference between *capturing* and *injecting* an inference pattern. Indeed, rules with negation, such as mutual exclusion, can be easily captured with any disjointness between r_1 and r_2, but enforcing such a rule leads to non-determinism. To illustrate, r_1 and r_2 can be disjoint between their (i) head boxes, or (ii) tail boxes, or (iii) both, and at any combination of dimensions (for a d-dimensional embedding, there are $2^d - 1$ combinations). This non-determinism only becomes more intricate as interactions across different rules are considered.

In this subsection, we show that the positive fragment of the rule language that can be captured by BoxE, i.e., symmetry, inversion, hierarchy, and intersection, can also be injected. To our knowledge, this is a first such result for KGC. Indeed, existing KGC rule injection methods either cannot provably enforce rules or can only enforce a limited language. Concretely, rule injection is commonly approached using rule-based training loss [47, 143], which leverages fuzzy logic [71] and adversarial training [123], but cannot provably enforce rules. Furthermore, other approaches [49, 143] constrain embeddings explicitly, but only enforce very limited rules (e.g., inversion, linear implication). In fact, most popular standard KGC methods fail to capture simple sets of rules [72]. BoxE is thus a powerful model for rule injection in that it can explicitly and provably enforce a rich rule language and incorporate a strong bias by appropriately constraining the learning space. Our study is therefore related to the broader goal of making gradient-based optimization and learning compatible with reasoning [98]. We now formally state the rule injection result for BoxE and provide its proof.

Theorem 3.4.4. *Let $\mathcal{L}^+$ be the rule language that is the union of inverse, symmetry, hierarchy, and intersection rules. BoxE can be injected with any finite set of rules from the rule language $\mathcal{L}^+$.*

Proof. We start with a randomly initialised box configuration. First, we inject inversion and symmetry rules using box sharing: For symmetry rules, we set $\boldsymbol{r^{(1)}} = \boldsymbol{r^{(2)}}$, and for inversion rules, we set $\boldsymbol{r_1^{(1)}} = \boldsymbol{r_2^{(2)}}$ and $\boldsymbol{r_2^{(1)}} = \boldsymbol{r_1^{(2)}}$, and this can be done in linear time with respect to the number of inversion and symmetry rules. This achieves the same result as the node sharing in Step 1 of the proof of Theorem 3.4.3, except that the box configuration is a concrete random initialisation, as opposed to an abstract configuration known to exist due to boxicity. We then proceed with the box reconfiguration procedure in Step 3 of this same proof to enforce hierarchy and intersection rules on top of inversion and symmetry rules. This step is guaranteed to enforce these rules and their deductive closure, and maintains box sharing, so preserves symmetry and inversion.

We now analyse the worst-case runtime complexity of the box reconfiguration procedure. We assume the worst-case, that any pairwise intersections should be expressible, and thus use a dimensionality $d = O(|\mathbf{R}|\sqrt{\log(|\mathbf{R}|)})$. The worst-case running time of the box reconfiguration procedure for enforcing a single hierarchy/intersection rule is $O(|\mathbf{R}|d) = O(|\mathbf{R}|^2\sqrt{\log(|\mathbf{R}|)})$, corresponding to the maximum number of boundary changes needed per call. However, this upper bound is independent of

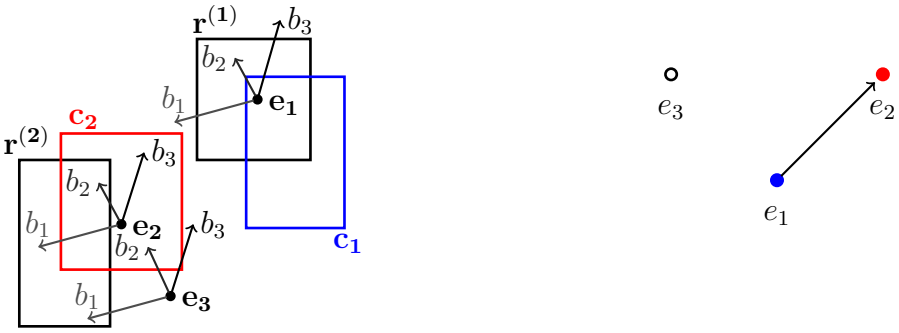


Figure 3.4: A sample BoxE model (left) for $d = 2$ with relations and classes representing the KG G (right). The binary relation r is shown using two black boxes $\mathbf{r^{(1)}}$ and $\mathbf{r^{(2)}}$, whereas classes c_1 and c_2 are depicted in red and blue, respectively.

the number of rules in S, as no more than $O(|\mathbf{R}|d)$ steps can be made across all rules. Thus, the worst-case running time for rule injection across all hierarchy and intersection rules is $O(|\mathbf{R}|d) = O(|\mathbf{R}|^2\sqrt{\log(|\mathbf{R}|)})$.

Hence, rule injection for hierarchy and intersection rules runs at worst in near-quadratic time with respect to $|\mathbf{R}|$, a typically small number, irrespective of the number of these rules. This result, combined with the efficiency of enforcing symmetry and hierarchy, imply that BoxE can be efficiently injected with arbitrary sets of symmetry, inversion, hierarchy and intersection rules. □

3.4.4 Representing Node Classes

As mentioned in Section 3.3, BoxE offers a natural representation for relations of arbitrary arity, which complements its inductive capacity. In particular, BoxE represents binary relations using 2 boxes per relation, and represents classes using a single box. Importantly, BoxE can represent both relations and classes *jointly*, and thus can use class information and naturally include it when modelling knowledge graphs/knowledge bases.

To illustrate, we consider a KG G with one binary relation r, and two classes c_1 and c_2. This KG contains two unary facts $c_1(e_1)$, $c_2(e_2)$ and one binary fact $r(e_1, e_2)$. A BoxE configuration in two dimensions representing this KG is shown on the left in Figure 3.4, with the graph representation of the KG shown on the right. In this figure, c_1 and c_2 are shown in blue and red respectively: In the BoxE configuration, they are represented by the boxes $\mathbf{c}_1$ and $\mathbf{c}_2$ respectively (superscript dropped for readability), and they are portrayed in the corresponding knowledge graph by the

blue and red node fill colours. Furthermore, r is depicted as a black arrow in the KG, and by two black boxes $\boldsymbol{r^{(1)}}$ and $\boldsymbol{r^{(2)}}$ in the BoxE space.

Observe that the unary facts $c_1(e_1)$ and $c_2(e_2)$ hold as the base embeddings $\boldsymbol{e_1}$ and $\boldsymbol{e_2}$ are contained in $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$, respectively, whereas e_3 does not belong to any class. Indeed, translational bumps, represented by gray-scale arrows in Figure 3.4, do not apply in the unary setting. Hence, class boxes define *absolute* spatial regions for base entity positions where class membership holds, and base entity embeddings, via their position, capture proximity to a class explicitly by means of their distance to the corresponding class box. In terms of the binary relation r, we see that only $r(e_1, e_2)$ holds, as $\boldsymbol{e_1} + \boldsymbol{b_2}$ lies in $\boldsymbol{r^{(1)}}$, and $\boldsymbol{e_2} + \boldsymbol{b_1}$ lies in $\boldsymbol{r^{(2)}}$, but this is not true for any other entity combination.

Overall, the interplay between base embeddings, class boxes, relation boxes, and translational bumps offers a holistic perspective of knowledge graph dynamics: Boxes define absolute regions in the embedding space, base position embeddings capture intrinsic entity information, and bumps encode interactions between entities which implicitly lead to new information and context. Hence, BoxE is highly interpretable, both in its inductive capacity and in its fact representation. Specifically, inference patterns can simply be "read" from the BoxE configuration, and this enables an informed understanding of what the model learns, and how it reaches its scores. Furthermore, the unified representation of facts offers a simple means of storing information, where representations for higher-arity facts are progressively built up based on fundamental base embeddings and translational bumps, and where all representations use identical semantics. By contrast, standard models jointly modelling binary relations and classes require distinct parametrisations and scoring functions [181, 36, 112]. For example, TransC [112] builds on TransE, and uses hyper-spheres to encode classes, but trains unary facts using a distinct objective function to the TransE objective for binary facts. This therefore further highlights the strong inductive capacity of BoxE, and its position as a unifying model for multi-arity knowledge base completion.

3.4.5 Runtime and Space Complexity

Runtime. For any fact $r(e_1, \ldots, e_n)$, we can compute the entity representations $\boldsymbol{e^{r(e_1,\ldots,e_n)}}$, in time $O(nd)$, by first computing $\sum_{1\leq i\leq n} \boldsymbol{b_i}$ in $O(nd)$, then subtracting $\boldsymbol{b_i}$ from the overall sum for every entity e and finally adding the base position $\boldsymbol{e}$, resulting in $3n$ d-dimensional addition/subtraction operations. The distance function dist runs in $O(d)$ for every box and entity, as it involves a fixed number of d-dimensional operations. Thus, running dist for all n positions yields a running time of $O(nd)$.

Table 3.2: Base statistics of benchmark datasets FB15k-237, WN18RR, YAGO3-10, JF17K, and FB-AUTO. In this table, $|\mathbf{E}|$, $|\mathbf{R}|$, and $|\mathbf{C}|$ denote the number of entities, relations and classes (for YAGO39K, to distinguish from binary relations).

Task	Dataset	$\|\mathbf{E}\|$	$\|\mathbf{R}\|$	$\|\mathbf{C}\|$	Training Facts	Validation Facts	Testing Facts
KGC	FB15k-237	14,541	237	N/A	272,115	17,535	20,466
	WN18RR	40,943	11	N/A	86,835	3,034	3,034
	YAGO3-10	123,182	37	N/A	1,079,040	5,000	5,000
KGC (Classes)	YAGO39K	39,374	39	46,110	792,832 (437,836 unary)	9,341	9,364
KGC (Rule Injection)	SportsNELL	4,252	11	N/A	163,742	81,454	81,454
KBC	JF17K	29,257	327	N/A	61,911	15,822	24,915
	FB-AUTO	3,388	8	N/A	6,778	2,255	2,180

Hence, BoxE scoring runs in $O(nd)$ overall. This implies that BoxE scales linearly with the arity of the relations in a KB, and thus can be applied to this setting with minimal computational overhead. Assuming that n is bounded, as is the case for knowledge graphs, BoxE runs in *linear time* with respect to dimensionality d.

Space complexity. In terms of space complexity, BoxE stores 2 d-dimensional vectors per entity e, namely its base position $\boldsymbol{e}$ and bump $\boldsymbol{b}$, and stores 2 d-dimensional vectors per box, denoting its lower and upper corners. Hence, for a KB with $|\mathbf{E}|$ entities and $|\mathbf{R}|$ relations with arity n, BoxE requires $(|\mathbf{E}| + n|\mathbf{R}|)d$ parameters.

3.5 Experimental Evaluation

In this section, we evaluate BoxE on a variety of tasks. First, we study its performance in the knowledge graph completion setting on standard benchmarks in the literature, namely FB15k-237, WN18RR, and YAGO3-10. We then perform a case study on YAGO3-10 to showcase the relational patterns BoxE learns and perform a robustness analysis to evaluate its performance with a tight computational budget. Afterwards, we consider a benchmark KG including classes and unary facts, YAGO39K, and study the performance of BoxE in this setting. For clarity, we denote the set of classes, i.e., unary relations, as $\mathbf{C}$, and separate these from the set of relations $\mathbf{R}$.

In addition to KGC, we also empirically evaluate BoxE on higher-arity KBs JF17K and FB-AUTO. Finally, we study the performance of BoxE with rule injection using a new dataset, SportsNELL, which includes a rich, dedicated set of rules. Across all experiments, we report state-of-the-art results, empirically confirming the theoretical

Table 3.3: KGC results (MR, MRR, Hits@10) for BoxE and competing approaches on FB15k-237, WN18RR, and YAGO3-10. Other approach results are best published, with sources cited per model.

Model	**FB15k-237**			**WN18RR**			**YAGO3-10**		
	MR	MRR	H@10	MR	MRR	H@10	MR	MRR	H@10
TransE(u) [145]	-	.313	.497	-	.228	.520	-	-	-
RotatE(u) [159]	185	.297	.480	*3254*	***.470***	***.564***	***1116***	*.459*	*.651*
BoxE(u)	**172**	**.318**	**.514**	**3117**	.442	.523	1164	**.567**	**.699**
TransE(a) [159]	170	.332	.531	3390	.223	.529	-	-	-
RotatE(a) [159]	177	**.338**	.533	3340	**.476**	**.571**	1767	.495	.670
BoxE(a)	**163**	.337	**.538**	**3207**	.451	.541	**1022**	**.560**	**.691**
DistMult [145, 185]	-	.343	.531	-	.452	.531	5926	.34	.54
ComplEx [145, 185]	-	.348	.536	-	**.475**	**.547**	6351	.36	.55
TuckER [10]	-	**.358**	**.544**	-	.470	.526	***4423***	***.529***	***.670***

strengths of BoxE. A list of all datasets used in this empirical evaluation, as well as their basic statistics, is provided in Table 3.2. Further details about all experimental setups and hyperparameter choices can be found in the appendix of this thesis.

3.5.1 Knowledge Graph Completion

In this experiment, we run BoxE on the KGC benchmarks FB15k-237, WN18RR, and YAGO3-10, and compare it with translational models TransE [21] and RotatE [159], both with uniform and self-adversarial negative sampling [159], and with bilinear models DistMult [185], ComplEx [166], and TuckER [10]. We train BoxE for up to 1000 epochs, with validation checkpoints every 100 epochs and the checkpoint with highest MRR used for testing. We report the best published results on every dataset for all models, and, when unavailable, report our best computed results in italic. All results are for models with $d \leq 1000$, to maintain comparison fairness [10]. We therefore exclude results by ComplEx [97] and DistMult [145] using $d \geq 2000$. The best results by category are presented in bold, and the best results overall are highlighted by a surrounding rectangle. "(u)" indicates uniform negative sampling, and "(a)" denotes self-adversarial sampling.

Results. For every dataset and model, MR, MRR, and Hits@10 are reported in Table 3.3. On FB15k-237, BoxE performs best among translational models, and is competitive with TuckER, especially in Hits@10. Furthermore, BoxE is comfortably state-of-the-art on YAGO3-10, significantly surpassing RotatE and TuckER. This result is especially encouraging considering that YAGO3-10 is the largest of all three

datasets, and involves a challenging combination of inference patterns, and many fact appearances per entity. Strong BoxE performance on FB15k-237, which contains several composition patterns, suggests that BoxE can perform well with compositions, despite not capturing them explicitly as an inference pattern.

On WN18RR, BoxE performs well in terms of MR, but is less competitive with RotatE in MRR. We investigated WN18RR more deeply, and identified two main factors for this. First, WN18RR primarily consists of hierarchical knowledge, which is logically flattened into deep tree-shaped compositions, such as $\mathsf{hypernym}(\mathsf{spoon}, \mathsf{utensil})$. Second, symmetry is prevalent in WN18RR, e.g., $\mathsf{derivationally\_related\_form}$ accounts for 29,715 ($\sim$34.5%) of WN18RR facts, which, combined with compositions, also helps RotatE. Indeed, in RotatE, the composition of two symmetric relations is (incorrectly) symmetric, but this is useful for WN18RR, where 4 of the the 11 relations are symmetric. That is, the modelling limitations of RotatE become an advantage on WN18RR, and enable it to achieve state-of-the-art performance on this dataset.

Overall, BoxE is competitive on all benchmarks, and is state of the art on YAGO3-10. Hence, it is a strong model for KGC on large, real-world KGs.

3.5.2 Analysing BoxE Representations on YAGO3-10

To better understand the performance of BoxE on YAGO3-10, we perform two additional studies. First, we observe the learned BoxE configuration on this dataset and study the patterns appearing therein. Second, we study the robustness of BoxE, to confirm the efficiency and compactness of BoxE representations, and its ability to maintain its state-of-the-art performance even with strict computational restrictions.

Learned YAGO3-10 boxes. YAGO3-10 contains 37 relations, with highly variable sets of properties. For instance, it contains the relation $\mathsf{isMarriedTo}$, which is symmetric, and largely one-to-one, but also contains relations like $\mathsf{wasBornIn}$ and $\mathsf{worksAt}$, which are many-to-one. To quantify relation dynamics, we therefore compute the *volume* of every relation box for all 37 YAGO3-10 relations from a BoxE model trained analogously to the earlier experiment using uniform negative sampling. More specifically, we compute the geometric mean of interval sizes for a given box in the learned configuration across all dimensions, such that relative box mean volumes offer insight as to the underlying relation semantics. All 37 relations, as well as their geometric mean box volumes, are shown in Table 3.4[1].

[1]Note that boxes in BoxE are mapped to the space $[-1, 1]^d$ using the hyperbolic tangent function, so the geometric mean volume is upper-bounded by 2.

From this table, we can make the following four observations. First, more popular relations, in terms of entities they connect, tend to be represented with larger boxes in the embedding space. This confirms our intuition that boxes effectively define entity classes, and thus larger classes are embedded with larger boxes in the latent space. For example, the sparse relation hasWebsite has very small boxes of mean volume about 0.15, as it is only makes up 68 facts in the YAGO training dataset. By contrast, the relation created has both boxes with mean volume above 0.9, and appears in over 1,400 facts.

Second, we observe that the size of relation boxes also correlates with implicit entity types, in addition to relation popularity. Indeed, the relation playsFor, despite appearing over 300,000 times, only has mean box volumes 0.284 and 0.469 respectively, whereas isLeaderOf, with less than 1,000 facts, has a tail box of mean volume exceeding 1. This is due to the diversity in entity types appearing at these relations: For playsFor, head entities are athletes, which cluster together in a smaller region of the embedding space, and tail entities are football/sports clubs, which are more diverse, but still quite similar semantically. By contrast, head entities for isLeaderOf are individuals, with medium variability, but tail entities can be anything from very different countries (e.g., Mali,

Table 3.4: Geometric mean volume per dimension for head and tail relation boxes in YAGO3-10 following training.

Relation	**Head**	**Tail**
actedIn	0.456	0.479
created	0.966	0.905
dealsWith	0.373	0.366
diedIn	0.383	0.480
directed	0.474	0.461
edited	0.461	0.441
exports	0.238	0.260
graduatedFrom	0.608	0.526
happenedIn	0.453	0.363
hasAcademicAdvisor	0.655	0.605
hasCapital	0.390	0.347
hasChild	0.299	0.761
hasCurrency	0.228	0.239
hasGender	0.669	0.688
hasMusicalRole	0.328	0.427
hasNeighbor	0.311	0.312
hasOfficialLanguage	0.213	0.255
hasWebsite	0.159	0.143
hasWonPrize	0.264	0.381
imports	0.249	0.241
influences	0.510	0.567
isAffiliatedTo	0.257	0.557
isCitizenOf	0.544	0.614
isConnectedTo	0.403	0.388
isInterestedIn	0.644	0.496
isKnownFor	0.632	0.623
isLeaderOf	0.446	1.005
isLocatedIn	0.496	0.547
isMarriedTo	0.923	0.924
isPoliticianOf	0.361	0.521
livesIn	0.536	0.341
owns	0.907	0.485
participatedIn	0.389	0.471
playsFor	0.284	0.469
wasBornIn	0.465	0.445
worksAt	0.498	0.488
wroteMusicFor	0.450	0.646

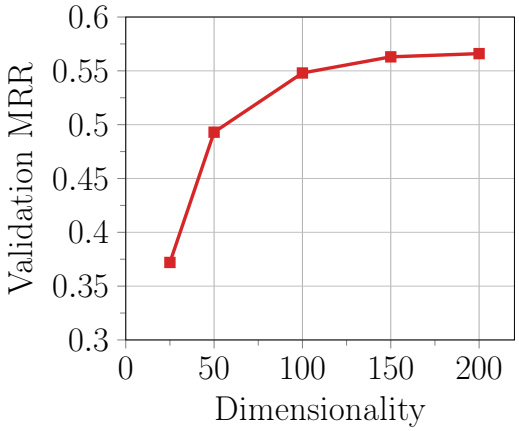


Figure 3.5: BoxE validation performance over YAGO3-10 versus dimensionality.

Kuwait) to cities, districts, and towns (e.g., Toronto, Oxnard (California)), to political parties and associations (e.g., Democratic Governors Association, Hungarian Communist Party), which are vastly different types of entities, and this results in an extremely large tail box for isLeaderOf.

Third, we observe that relative box sizes accurately reflect the type of their underlying relation. More specifically, larger tail boxes tend to denote one-to-many relations, larger head boxes indicate a many-to-one relation, and similar sizes indicate many-to-many or one-to-one relations. This is especially evident for the one-to-many relations hasChild (0.299 vs 0.761), and isAffiliatedTo (0.257 vs 0.557), and for many-to-one relations isInterestedIn (0.644 vs 0.496), and graduatedFrom (0.608 vs 0.526).

Finally, symmetric relations in YAGO3-10, namely hasNeighbor and isMarriedTo, are represented with near-identically sized boxes. This is a very important finding, as it indicates that BoxE successfully captures the symmetry inference pattern, for which a necessary condition is having identical head and tail boxes.

All in all, these results further highlight the interpretability of BoxE, in terms of capturing inference patterns, accurately inferring and portraying entity classes, and inferring and modelling relation types.

Robustness analysis. In this experiment, we evaluate the dependence of BoxE on dimensionality d on YAGO3-10 to understand its performance in a computationally restricted setting. We train BoxE with uniform negative sampling on YAGO3-10 using $d = \{25, 50, 100, 150, 200\}$. Relative to the best setup in Section 3.5.1, we only tune the margin. That is, we fix the learning rate, batch size, and number of negative samples analogously to the setup used in Section 3.5.1. At each dimensionality, we

Table 3.5: KGC results on YAGO39K for BoxE and competing model TransC.

Model	YAGO39K				
	MR	MRR	H@1	H@3	H@10
TransC	-	.421	.283	.500	.692
BoxE(u)	**326**	**.542**	**.412**	**.632**	**.782**

report peak MRR recorded over the validation set. The optimal margins in this analysis were $\gamma = 6$ for $d = 25$ and $\gamma = 10.5$ otherwise.

A plot of validation MRR versus dimensionality is drawn in Figure 3.5. From this plot, we can observe that BoxE maintains very strong performance, even at $d = 50$, rivaling that of state-of-the-art translational model RotatE, even with just uniform negative sampling. Furthermore, it performs at near-optimal level with $d = 100$, and is already state-of-the-art on YAGO3-10 at this small dimensionality. Hence, BoxE proves to be very robust for performing knowledge base completion with restricted computational power.

3.5.3 Knowledge Graph Completion with Classes

Building on the standard KGC setting, we train BoxE on a benchmark including a diverse set of unary facts and classes to quantify the value and importance of its unified fact semantics and representations. To this end, we use the YAGO39K dataset proposed in the TransC paper [112], and compare with TransC. More concretely, we use the binary facts in YAGO39K as normal, and map its instanceOf facts into unary facts. However, we do not use its subClassOf facts, as these specify a large yet sparse set of class hierarchies which BoxE cannot train to softly enforce[2]. As earlier, we train BoxE on this data using uniform negative sampling. Furthermore, we use a dimensionality of $d = 100$ in keeping with the used dimensionality for TransC, to maintain fairness of comparison. Finally, we use an analogous experimental setup to tune our model.

We report results for BoxE and TransC on YAGO39K in Table 3.5. Here, we can see that BoxE substantially outperforms TransC by an MRR margin of over 0.1. This is all the more significant considering that TransC additionally is given access to

[2]Alternatively, BoxE can be injected with these hierarchies a priori so as to provably enforce them, but we opt against this (i) to maintain fairness of comparison against TransC, and (ii) given the sparsity and limited information presented in these rules. Indeed, most rules simply map a *small fraction* of classes to super-classes, and these hierarchies define mutually exclusive tree-shaped structures, with limited interactions between them.

Table 3.6: KBC results on JF17K and FB-AUTO for BoxE and competing models.

Model	JF17K					FB-AUTO				
	MR	MRR	H@1	H@3	H@10	MR	MRR	H@1	H@3	H@10
m-TransH	-	.446	.357	.495	.614	-	.728	0.727	.728	.728
m-DistMult	-	.460	.367	.510	.635	-	.784	.745	.815	.845
m-CP	-	.392	.303	.441	.560	-	.752	.704	.785	.837
HypE	-	.492	.409	.533	.650	-	.804	.774	.823	.856
HSimplE	-	.472	.375	.523	.649	-	.798	.766	.821	.855
BoxE(u)	**363**	.553	.467	.596	.711	**110**	.837	.804	.858	.895
BoxE(a)	372	**.560**	**.472**	**.604**	**.722**	122	**.844**	**.814**	**.863**	**.898**

class hierarchy information (the subClassOf relation between classes), whereas BoxE is solely trained on unary and binary facts. These results align with our theoretical expectations. First, BoxE uses a unified semantics for unary and binary facts, which allows for a better overall training procedure relative to that of TransC, where unary facts are fit using hyperspheres, and binary facts are fit using the TransE objective. Second, BoxE has a strong expressiveness advantage over TransC. Indeed, BoxE is fully expressive, whereas TransC is not, as it is built on TransE. Hence, these results highlight the strength of BoxE as a means to naturally leverage class information towards improved knowledge graph completion.

3.5.4 Higher-arity Knowledge Base Completion

In this experiment, we evaluate BoxE on datasets with *higher arity* facts, namely the publicly available JF-17K and FB-AUTO. These datasets contain facts with arities up to 6 and 5, respectively, and include facts with *different arities*. We then compare BoxE with the best-known reported results over the same datasets [57]. For this experiment, we set $d = 200$, for fairness with other models, and perform hyperparameter tuning analogously to Section 3.5.1. We report MR, MRR, Hits@1, Hits@3, and Hits@10 for all evaluated models in Table 3.6.

On both datasets, BoxE comfortably achieves state-of-the-art performance, surpassing the next best models by 0.06 and 0.03 in terms of MRR on JF17K and FB-AUTO, respectively. This improvement can be traced back to the unified BoxE semantics, which naturally extend to this application setting, and can easily scale to facts with non-uniform and higher arity. Indeed, the BoxE semantics compartmentalise the scoring of facts based on arity positions, such that each arity position contributes to the final score, and such that translational bumps enable interactions

across these positions to maximise flexibility. Hence, different-arity facts in BoxE fundamentally share entity position embeddings and translational bumps, but use them *differently*, based on the given fact, e.g. a higher-arity fact applies more bumps than a binary fact. This is turn makes that training BoxE simultaneously on different-arity relations results in substantial parameter sharing. Moreover, BoxE uses an identical underlying mechanism, point-to-box distance, to score facts across all different arities. By contrast, all other models learn identical relation representations, even across different arities. As a result, these models suffer from a bottleneck in their learning process which is especially prominent for benchmarks with non-uniform arity relations.

In addition to its semantics, BoxE also provides an advantage in terms of inductive capacity in the higher-arity setting. Concretely, the inductive capacity of BoxE also naturally extends to higher-arity facts by design, as the same box configurations studied in Section 3.4 can be studied in higher-arity simply by considering patterns across more arity positions. For instance, a single higher-arity hierarchy rule $r(e_1, \ldots, e_n) \Rightarrow s(e_1, \ldots, e_n)$ can be captured via the subsumption of r boxes in s boxes across all arity positions, and analogous observations can be made for single higher-arity intersection, mutual exclusion, and inversion. This inductive capacity further improves the learning ability of BoxE in this setting, and offers a strong inductive bias for the knowledge base completion task.

3.5.5 Rule Injection

In this experiment, we investigate the impact of rule injection on BoxE performance. To this end, we propose the SportsNELL dataset, a subset of NELL [124] with a rich known ontology, shown in Figure 3.6a. The training facts in SportsNELL are directly extracted from NELL, whereas the validation and testing facts are a mixture of known facts and logically deduced facts from the training set and the SportsNELL ontology. More specifically, we use the SportsNELL ontology and training facts to define a larger set of facts, which is precisely the logical closure of the SportsNELL dataset w.r.t. the given ontology, i.e., the completion of SportsNELL under the ontology rules.
Dataset construction. Initially, we start with 181,936 facts extracted from NELL, which span 11 sports relations and 4,252 sports-related entities. These facts are extracted such that all selected entities appear 50 or more times in NELL across the 11 relations. This set of facts is then used to define the SportsNELL training set, by considering a random subset of 90%, i.e., 163,742 facts. Based on this training set and the SportsNELL ontology, the *logical closure* is then computed., i.e., ontology

Table 3.7: Rule injection experiment results on the SportsNELL full and filtered evaluation sets.

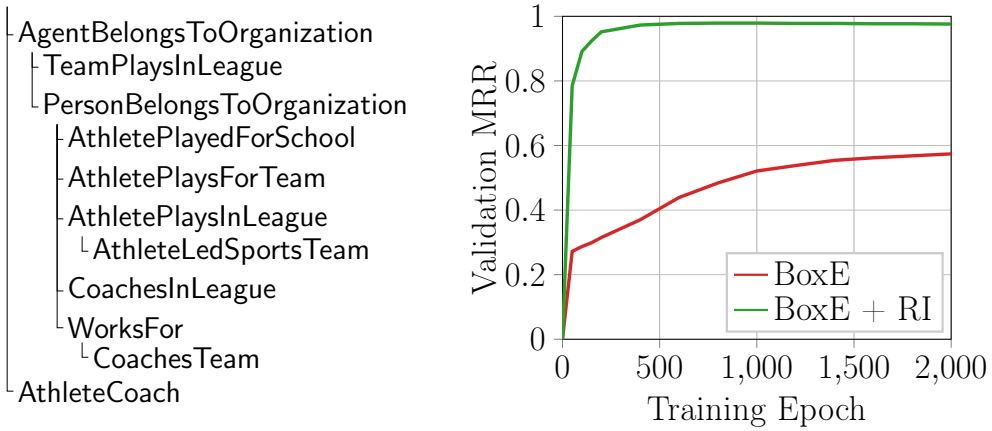
Model	Full Set			Filtered Set		
	MR	MRR	H@10	MR	MRR	H@10
BoxE	17.4	.577	.780	19.1	.713	.824
BoxE+RI	**1.74**	**.979**	**.997**	**5.11**	**.954**	**.984**

rules are repeatedly applied to deduce new facts until no new facts can be deduced. These facts are then subsequently used for evaluation within SportsNELL. Observe that these new facts in the logical closure are direct results of rule application, and thus their correct prediction indicates a good capturing of the underlying ontology. All in all, the computation of logical closure yields 144,714 new facts. These logically deduced facts are combined with the remaining 10% of the NELL-extracted facts, and split evenly across validation and testing sets to yield a size of 81,454 facts for both sets. Hence, SportsNELL contains a total of 326,650 facts.

Experimental setup. We compare plain BoxE, i.e., the model used so far in this experimental evaluation, against BoxE injected with the SportsNELL ontology, denoted BoxE+RI. We train both models for 2000 epochs on the SportsNELL training set, and evaluate both models on its validation and test facts, which we refer to as the *full evaluation set*, to measure the effect of rule injection. Second, we evaluate both models on a subset of these evaluation sets, such that only facts that are *not* directly deducible via the ontology from the training set, i.e., eliminating all inferences that can be made by a rule-based approach alone, are used. This setting, which we call the *filtered evaluation setting*, thus carefully tests the impact of rule injection on model inductive capacity.

Results. The results on both evaluation datasets are shown in Table 3.7, and the learning curves for both model variations relative to MRR are shown in Figure 3.6b. On the full evaluation dataset, BoxE+RI performs significantly better (~ 0.4 MRR difference) than BoxE, and reaches its optimal performance in far fewer epochs. This shows that rule injection clearly improves the performance, training, and convergence of BoxE. Indeed, BoxE+RI converges to a near-perfect peak performance within 500 epochs, and mostly stabilises following this point, whereas standard BoxE does not fully converge, even after the whole 2000 epochs have elapsed, and fails to surpass 0.6 MRR.

Importantly, we see that BoxE+RI maintains a very significant advantage relative to BoxE even in the filtered evaluation setting, where logical deduction and rule



(a) The SportsNELL ontology.

(b) BoxE and BoxE+RI learning curves.

Figure 3.6: Ontology (left) and learning curves (right) for the rule injection experiments.

enforcement is not a direct means for high-quality prediction. Therefore, rule injection also indirectly improves BoxE by making its embeddings and boxes more *structured*, which improves its inductive bias, as all predictions necessarily conform with the given set of rules. Intuitively, predictions in BoxE+RI get amplified with the help of the injected rules, and this is highly desirable for real-world application, as many real-world KBs have an associated schema, or a simple ontology. Concretely, this capability of BoxE entails that inductive capacity, in the shape of the rule language captured by BoxE, can be complemented with strong *deductive capacity*, in the shape of explicit rule injection.

While allowing to amplify predictions in our study, rule injection can potentially lead to poor performance with existing metrics. Indeed, if a model mostly predicts wrong facts, these would lead to further wrong conclusions due to rule application. To illustrate, consider a simple hierarchy rule $r(x, y) \Rightarrow s(x, y)$. If a model with this rule injected can accurately predict a vast majority of r facts, then it will automatically, and correctly predict their corresponding s facts. Conversely, a poor performance in r predictions can be *exacerbated* by rule injection, as wrong r facts likely yield yet more incorrect s facts. Hence, a low-quality prediction model can find its performance further hindered by rule injection, as false predictions create yet more false positives, thereby lowering the rank of any good predictions in evaluation. Therefore, rule injection must be complemented with models having good inductive capacity (for sparser and simpler datasets) and expressiveness (for more complex and rich datasets), such that they yield high-quality predictions in all data settings [165].

3.6 Summary and Outlook

In this chapter, we presented BoxE, a spatio-translational model for KBC that represents entities using base positions and translational bumps in a latent embedding space, and encodes relations as sets of hyper-rectangles, i.e., boxes. We showed that BoxE is fully expressive and can capture a rich rule language of inference patterns. We also showed that BoxE, by design, can naturally scale to higher-arity facts and fit unary facts and classes. Finally, we showed that BoxE can also be injected with a rich rule language, so as to provably enforce a set of positive rules prior to training.

Empirically, we studied the performance of BoxE on knowledge graph completion, including scenarios with rule injection and with class information, and also conducted experiments on higher-arity knowledge base completion. Across these experiments, BoxE achieved state-of-the-art performance, and validated its theoretical strengths. Indeed, BoxE successfully learned representations for facts of arbitrary arity, and managed to capture key relational properties during training. Moreover, it substantially improved as a result of rule injection, and even improved its performance beyond the direct application of the enforced rules.

Beyond our study, an interesting direction for future research is to explore novel applications to the box embedding paradigm beyond simple link prediction, particularly multi-hop reasoning. This is a natural extension to BoxE, as the model already can represent higher-arity facts. Hence, an interesting question is how to extend these box representations to consider *queries* over knowledge bases, and appropriately account for query variable *quantification*. Another interesting direction is to consider additional information within knowledge graphs. For instance, we have already extended BoxE to temporal knowledge graphs in concurrent work [122], with state-of-the-art results, and believe that BoxE representations are highly promising for settings with rich domain knowledge, e.g., bio-medical knowledge graphs, as well as multi-modal knowledge graphs with entity features.

Overall, BoxE presents a strong theoretical backbone for KBC, combining theoretical expressiveness with strong inductive capacity and promising empirical performance. We hope that this work leads to a more holistic approach to KBC model development, basing itself equally on underlying logical concepts and on empirical findings, so as to deliver more comprehensively sound models for this challenging task.

Chapter 4

Node Classification Meets Link Prediction on Knowledge Graphs

4.1 Introduction

Within graph machine learning, a main task is *node classification* [73], in which nodes in the input graph are mapped to one of finite many target classes. Node classification is typically studied in two settings, namely the *transductive* and *inductive* settings, respectively. In the transductive setting, all nodes and their features, along with incomplete node labels, are assumed to be accessible during training, and the goal is to predict the missing node labels. By contrast, the inductive setting requires models to make predictions on nodes that are completely unseen during training. In this chapter, we focus on the transductive setting, and thus consider a single, fixed input graph with incomplete node labels.

Broadly speaking, message passing neural networks (MPNNs) [62, 169, 92] have become the *de facto* standard models for transductive node classification, and have been applied to multiple distinct types of graph-structured data, including heterogeneous, multi-relational graphs [189]. However, MPNNs introduce an implicit relational completeness assumption as part of their computation. More specifically, MPNNs perform message passing over known edges, and make no account for potentially missing edges. Hence, message passing fails to consider latent connections that potentially could be inferred from the input graph.

In this chapter, we inspect this relational completeness assumption more closely, and argue that it is unrealistic and potentially detrimental for node classification across multiple application domains, particularly heterogeneous, multi-relational graphs. Indeed, such graphs are widely known to be incomplete, with prominent graphs such as Freebase [19] missing several important links, e.g., nationality information for a

majority of its person entities [180]. Given this context, we propose the more holistic problem of node classification over *incomplete* graphs. This problem is more difficult than standard node classification, as it requires models to additionally factor in the incompleteness of the underlying graph when making class predictions, so as to best benefit from sound representation and completion of relational edges. Observe that node classification over incomplete graphs naturally connects the node classification task with *link prediction*, which is the task of inferring missing edges from existing knowledge an input knowledge graph [21, 10].

By considering node classification with respect to an incomplete knowledge graph, we automatically consider an interesting dual problem, namely the link prediction task with respect to a heterogeneous, feature-enhanced, knowledge graph with classes. Hence, considering relational incompleteness offers a unifying perspective that naturally combines node classification with link prediction, and studies the interplay between the two tasks. By contrast, current literature and models only study node classification or link prediction in isolation.

The rest of this chapter is organised as follows: We formally define and state the two problems in our unified perspective in Section 4.2. We then present a unified framework, MLP-X, to jointly address both problems using standard shallow embedding models in Section 4.3, and provide some example models. To establish a rigorous evaluation setup for our unified setting, we propose a novel, carefully designed benchmark, WikiAlumni, in Section 4.4, and present its main properties. We conduct an empirical analysis of MLP-X models on WikiAlumni and other graph representation learning benchmarks in Section 4.5, for both node classification with incompleteness and link prediction with classes and features. Finally, we conclude this chapter in Section 4.6.

To our knowledge, this work is the first benchmarking study that combines node classification and link prediction on knowledge graphs and addresses both problems simultaneously in a unified setup. Hence, this study paves the way for future research across the whole spectrum of node embedding models ranging from shallow to deep embedding models.

4.2 Problem Formulation

Notation. Throughout this chapter, we consider a relational vocabulary consisting of finite sets $\mathbf{E}$, $\mathbf{C}$, $\mathbf{R}$ of *entities*, *classes*, and *binary relations* respectively. As in the experimental section of Chapter 3, the separation between classes and binary

relations, which is equivalent to the unified definition in Chapter 2, is primarily for notational convenience and clarity. Within this vocabulary, a *unary fact* is of the form $c(e)$, where $c \in \mathbf{C}$ is a class, i.e., a unary relation, and $e \in \mathbf{E}$ is an entity. As before, a binary fact is of the form $r(e_h, e_t)$, where $r \in \mathbf{R}$ is a binary relation and $e_h, e_t \in \mathbf{E}$ are entities. Throughout this chapter, we refer to e_h as the *head* entity, and e_t as the *tail* entity. A *knowledge graph (KG)* G is therefore a finite set of *facts* defined over our relational vocabulary, which, in our setting, includes both unary and binary facts. Note, however, that this is not standard in most common knowledge graph benchmarks, as unary facts are usually not included. Nonetheless, we consider this more general formulation, as it better suits our unifying perspective.

In addition to the relational vocabulary, we consider *node features*, such that for each entity $e \in \mathbf{E}$, there exists a d-dimensional *feature vector* $x \in \mathbb{R}^d$ describing the entity that is available to the model. For simplicity, we denote the set of all node features as a matrix $\mathbf{X} \in \mathbb{R}^{|\mathbf{E}| \times k}$. Hence, knowledge graphs in our setting can be viewed as a *labelled* graph, where nodes have features and correspond to entities in $\mathbf{E}$, node labels correspond to classes in $\mathbf{C}$, and edge labels correspond to relations in $\mathbf{R}$.

Problems. In this chapter, we focus on the following problems over KGs:

1. Given a relationally incomplete KG with features, where only a subset of entities are labelled with class information, we are interested in *transductive node classification*, and thus wish to make class predictions for the entities whose class is not already known. In particular, we would like to exploit existing links and use potentially inferred links to improve node classification quality. In the transductive setting, all nodes in the KG are seen during training.

2. Analogously to the node classification task, we consider the special case of *entity classification*, where the input graph has no features [150], similarly to standard knowledge graph completion baselines (e.g., FB15k-237, YAGO3-10, cf. Chapter 3).

3. We also investigate link prediction in a more general sense, in that we consider KGs with classes and features, allowing a link prediction model to be informed by features towards learning better representations leading to higher quality predictions.

4.3 Embedding Models for Node Classification and Link Prediction

In this section, we propose the MLP-X framework, which extends shallow embedding models to leverage node features, jointly train on transductive node classification and link prediction, and subsequently solve both problems in a unified fashion. Fundamentally, MLP-X introduces a multi-layer perceptron (MLP) to process features in isolation for every node, and then combines all MLP outputs with learned node embeddings from the original embedding model by means of a weighted summation. As MLP-X models are intended to jointly tackle link prediction and node classification, we adapt the MLP-X training procedure to better align with both tasks. In particular, we consider an adapted negative sampling mechanism which exploits mutual exclusion between node classes and negatively samples classes, so as to better match class structures and yield a better inductive bias.

Feature processing. In a standard shallow embedding model, every KG entity $e_i \in \mathbf{E}$ is represented with a d-dimensional vector $\boldsymbol{e_i} \in S^d$ in a latent space S, e.g., the d-dimensional real space $\mathbb{R}^d$ or the complex space $\mathbb{C}^d$. This entity representation is learned by training with the model scoring function to make existing facts in the knowledge graph rank highly, and negatively sampled facts rank poorly. This training is usually feature-agnostic, as standard KGC benchmarks do not include node features.

As a first step towards studying node classification and link prediction in a unified setting, we therefore propose a simple feature processing mechanism, based on an MLP $f : \mathbb{R}^k \to S^d$. At a high level, this MLP maps input node features from a k-dimensional input domain into the latent embedding space of the original shallow node embedding model. However, these representations are not used in isolation to capture nodes, but instead are *combined* with existing node embeddings using a weighted summation. More formally, for each entity e_i, the feature-aware representation in MLP-X is defined as:

$$\boldsymbol{e'_i} = \lambda \boldsymbol{e_i} + f(\boldsymbol{x_i}), \tag{4.1}$$

where $\lambda \in \mathbb{R}^+$ is a tunable hyper-parameter and $\boldsymbol{x_i} \in \mathbb{R}^k$ is the feature vector of e_i. Note that some models, such as BoxE, define multiple representations per entity. In this case, we introduce as many MLPs as there are representations and parallelise our framework and Equation (4.1) accordingly.

By combining feature representation (via MLPs) and structure encoding (via embeddings), MLP-X preserves the expressive power of the underlying model while introducing a feature-level inductive bias. In turn, this feature bias is *structure-agnostic*, as the MLP simply processes every node feature independently of the overall graph structure. Hence, MLP-X avoids assumptions such as relational completeness, and disentangles entity representations into separate feature components and structure-based entity embeddings.

Representing classes as binary facts. As several prominent node embedding models in the literature, e.g., TransE, RotatE, ComplEx, fail to capture classes, i.e., unary facts, MLP-X performs a simple *binarisation*, where each unary fact $c(e_i)$ is viewed as a binary fact $r_c(e_i, \gamma_c)$, where r_c is a binary relation, and γ_c is an auxiliary entity, both of which are specific to the class c. For all classes, auxiliary entity and relation representations are learnable, so as to offer maximum flexibility and maintain model expressiveness. In particular, learning class-specific representations, unlike, e.g., using a unified auxiliary variable, allows MLP-X to enforce parameter sharing across all unary facts for a given class, while using overall node representations to train all class representations jointly.

Class-aware negative sampling. Node embedding models typically perform negative sampling on the entity level. That is, given a binary fact $r(e_h, e_t)$, negative sampling randomly selects a position between head and tail and replaces the corresponding entity with a random distinct entity e', yielding facts such as $r(e', e_t)$ and $r(e_h, e')$. This negative sampling procedure is plausible in the binary setting, but is somewhat problematic for binarised unary facts. To see this, consider a fact $r_c(e_i, \gamma_c)$. With standard negative sampling, negatively sampled facts will have the non-sensical form $r_c(e_i, e')$, with probability 0.5. Moreover, simply ignoring tail replacement and exclusively sampling facts of the form $r_c(e', \gamma_c)$ is not well-aligned with class semantics, as classes are usually (i) mutually exclusive and (ii) small in number. Hence, sampling facts of the $r_c(e', \gamma_c)$ is likely to have a high false negative rate.

To address these issues, we build on the assumption that classes are *mutually exclusive*, a common assumption in practice, as classes denote distinct types, e.g., person, institution, lecture, etc, and propose *class-aware* negative sampling. More concretely, we alter the *class* for an entity, rather than selecting an alternate entity for the class. Hence, our class-aware negative sampling, for a binarised true fact $r_c(e_i, \gamma_c)$, yields negative facts of the form $r_{c'}(e_i, \gamma_{c'})$, where $c' \neq c$. Given mutual exclusion between classes, this negative sampling is guaranteed to produce true negative facts, unlike standard negative sampling. Moreover, class-aware negative sampling allows

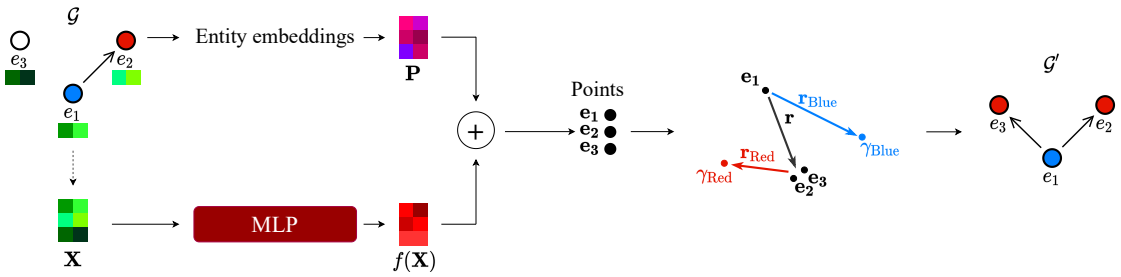


Figure 4.1: Given a KG $\mathcal{G} = \{Blue(e_1), Red(e_2), r(e_1, e_2)\}$ over entities e_1, e_2, e_3, MLP-TransE processes node features $\mathbf{X}$ using an MLP f, and separately instantiates point embeddings $\mathbf{P}$ for every entity. Then, these embeddings are summed with their corresponding MLP outputs to obtain representations in the TransE space, which, with the help of binarisation, then predicts the graph $\mathcal{G}' = \mathcal{G} \cup \{Red(e_3), r(e_1, e_3)\}$.

shallow embedding models to make similar modelling assumptions as MPNNs, which exploit mutual exclusion at the prediction level by means of, e.g., a softmax function.

4.3.1 Model Instantiations

In this subsection, we present model-specific MLP-X instantiations of TransE, RotatE, and BoxE. Detailed explanations of the base models TransE and RotatE can be found in Chapter 2, whereas BoxE explanations can be found in Chapter 3.

MLP-TransE. MLP-TransE extends the translational model TransE [21] with feature processing via MLPs, and combines with model embeddings using a weighted summation, as in Equation (4.1). As TransE does not support classes, we use binarisation to describe unary facts within MLP-TransE. An illustration for MLP-TransE is shown in Figure 4.1.

At a conceptual level, we note that MLP-TransE inherits the model properties of TransE. In particular, MLP-TransE is also not fully expressive, and fails to capture simple inference patterns such as symmetry, but captures, e.g., single relational composition rules. Moreover, MLP-TransE extends TransE into the unary setting through binarisation. However, this binarisation, coupled with the inherent limitations of TransE, produces potentially negative ramifications for MLP-TransE in terms node classification performance.

To illustrate this limitation, we consider the following example:

Example 4.3.1. *Consider the set of facts*

$$c(a_1), s(a_1, a_2), c(a_2), s(a_2, a_3),$$

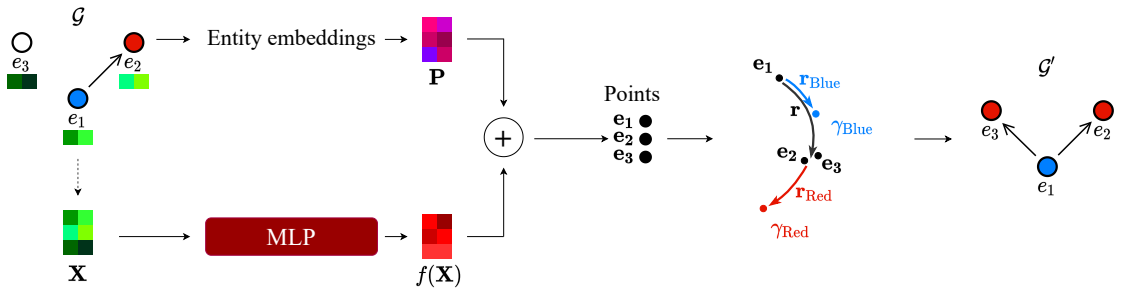


Figure 4.2: Given a KG $\mathcal{G} = \{Blue(e_1), Red(e_2), r(e_1, e_2)\}$ over entities e_1, e_2, e_3, MLP-RotatE processes node features $\mathbf{X}$ using an MLP f, and separately instantiates point embeddings $\mathbf{P}$ for every entity. Then, these embeddings are summed with their corresponding MLP outputs to obtain representations in the RotatE space, which, with the help of binarization, then predicts the graph $\mathcal{G}' = \mathcal{G} \cup \{Red(e_3), r(e_1, e_3)\}$.

defined on the domain $\{a_1, a_2, a_3\}$ *with a single unary predicate* c *and a single binary predicate* s*. To represent* c *in TransE, we must define a corresponding domain element* γ_c *and binary predicate* r_c*. This yields the corresponding facts:*

$$r_c(a_1, \gamma_c), s(a_1, a_2), r_c(a_2, \gamma_c), s(a_2, a_3).$$

If TransE perfectly fits these facts, then the following equations hold:

$$\boldsymbol{a_1} + \boldsymbol{r_c} = \boldsymbol{\gamma_c},\ \boldsymbol{a_1} + \boldsymbol{s} = \boldsymbol{a_2},\ \boldsymbol{a_2} + \boldsymbol{r_c} = \boldsymbol{\gamma_c},\ \text{and}\ \boldsymbol{a_2} + \boldsymbol{s} = \boldsymbol{a_3}.$$

In this scenario, the first and third equations falsely imply that $\boldsymbol{a_1} = \boldsymbol{a_2}$*, and this behaviour stems from the limitation of TransE with regards to one-to-many relations. However, this behaviour is caused more easily as a result of class representations, and may create undesirable side-effects on the representations of binary facts. In fact, applying* $\boldsymbol{a_1} = \boldsymbol{a_2}$ *to the second equality implies* $\boldsymbol{s} = 0$*! Moreover, applying* $\boldsymbol{s} = 0$ *to the fourth equality implies* $\boldsymbol{a_2} = \boldsymbol{a_3}$*, and thus* $c(a_3)$ *(falsely) holds. Hence, the aforementioned set of facts, if fit perfectly, results in a highly degenerate configuration in the TransE latent space that leads to erroneous fact predictions.*

MLP-RotatE. Analogously to MLP-TransE, we apply an MLP on node features and combine with existing entity embeddings using a weighted summation following Equation (4.1). Note, however, that MLP outputs must be in a complex space, and thus we produce an output in $\mathbb{R}^{2d}$ and map this to the complex plane. As with MLP-TransE, we use binarisation to support classes. Finally, the same ramifications apply as in MLP-TransE. Indeed, the limited expressiveness of RotatE also affects node classification, with the same example provided for MLP-TransE also holding for MLP-RotatE. An illustration for MLP-RotatE is shown in Figure 4.2.

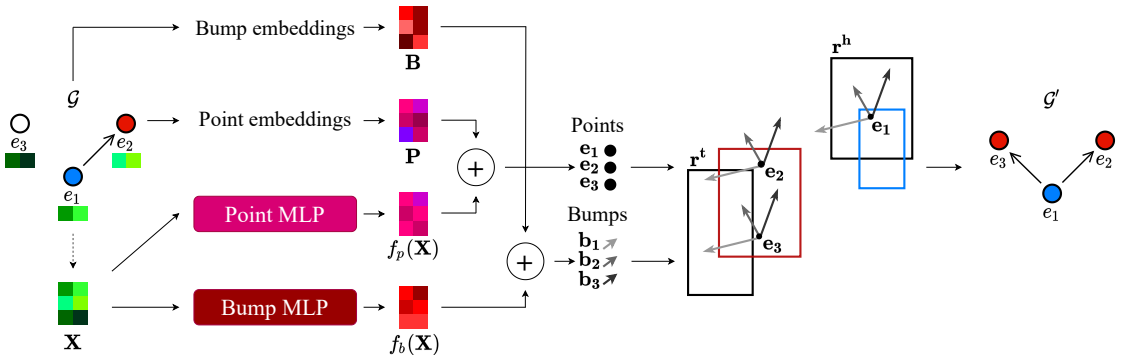


Figure 4.3: Given a KG $\mathcal{G} = \{Blue(e_1), Red(e_2), r(e_1, e_2)\}$ over entities e_1, e_2, e_3, MLP-BoxE processes node features $\mathbf{X}$ using two (point and bump) MLPs f_p and f_b, and separately instantiates point ($\mathbf{P}$) and bump ($\mathbf{B}$) embeddings for every entity. Then, these embeddings are summed with their corresponding MLP outputs, yielding representations in the BoxE space which are used to predict the graph $\mathcal{G}' = \mathcal{G} \cup \{Red(e_3), r(e_1, e_3)\}$.

MLP-BoxE. In BoxE (cf. Chapter 3), every entity $e_i \in \mathbf{E}$ is represented by two vectors, a *base position* vector $\boldsymbol{e_i} \in \mathbb{R}^d$, and a *translational bump* vector $\boldsymbol{b_i} \in \mathbb{R}^d$, which translates all the entities co-occurring in a fact with e_i. Therefore, we must apply Equation (4.1) on both entity base positions and translational bumps using separate MLPs. More formally:

$$\boldsymbol{e_i'} = \lambda \boldsymbol{e_i} + f_p(\boldsymbol{x_i}), \qquad \boldsymbol{b_i'} = \lambda \boldsymbol{b_i} + f_b(\boldsymbol{x_i}),$$

where f_p and f_b are two distinct MLPs for position and bump representations, respectively. Unlike TransE and RotatE, BoxE naturally represents classes $c \in \mathbf{C}$ using a d-dimensional box $\boldsymbol{c}$. Hence, binarisation is not needed within MLP-BoxE. However, MLP-BoxE still employs class-aware negative sampling to provide a better inductive bias. The overall structure of MLP-BoxE is shown in Figure 4.3.

As BoxE is fully expressive, MLP-BoxE is also fully expressive given a sufficient dimensionality d. This is shown in the following corollary, which extends on the original BoxE expressiveness proof in Theorem 3.4.1.

Corollary 4.3.1. *Consider a relational vocabulary, consisting of finite sets* $\mathbf{E}$ *of entities,* $\mathbf{C}$ *of classes, and* $\mathbf{R}$ *of relations. Let* $\mathcal{S}$ *be the set of all facts over this vocabulary, where every entity has a matching feature from a feature matrix* $\mathbf{X} \in \mathbb{R}^{|\mathbf{E}| \times k}$*. Then, for any set* $\mathcal{T} \subseteq \mathcal{S}$ *of true and any disjoint set* $\mathcal{F} \subseteq \mathcal{S}$ *of false facts, there exists a MLP-BoxE model* M *with dimensionality* $d = |\mathbf{E}||\mathbf{R}| + |\mathbf{C}|$ *that maps all facts of* $\mathcal{T}$ *to true and all facts of* $\mathcal{F}$ *to false.*

Proof. This is a straightforward extension of Theorem 3.4.1. BoxE can represent a knowledge graph over $|\mathbf{E}| = n$ entities and $|\mathbf{R}|$ binary relations using a dimensionality $n|\mathbf{R}|$. Therefore, by considering classes as binary relations, one can directly apply the theorem to obtain a dimensionality bound of $n(|\mathbf{R}| + |\mathbf{C}|)$. However, this is sub-optimal, and thus we propose a tighter bound by adapting the theorem proof.

We start by constructing a $n|\mathbf{R}|$-dimensional BoxE configuration capturing all binary facts, as in Theorem 3.4.1, and introduce $|\mathbf{C}|$ new dimensions to capture all class memberships. More specifically, let $\boldsymbol{e_m}$ and $\boldsymbol{b_m}$ denote position and bump embeddings for every entity e_m in BoxE, and, for every binary relation r_j, let $\boldsymbol{r}_j{}^{(1)}$ and $\boldsymbol{r}_j{}^{(2)}$ denote their head and tail boxes, and let $\boldsymbol{l}_j^{(1)}, \boldsymbol{l}_j^{(2)}$, and $\boldsymbol{u}_j^{(1)}, \boldsymbol{u}_j^{(2)}$ indicate the lower and upper corners of both boxes respectively. Finally, we denote the k^{th} dimension value of a vector $\boldsymbol{v}$ by $v(k)$ for convenience.

Given the initial BoxE configuration capturing binary facts, we now additionally define the class boxes and enforce class memberships as follows:

1. For every class $c_i \in \mathbf{C}$, we define a unique box $\boldsymbol{c}_i$, such that for $k \in \{1, ..., n|\mathbf{R}|\}$, its lower corner is set as $\boldsymbol{l}_i(k) = \min_m \boldsymbol{e}_m(k) - \epsilon$, $\epsilon \in \mathbb{R}, \epsilon > 0$, whereas its upper corner is defined as $\boldsymbol{u}_i^{(1)}(k) = \max_m \boldsymbol{e}_m(k) + \epsilon$. Intuitively, the class boxes fit all points in the first $n|\mathbf{R}|$ dimensions, and thus hold true in these dimensions for all entities.

2. For $i \in \{1, ..., |\mathbf{C}|\}$, we then define additional dimensions, such that, for every class c_i with box $\mathbf{c_i}$, bounded by $\mathbf{l}_i$ and $\mathbf{u_i}$ respectively, we set $\boldsymbol{l}_i(n|\mathbf{R}| + i) = -1$ and $\boldsymbol{u}_i(n|\mathbf{R}|+j) = 1$. This defines a new dimension per class, i.e., $|C|$ dimensions in total, such that each class c_i has a unique dimension where its box initially spans the interval $[-1, 1]$. The remaining dimension ranges for index $k \neq i$ are defined in Step 5.

3. For $m \in \{1, ..., n\}$, and $i \in \{1, ..., |\mathbf{C}|\}$,

   (a) If $c_i(e_m) \in \mathcal{T}$, we set $\boldsymbol{e}_m(n|\mathbf{R}| + i) = 0$, and thus place this base position embedding inside $\boldsymbol{c}_i$.

   (b) Otherwise, if $c_i(e_m) \in \mathcal{F}$, we set $\boldsymbol{e}_m(n|\mathbf{R}| + i) = 2$, outside $\boldsymbol{c}_i$.

4. Bump vectors are inconsequential for fitting unary classes, and thus we set added bump dimensions to 0, i.e., $\forall i \in \{1, ..., |\mathbf{C}|\}, \boldsymbol{b}_m(n|\mathbf{R}| + i) = 0$.

5. Finally, we ensure that the new class dimensions do not render previously true binary facts false, by setting the relation boxes in these $|\mathbf{C}|$ dimensions to include all possible class values. More concretely, for $i \in \{1, ..., |\mathbf{C}|\}$, for $j \in \{1, ..., |\mathbf{R}|\}$:

   (a) $\boldsymbol{l}_j^h(n|\mathbf{R}| + i) = -3$, $\boldsymbol{l}_j^t(n|\mathbf{R}| + i) = -3$,

   (b) $\boldsymbol{u}_j^h(n|\mathbf{R}| + i) = 3$, $\boldsymbol{u}_j^t(n|\mathbf{R}| + i) = 3$.

   Hence, all relation boxes cover the interval $[-3, 3]$ in the added dimensions, and thus include all entity representations in those dimensions by construction.

Therefore, BoxE admits a parametrisation $\mathcal{W}$ with $d = n|\mathbf{R}| + |\mathbf{C}|$ that can represent all possible knowledge graphs over $\mathbf{C}$ and $\mathbf{R}$. This naturally extends to MLP-BoxE. In particular, given a BoxE parametrisation $\mathcal{W}$ and MLPs f_p and f_b, we can exactly reconstruct $\mathcal{W}$ using MLP-BoxE, simply by setting its embedding vectors for every entity e_m as:

$$\bar{\boldsymbol{e}}_m = \boldsymbol{e}'_{m,\mathcal{W}} - f_p(\mathbf{x}_m), \text{and}$$
$$\bar{\boldsymbol{b}}_m = \boldsymbol{b}'_{m,\mathcal{W}} - f_b(\mathbf{x}_m),$$

where the $\mathcal{W}$ subscript indicates that this vector originates from $\mathcal{W}$. □

4.3.2 Model Properties

Expressive power. MLP-X maintains the expressive power of its base models: If X is fully expressive, then MLP-X is also fully expressive. Furthermore, any set of facts not captured by X cannot be reliably captured using MLP-X, as indistinguishable facts can simply be assigned identical features, which would also not be distinguishable by an MLP. Therefore, the MLP in MLP-X does not contribute in terms of expressive power, but instead plays a role more akin to regularisation. Indeed, this MLP learns a simple mapping from features to the latent embedding space that could act as a useful "starting point", which individual node embeddings can later refine. As representations in MLP-X are in the same space as X, and since MLPs are universal approximators [43], they can map features to arbitrary values in the latent space, which offers useful flexibility to complement the learned model embeddings.

Relational inductive capacity. As MLP-X fundamentally maintains the semantics of its underlying shallow node embedding models, it maintains their relational inductive bias. In particular, MLP-X learns analogous embeddings for entities, maintains standard relation embeddings, and trains using the same scoring function. Therefore, MLP-X models can still capture structural information and relational properties,

namely inference patterns such as relational symmetry, hierarchies, via their learned embedding components. The only tangible difference between shallow node embedding models and MLP-X in this regard is that feature representations of input entities directly influence latent representations in the embedding space, and thus can lead to new (or different) inference patterns being captured.

MLP-X vs. MPNNs. Compared with MPNNs, MLP-X models are not restricted to known neighbours when computing node representations, as latent edges are still implicitly considered through the scoring function. This remains the case even with feature processing, as MLPs do not introduce additional modelling assumptions, with their output exclusively computed from node features. By contrast, MPNNs explicitly rely on the input graph structure, and process features by passing messages exclusively over known edges. Hence, these models are more vulnerable if the input graph structure is incomplete or noisy.

Beyond modelling assumptions, MLP-X offers a unifying perspective of link prediction and node classification over an incomplete input graph. More concretely, MLP-X proposes scores both for classes and binary facts from a *unified* training run on all relational data. However, MPNNs in the literature are usually only trained to perform one of node classification and link prediction at a time. In particular, MPNNs for node classification are trained without considering potentially incomplete edges. Furthermore, MPNNs used for link prediction [130, 150, 162] optimise exclusively for the link prediction objective. Indeed, a node classification MPNN and a link prediction MPNN use wholly different loss and scoring functions, e.g., softmax and DistMult [185] scoring respectively, and thus train in incompatible ways.

MLP-X vs. shallow node embedding models. MLP-X builds on shallow node embedding models and supplements them with feature processing via an MLP. Moreover, MLP-X includes useful inductive biases on node classes, namely better negative sampling that emulates the role of softmax in MPNNs, and represents unary and binary facts in a unified fashion through binarisation. Though the latter point is simple, it is especially important, as most existing models currently operate exclusively on binary facts, with no attribution to unary facts. In fact, only few models propose natural unary fact representations: BoxE (cf. Chapter 3) can explicitly handle class information by viewing classes as boxes in the space, and TransC [112] extends TransE with hyper-spheres to explicitly represent classes. However, TransC has limited expressive power, and is not competitive with state-of-the-art models.

All in all, MLP-X enables a joint view of node classification and link prediction that addresses limitations across both MPNNs and shallow node embedding models.

4.4 WikiAlumni: A Knowledge Graph with Node Features

In the literature, node classification models have primarily been evaluated on citation networks such as Cora, Citeseer, and PubMed [153, 154], whereas link prediction models have been evaluated using KG benchmarks such as FB15k-237 [164] and WN18RR [48]. These benchmarks are not suitable for our unified perspective for multiple reasons. One one hand, citation networks typically only include a single relation cites, indicating a paper citing another. Therefore, these benchmarks have limited relational type information. Furthermore, these networks, by nature, are quite complete, as paper co-citations are regularly and automatically logged. Hence, citation networks are not representative datasets for studying relational incompleteness. On the other hand, link prediction benchmarks are usually extracted from large-scale knowledge bases, and thus share their rich relational structure. However, these benchmarks do not incorporate node classes or features.

Recently, new multi-relational node classification benchmarks, also based on citation networks, such as OGBN-MAG [78], have been introduced. OGBN-MAG introduces four edge types: cites, isAffiliatedTo, author, and hasTopic. However, these relation types provide limited relational information, as they follow directly from the classes of nodes they connect. Indeed, for OGBN-MAG, entities have four distinct types, paper, institution, fieldOfStudy and person, and relations follow from these types. For example, a cites edge necessarily connects two paper nodes, and hasTopic edges connect papers to academic fields. Therefore, these benchmarks, though potentially including millions of entities, also have limited relational information. Thus, neither link prediction nor node classification currently presents a holistic benchmark allowing a joint study of node classification with relational incompleteness, as well as link prediction with classes and node features.

In light of this landscape, we introduce a new dataset, WikiAlumni, based on a subset of the YAGO4 [161] English Wikipedia knowledge graph. In WikiAlumni, the node classification goal is to predict the *alma mater* of person entities. Classes are produced using the alumniOf relation, and are made mutually exclusive by only considering entities with a *unique* alma mater.

Knowledge graph extraction. To construct WikiAlumni, we extract a sub-graph from the English Wikipedia YAGO4 knowledge graph. We do this by first selecting challenging target classes, i.e., alma maters, and subsequently using these to extract relevant binary relations.

To make WikiAlumni challenging, we select target classes to be both (i) balanced and (ii) non-trivially deducible from node features and other sub-graph relations. To achieve class balance, we restrict the set of eligible classes to those appearing 700 or more times in the tail of alumniOf in YAGO4. This ensures that classes offer sufficient and evenly distributed training data. Furthermore, we achieve a non-trivial separation between classes and existing relations and features in WikiAlumni by exclusively considering North American institutions as target classes, as (i) North American institutions are minimimally homophilic, i.e., the alma mater of an entity cannot be inferred by considering neighbouring nodes, and (ii) these institutions are minimally correlated with other relations in YAGO4. In particular, correlations between nationality and alma mater are far less pronounced for North American institutions than they are for, e.g. French/European institutions, where leading French universities are primarily attended by French nationals.

All in all, our selection criteria yield a final set of 24 classes for WikiAlumni, which ultimately dictate the final structure of this dataset. Given these 24 classes, we then proceed to extract the WikiAlumni knowledge graph from YAGO4 as follows:

1. For each class, we select the 700 nodes with the highest degrees in YAGO4 (ties broken arbitrarily). This step yields a total of 16,800 target classification nodes.

2. We add all nodes within the direct neighbourhood of the 16,800 nodes, excluding literals, e.g., code strings, latitude, longitude, etc. Furthermore, we exclude the alumniOf relation and *all* nodes appearing in its tail, even those beyond our target classes, to avoid simplifying the task by considering institution similarities.

3. Finally, we exclude sparse relations appearing in less than 10 facts.

Overall, this extraction procedure yields a knowledge graph with 52,714 nodes and 121,857 edges spanning 29 relations. As all direct node neighbourhoods are considered, the full YAGO4 relational schema is considered for selection, and hence the extracted knowledge graph includes a rich set of entities and relations. In particular, this knowledge graph extends well beyond the 16,800 person nodes and additionally includes other entity types originating from their neighbourhoods, such as locations, countries, languages, and creative works, in keeping with the YAGO4 ontology.

In terms of relations, WikiAlumni presents a diverse and rich vocabulary, which provides significant information about nodes, e.g., knowsLanguage, birthPlace, nationality

Table 4.1: Base statistics of WikiAlumni. Subsets WikiAlumni-90% and WikiAlumni-80% only include 109,652 and 97,468 edges, respectively.

Dataset	Nodes	Edges	Classes	Relations	Training Labels	Validation Labels	Testing Labels
WikiAlumni	52678	121836	24	29	10080	5040	1680

and hasOccupation. This diverse set of entity types and relations provides ample information to describe the 16,800 target nodes, and, importantly, is sufficiently distinct from the classification target to make alma mater prediction challenging.

Feature computation. Given the extracted knowledge graph, we now produce node features to complete the benchmark. To this end, we scrape English Wikipedia to obtain the introduction text for all selected entities. When this page does not exist, which is the case of around 0.2% of nodes, we instead use its WikiData summary. Using the scraped text, we compute feature vectors as follows:

1. We tokenise the text into words.
2. We map words to 300-dimensional GloVe embeddings [136]. Words without an embedding in GloVe are discarded, and
3. We compute the mean of all word embedding vectors as the node feature vector.

Note that this feature generation not does produce features for all nodes, as some entities do not have high-quality descriptions, or do not include words with GloVe embeddings. Therefore, we discard all entities without a description or GloVe embedding, and this yields the final WikiAlumni dataset consisting of 52,678 nodes and 121,836 edges. Finally, given this fully-featured dataset, we select and fix a split, where 60% of target nodes are used for training, 30% are used for validation, and 10% are used for testing. Overall, key WikiAlumni statistics are shown in Table 4.1, and a breakdown of relations and classes in WikiAlumni is provided in Table 4.2.

Incomplete subsets. For consistent empirical evaluation of node classification and link prediction, we provide two carefully crafted fixed splits over edges, WikiAlumni-90% and WikiAlumni-80%, where we omit 10% and 20% of existing edges from the training set, respectively. These splits ensure that no further nodes are disconnected, i.e., are isolated, relative to WikiAlumni. We achieve this as follows:

1. We compute a random "spine" of nodes, in which all nodes preserve at least one edge. This "spine" is unaffected by edge removal.

Table 4.2: The classes and relations of WikiAlumni.

Classes		**Relations**	
Class Name	Appearances	Relation Name	Appearances
BrownUniversity	700	about	21
ColumbiaUniversity	700	actor	15752
CornellUniversity	700	affiliation	27
HarvardUniversity	700	author	3360
MassachusettsInstituteofTechnology	700	award	8031
MichiganStateUniversity	700	birthPlace	14086
NorthwesternUniversity	700	character	24
NewYorkUniversity	700	children	2028
OhioStateUniversity	700	competitor	75
PrincetonUniversity	700	composer	155
SyracuseUniversity	700	contributor	11
UnitedStatesMilitaryAcademy	700	creator	873
UnitedStatesNavalAcademy	700	deathPlace	7143
UniversityofCaliforniaBerkeley	700	director	3761
UniversityofCaliforniaLosAngeles	700	editor	159
UniversityofChicago	700	founder	482
UniversityofMichigan	700	gender	18
UniversityofPennsylvania	700	hasOccupation	20572
UniversityofTexasatAustin	700	homeLocation	1187
UniversityofToronto	700	knowsLanguage	4847
UniversityofVirginia	700	lyricist	111
UniversityofWashington	700	memberOf	12337
UniversityofWisconsin − Madison	700	musicBy	1794
YaleUniversity	700	nationality	16382
		parent	2021
		producer	4267
		publisher	51
		spouse	2242
		worksFor	19

2. We randomly sample a corresponding percentage of all non-"spine" nodes and drop these from the graph, so as to yield an overall total number of edges (including "spine" edges) equal to 90% (resp., 80%) of all initial edges.

Maintaining edges for existing nodes ensures a maximal relational contribution to the node classification task, and eliminates the possibility of producing an artificial reliance on pure feature-based computation. In other words, this splitting protocol allows all models (including MPNNs) to exploit edge information where it exists in WikiAlumni, even in its incomplete subsets.

Dataset properties. WikiAlumni provides features and relational structure to support node classification, such that each of these components makes distinct contri-

butions to the prediction task. Furthermore, the relational structure in WikiAlumni is highly non-trivial, rich, minimally correlates with node features and the target classes, which themselves are selected to minimise homophily.

Fundamentally, the 24 mutually exclusive WikiAlumni classes and 29 relations each introduce distinct semantics, and appear with varying prominence in the data. More specifically, classes and relations in WikiAlumni are not semantically intertwined, and thus a class cannot be predicted by detecting a given edge type, and vice-versa. For example, knowing the alma mater of an entity does not inform about the existence of any relation, e.g. actor, and no relation gives away a specific alma mater. Hence, WikiAlumni avoids the interdependence between classes and relations found in current citation network benchmarks [78]. Moreover, the binary relations in WikiAlumni are highly distinct among themselves and do not present any simplifying structures. In particular, simple redundancies, such as excessive relational symmetries and inverse relations, do not arise in the relational schema of WikiAlumni. This aligns well with recently proposed refinements to standard link prediction datasets [48, 164], where such redundancies are removed. Hence, the relations in WikiAlumni are challenging to predict from a link prediction perspective, and WikiAlumni is well-suited as a benchmark in this application domain.

4.5 Experimental Evaluation

In this section, we use WikiAlumni to evaluate MLP-X models, as well as standard MPNNs and baseline shallow node embedding models in our unified setting. In particular, we train these models on WikiAlumni, which presents both class and relational incompleteness, and consider two evaluation dimensions: node classification performance with edge incompleteness, and link prediction performance with node features and classes. Furthermore, we report additional experiments on standard citation networks, and conduct an ablation study on the features of WikiAlumni, to explore the effect of feature quality of node classification performance.

4.5.1 Node Classification with Incomplete Edges

Experimental setup. We train MLP-TransE, MLP-RotatE, and MLP-BoxE on WikiAlumni and its incomplete subsets WikiAlumni-90% and WikiAlumni-80%. More specifically, we jointly train on all facts (both classes and standard binary facts) for each dataset, and evaluate node classification accuracy on the validation set. For more reliable results, we repeat this experiment 5 times and report the average across

all runs. We then observe and compare performance across the three WikiAlumni versions, so as to appraise how each model reacts to increased relational incompleteness as we transition from standard WikiAlumni to WikiAlumni-90%, and then to WikiAlumni-80%. Across all MLP-X models, we set all feature processing MLPs to have two hidden layers of size 1000, each using the ReLU activation function.

Parallel to the standard node classification setting, we additionally replicate these experiments in the entity classification setting, where we discard node features. For entity classification, we therefore discard MLP feature processing and use the original embedding models TransE, RotatE, and BoxE, only keeping the refined negative sampling of MLP-X. Across both node classification and entity classification experiments, we train MLP-X models (i.e., shallow node embedding models with class-aware negative sampling) using both negative sampling (NS) and cross-entropy (CE) loss with 100 negative facts per positive fact, and present results for both losses. We then compare our results against the following baselines:

- **Label Propagation (LP)** [191]: Label Propagation (LP) is an iterative algorithm that exclusively make predictions for target node labels by propagating partial node label information (from training nodes) across edges, irrespective of edge type, i.e., LP treats graphs as single-relational. LP is agnostic to node features, and thus only propagates information based on (i) the existence or absence of an edge, (ii) the existence of target class information.

- **Multi-Layer Perceptron (MLP)**: An MLP with two hidden layers of size 512. This MLP trains exclusively on node features, and has no access to edge information, i.e., each node class prediction is made independently from other node predictions and features. This MLP provides a baseline for what performance levels can be achieved solely using node features, and ignoring the graph structure.

- **rGCN** [150]: A 2-layer relational GCN that aggregates neighbourhood features based on edge type, i.e., aggregation is done on a relational basis, with relation-specific aggregates later combined to compute representation updates. We use rGCN as a baseline for heterogeneous GNNs, and apply its basis regulariser with 10 basis vectors, as this was the best-performing configuration in our experiments.

- **GAT** [169]: A 2-layer graph attention network (GAT), which uses attention to aggregate neighbourhood features. GAT does not explicitly use edge types, i.e., it treats graphs as single-relational, but performs strongly in practice on heterogeneous graphs, often outperforming heterogeneous GNN models [189].

Table 4.3: Node and entity classification results (validation accuracy) for MLP-X models, using both NS and CE loss, and benchmark models on WikiAlumni and its incomplete splits. The top three results are reported in bold, with opacity increasing as rank improves. For greater visibility, we drop standard deviations from this table, as these quantities are small throughout the study, not exceeding 0.2%.

Node Classification				**Entity Classification**			
Model	80%	90%	100%	Model	80%	90%	100%
LP	28.0	30.2	32.2	LP	28.0	30.2	32.2
MLP	35.5	35.5	35.5	MLP	-	-	-
rGCN	39.0	39.7	**42.4**	rGCN$^+$	24.8	27.1	30.5
GAT	**40.0**	**41.2**	**42.4**	GAT$^+$	22.4	24.8	26.0
MLP-TransE(NS)	38.4	39.2	39.8	TransE(NS)	**29.8**	**32.2**	**34.0**
MLP-RotatE(NS)	37.2	38.5	39.1	RotatE(NS)	**29.8**	**32.0**	**33.7**
MLP-BoxE(NS)	38.8	39.3	40.4	BoxE(NS)	**29.3**	**31.3**	**33.4**
MLP-TransE(CE)	**40.3**	**41.0**	41.9	TransE(CE)	25.5	27.8	30.1
MLP-RotatE(CE)	**40.0**	40.7	41.4	RotatE(CE)	24.5	26.6	29.5
MLP-BoxE(CE)	**40.5**	**41.4**	**42.1**	BoxE(CE)	26.5	29.3	31.0

Note that these baselines exclusively address node and entity classification, and do not jointly return edge scores for link prediction. Furthermore, these baselines include MPNNs, which are state-of-the-art models for node classification. However, MPNNs employ a relational completeness assumption, and thus are unable to recover missing edges. Hence, comparing between MPNNs and MLP-X will shed light on the effect of joint link prediction towards node classification under relational incompleteness.

Finally, for fairness, we use a dimensionality of $d = 128$ across all models, and supplement MPNNs with node embeddings for entity classification, resulting in the models rGCN$^+$ and GAT$^+$, respectively. Further details about hyper-parameter setup (learning rate, computational resources, optimal loss margins, etc.) and experimental protocol for all experiments can be found in the appendix of this thesis.

Results. Node and entity classification accuracy results on WikiAlumni, as well as its subsets WikiAlumni-90%, and WikiAlumni-80%, are shown in Table 4.3. First, we observe that baseline performance confirms the difficulty and balance of WikiAlumni. Indeed, feature-based accuracy, as measured by the performance of the MLP baseline, is at 35.5%, whereas edge-based accuracy, measured using the LP baseline (32%) and entity classification model results ($\sim$ 33.5%) are both substantial. These results highlight the importance of both features and edge structure on WikiAlumni. Moreover, MPNNs, which combine node features with edge information, achieve substantially higher performance (42.4%), which confirms that

both information sources do not overlap and can be combined effectively. Finally, all models fail to achieve even 50% accuracy, highlighting the difficulty of WIKIALUMNI.

On the node classification task, rGCN and GAT perform best among all models on the full WIKIALUMNI, with MLP-BoxE(CE) slightly behind, and MLP-TransE(CE) and MLP-RotatE(CE) performing competitively. This aligns with expectations, as MPNNs are strong on this task, and as WIKIALUMNI is at its most relationally complete. However, as we drop edges by moving to the sparser subsets, this performance trend reverses: The performance of rGCN and GAT falls significantly, while MLP-X models are more robust. In fact, MLP-BoxE(CE) and MLP-TransE(CE) start on average around 0.5% behind both MPNNs , but ultimately surpass the accuracy of GAT and rGCN by 0.3% and 1.3% respectively on WIKIALUMNI-80%. This result is statistically meaningful, as the standard deviations observed in our experiments do not exceed 0.2%. Hence, these results suggest that the link prediction capability of MLP-X models compensates for the loss of dropped edges.

To illustrate this behaviour, we present a case study on a WIKIALUMNI sub-graph, shown in Figure 4.4, using the best-performing MLP-X model, MLP-BoxE. In this sub-graph, entity "Edmund Dollard" appears with three neighbours. However, only one neighbour, the "United States" nationality, survives in WIKIALUMNI-80%. Clearly, this neighbour is not sufficient to classify the entity, and therefore MPNNs lose key information. By contrast, MLP-BoxE successfully completes these missing edges, and thus recovers the lost information due to its link prediction ability. In fact, MLP-BoxE predicts the memberOf edge with a better score (3.99) (scores are distances in BoxE, and thus lower scores are better, cf. Section 3.3) than the nationality edge it has trained on (4.33), and also predicts the hasOccupation edge with a comparable score (5.19). Hence, MLP-BoxE can

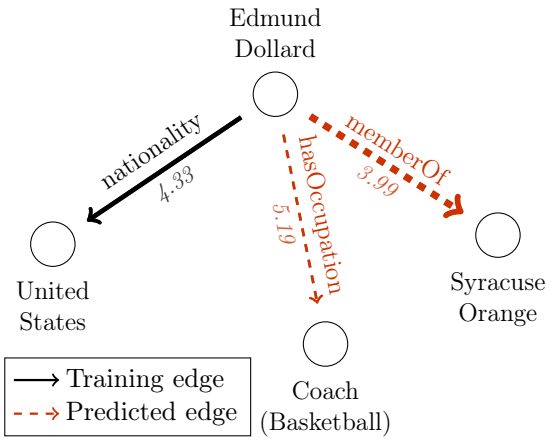


Figure 4.4: Visualisation of MLP-BoxE completion on a sub-graph of WIKIALUMNI. Based on the features of "Edmund Dollard" and KG structure, MLP-BoxE predicts his membership of Syracuse Orange, and also strongly predicts his occupation as a basketball coach. Both edge completions significantly simplify the class prediction of Syracuse University.

leverage the recovered edges to classify "Edmund Dollard" as an alumnus of Syracuse University.

This case study shows a subtle but insightful picture for transductive node classification with MPNNs, and paints a sharp contrast in performance relative to relational completeness. On one hand, more complete edge information enables both rGCN and GAT to perform high-quality aggregation and make good predictions without suffering from their lack of link prediction capacity. On the other hand, as relational data becomes incomplete, aggregation becomes noisier, and this compromises prediction quality: GAT and rGCN both suffer significantly with incompleteness. Interestingly, rGCN performance falls much more than GAT. However, this behaviour can be explained by differences in aggregation between the two models. Indeed, as GAT aggregates using attention, it is less susceptible to noise, as it can "ignore" noisy neighbours. However, rGCN, by construction, considers all nodes, however irrelevant, and aggregates uniformly across relations. Hence, GAT is more resilient to relational incompleteness than rGCN, but in return is more heavily dependent on node features.

Looking back at MLP-X models, we can see that these models perform less well than MPNNs when edges are relatively more complete. This is sensible, as their link prediction component will not recover many meaningful missing edges, and will essentially be tangential to node classification and produce minimal returns. However, with incomplete data, the benefit of link prediction becomes much more tangible, as models can recover potentially critical missing edges, thereby enabling stronger node classification performance. Note, however, that this benefit is intrinsically tied to the *quality* of link prediction. Indeed, were a model to jointly make poor link predictions, then this would result in noisy, and even wrong, neighbourhoods, which effectively hurt node classification and in fact makes performance deteriorate *faster* with incompleteness. Therefore, the resilience of MLP-X models indicates that these models perform link prediction *with high quality*.

On the entity classification task, MLP-X models, i.e., shallow node embedding models with class-aware negative sampling, trained with negative sampling loss [159] significantly outperform all other models, whereas MPNNs, even with learnable embeddings, cannot surpass the LP baseline. Therefore, the link prediction capacity of the node embedding models, paired with their relational inductive bias, offers them a strong advantage on entity classification. Interestingly, negative sampling loss enables better performance for entity classification, while cross-entropy is better for node classification. This can be explained by the assumptions made within each loss function (cf. Chapter 2). With negative sampling loss, the margin hyper-parameter

sets *absolute*, *uniform* ranges for fact scores, such that positive facts are trained to score slightly lower than the margin (lower is better), and negative facts score slightly higher. Therefore, margin sets a uniform score range for all facts in the data. This uniformity is sensible for entity classification, as it aligns with the lack of distinguishing prior knowledge, i.e., features in this task. However, this same uniformity is potentially harmful for node classification, as uniformly setting target scores ignores the variability provided by features. By contrast, cross-entropy optimises the *relative* difference between positive and negative fact scores, and does not set an absolute score range. Instead, scores are *data-driven*. Hence, cross-entropy benefits from features, and uses their variability to produce more representative fact scores.

Finally, we note that class-aware negative sampling is essential for achieving strong results. In fact, without this sampling, MLP-X model performance for node classification drops by around 2%. This confirms the insights discussed in Section 4.3, and highlights the importance of selecting higher-quality negative examples to enforce mutual exclusion between classes.

4.5.2 Link Prediction with Features and Classes

Experimental setup. We now evaluate MLP-X models on the link prediction task using WikiAlumni-80%. More precisely, we train on this subset and evaluate link prediction performance on the edges dropped from WikiAlumni, i.e., the removed 20% of edges. In this experiment, we follow an analogous protocol to the node classification experiment, in that we train with node features, as well as without (analogously to entity classification). Moreover, we also train all MLP-X models using both negative sampling and cross-entropy loss. However, in this setting, we additionally consider configurations where we omit node classes, so as to identify whether class information alone contributes positively to link prediction. Therefore, we study four different configurations, relative to the use/omission of node features and classes. Finally, we evaluate using standard metrics, namely mean rank (MR), mean reciprocal rank (MRR), and Hits@10 (H@10) [21].

Results. The results for all four link prediction configurations across all MLP-X models with both NS and CE loss are shown in Table 4.4. Across all models, the inclusion of node features allows for substantial improvements. Indeed, the best MR (1544 for TransE with classes, 1780 for BoxE without classes) is more than halved (to 747 and 781 respectively). Furthermore, MRR improves by at least 0.02, both with and without classes. The use of class information also slightly improves performance, and this improvement is more pronounced when node features are not used. In the

Table 4.4: Link prediction results for MLP-X models and their featureless counterparts on WikiAlumni-80%, using both NS and CE loss. Results are categorised per feature/class configuration and, for each setup, the top result is reported in bold.

Model	Classes			No Classes		
	MR	MRR	H@10	MR	MRR	H@10
TransE(NS)	**1544**	0.265	0.382	1843	0.265	**0.376**
RotatE(NS)	1655	**0.298**	**0.386**	1806	**0.285**	0.375
BoxE(NS)	1768	0.250	0.344	**1780**	0.245	0.341
TransE(CE)	2063	0.255	0.346	2162	0.246	0.340
RotatE(CE)	2934	0.236	0.335	3673	0.230	0.317
BoxE(CE)	2170	0.288	0.369	2298	**0.285**	0.366
MLP-TransE(NS)	930	0.285	0.407	988	0.275	0.402
MLP-RotatE(NS)	898	**0.316**	**0.417**	932	0.314	0.414
MLP-BoxE (NS)	1025	0.257	0.374	1077	0.256	0.368
MLP-TransE(CE)	**747**	0.242	0.378	**781**	0.235	0.370
MLP-RotatE(CE)	753	0.291	0.416	816	0.288	**0.416**
MLP-BoxE(CE)	883	**0.316**	0.415	947	**0.315**	0.413

latter setup, training on classes, i.e., unary facts, yields an average MR drop of 225 and Hits@10 improvement of 0.007 across all models.

These findings align with our expectations, as features provide rich context for node pairs. In particular, knowing entity features simplifies comparison between pairs and establishes more robust similarities and discrepancies, which ultimately lead to higher-quality entity representations. The improvement due to node features is consistent across all models and loss choices, which shows the universal value of features for representation learning. Indeed, features provide a dual advantage: They simplify node classification by providing additional information, and also indirectly improve class predictions by enabling better link prediction as mentioned earlier. Therefore, features are valuable resources for knowledge bases, and should be included within standard knowledge graph benchmarks and leveraged by existing models. This observation is a main motivation behind this work, and also motivates the development of many recent multi-modal knowledge graphs [105].

When features and classes are both used, all models perform strongly with cross-entropy loss, and achieve good mean rank ($\sim$ top 1.5-1.8% of all possible entities) and Hits@10. These strong results confirm our earlier intuition about the high quality of link prediction performed by MLP-X models, and further validate that this link

prediction indeed enables the earlier node classification results with relational incompleteness. Interestingly, we also observe the opposite phenomenon with RotatE(CE) (without node features). Here, this model achieves substantially worse link prediction performance relative to its counterparts, and this aligns with its poor performance on entity classification. Concretely, RotatE(CE) is the least robust of the MLP-X models on entity classification when edges are dropped, with its accuracy falling by over 5% (cf. Table 4.3), and this can now be explained by the aforementioned poor link prediction performance.

In terms of loss function choices, we also observe an interesting contrast in this link prediction setup. Indeed, MLP-TransE and MLP-RotatE perform better with negative sampling loss without node features, consistently with entity classification. However, with features, cross-entropy significantly improves MR but *worsens* MRR, despite this loss improving node classification performance for both models. This initially appears surprising, but can be attributed to a combination of improved node classification with the limited expressiveness of MLP-TransE and MLP-RotatE: As cross-entropy improves node classification performance (cf. Section 4.5.1), unary facts are better fit, and thus their representations more closely align with the TransE (resp., RotatE) scoring function. However, better fitting in the context of MLP-TransE and MLP-RotatE imposes, by construction, that entities belonging to a same class cluster around one same point (cf. Example 4.3.1). Hence, as cross-entropy improves node classification, entities in both models are more clustered. This implies better *overall* fact prediction, reflected by better MR, but also reduces both models' ability to optimise individual facts, and therefore worsens MRR. Hence, the interplay between node classification and link prediction takes on an added dimension in these models, as their limited expressiveness enforces clustering with improved node classification, and ultimately makes their link predictions less fine-grained.

Interestingly, MLP-BoxE does not exhibit the same behaviour as MLP-TransE and MLP-RotatE, and in fact achieves its best performance with cross-entropy loss. Looking deeper, MLP-BoxE is fully expressive, and thus does not suffer from the clustering limitation stemming from the many-to-one bottleneck of TransE and RotatE. Furthermore, cross-entropy is better suited to the BoxE scoring mechanism than negative sampling loss. Indeed, BoxE assigns scores at every arity position and sums these up to yield an overall fact score. Hence, the range of binary fact scores is much higher and larger than that of unary facts by design. Therefore, negative sampling loss, which sets absolute score ranges for all positive facts, irrespective of arity, leads to sub-optimal training. By contrast, cross-entropy is agnostic to score

Table 4.5: Node classification results (accuracy) for MLP-BoxE and competing models on public splits of standard citation network benchmarks (Test set). Results reported for other models are the best published for our setup.

Model	CiteSeer	Cora	PubMed	OGBN-arXiv
MLP [169, 78]	46.5	55.1	71.4	55.5
GCN [169, 92, 78]	70.9	81.5	79.0	**71.7**
GAT [169]	**72.5**	**83.0**	79.0	_
MLP-BoxE	70.2	78.4	**81.4**	69.4

values, and considers the relative difference between scores. Therefore, it provides a better optimisation for BoxE, yielding better results.

All in all, these results suggest that MLP-X models achieve strong performance on node classification and link prediction, despite their generality, and highlight that this unifying perspective leads to substantial benefits in terms of robustness and performance improvements.

4.5.3 Experiments on Standard Citation Network Benchmarks

Experimental setup. To evaluate the modelling ability of MLP-X models on less relationally informative data, we train MLP-BoxE, the best-performing node classification model among MLP-X models, on three standard citation network benchmarks, namely Citeseer, Cora, and PubMed [153], as well as the recently introduced OGBN-arXiv [78] benchmark. For Citeseer, Cora, and PubMed, we found that MLP-BoxE overfit with a two-layer MLP, given the small size of these benchmarks. Therefore, we only use a single hidden layer of size 1000 on these benchmarks. Furthermore, we use dropout on the MLPs, and tune this across the range [0,1] in increments of 0.1. For embedding dimensionality, we found that MLP-BoxE surprisingly performs best and in the most stable fashion with only 20 dimensions, also due to the sparsity and low label rate of these benchmarks. For OGBN-arXiv, we use the standard dimensionality of 256 used in the literature, a two-layer MLP, as in the main experiments, and train using negative sampling loss. Finally, across all experiments, we use a learning rate of 0.001, and train using 10 negative samples per positive fact. The final hyper-parameters used are shown in Table A.4, in the appendix of this thesis.
Results. The results of this experiment are shown in Table 4.5. Across all datasets, MLP-BoxE performs strongly, and remains competitive with GNNs, even achieving state-of-the-art performance on PubMed. This is very encouraging, as these benchmarks are single-relational and, with the exception of OGBN-arXiv, highly sparse in

terms of node labels. Therefore, MLP-BoxE performs competitively even in this very specialised setting where its relational inductive biases are not beneficial.

4.5.4 Experiments with Bag-of-Words Features on WikiAlumni

Experimental setup. To evaluate the impact of feature quality on different node classification models, we perform an ablation study on the features of WikiAlumni. In particular, we replace the original 300-dimensional GloVe mean vector features with a binarised 100-dimensional bag-of-words feature vector, consisting of the 100 most frequent words across the corpus of all entity Wikipedia descriptions. We then re-train all models on the new dataset following the same protocol as the original node classification experiments. For this experiment, we train MLP-X models with both cross-entropy and negative sampling loss, but only report the latter as CE was not competitive in this setup (likely due to poor feature quality, similarly to CE vs NS in entity classification).

Results. Results for this ablation study are provided in Table 4.6. Here, we observe that the performance of all models drops severely, as expected. However, GAT drops the most substantially, even falling behind rGCN. On the other hand, MLP-X models now comfortably outperform GAT, despite GAT previously outperforming these models with GloVe features on WikiAlumni. Nonetheless, all models comfortably beat the MLP baseline, and do so by a higher margin relative to the original experiment, which suggests that all models exploit edge information to compensate for lost features to a certain extent.

Table 4.6: Results with 100-dimensional bag-of-words features on the full WikiAlumni dataset.

Model	Accuracy (%)
MLP	19.0
rGCN	33.1
GAT	30.9
MLP-TransE(NS)	**34.3**
MLP-RotatE(NS)	**35.1**
MLP-BoxE(NS)	**34.9**

From these results, we can confirm that GAT heavily relies on feature quality, and suffers greatly when features are poor, or unavailable, as in entity classification. rGCN also relies on features, but responds better to the loss of feature quality, as it uses relational structure more effectively. Finally, MLP-X models, owing to their relational inductive bias, more effectively exploit relational information, and now gain an advantage against MPNNs, as these models can no longer make up the performance gap through feature processing.

4.6 Summary and Outlook

In this chapter, we proposed a unifying perspective combining transductive node classification over incomplete graphs with link prediction over graphs with node features and classes. To study this setting practically, we proposed a new framework, MLP-X, and dataset, WikiAlumni, and conducted an extensive benchmarking study. Through this study, we highlighted the complementarity of node classification and link prediction over incomplete graphs, and showed how jointly modelling both edges and nodes when learning over graphs achieves substantial improvements in terms of model robustness. Hence, a natural next question is whether these benefits can extend to other transductive tasks, such as reasoning tasks with long-range dependencies. In this setting, MLP-X models could not only perform strongly, but can also potentially circumvent limitations such as oversmoothing [100] and oversquashing [8].

The robustness of MLP-X models also raises important questions about assumptions made when processing graphs. In particular, MPNNs only aggregate information between known neighbours, and hence implicitly make no attempt to infer any edges beyond the provided graph edges. By contrast, MLP-X models do not assume edge completeness. Therefore, the results of this chapter motivate further research exploring different approaches more conceptually connected with embedding techniques, such as node2vec [70] and metapath2vec [52], combining these with the strengths of MPNNs, and ultimately proposing a comprehensive new setup and benchmarks to this end within graph representation learning.

Chapter 5

The Universality of Graph Neural Networks with Random Node Features

5.1 Introduction

As discussed in Chapter 2, graph neural networks (GNNs) [149, 67], particularly message passing neural networks (MPNNs) [62], naturally encode many desirable properties, such as permutation invariance (resp., equivariance) with respect to neighbourhood nodes, and local node-level computation. These properties in turn provide MPNNs with a strong inductive bias, enabling them to effectively learn and combine both local and global graph features [15]. As a result, MPNNs are very successful across multiple application domains, ranging from protein classification [62] and synthesis [188], protein-protein interaction [60], and social network analysis [74], to recommender systems [186] and combinatorial optimisation [18]. However, despite their popularity and empirical strength, MPNNs are limited in their expressive power to the Weisfeiler-Leman (1-WL) graph isomorphism heuristic [127, 183]. Therefore, MPNNs cannot discriminate between simple pairs of non-isomorphic graphs, e.g., regular graphs [29].

To address this limitation, alternative GNN models with provably higher expressive power, such as k-GNNs [127] and invariant (resp., equivariant) graph networks [115], have been proposed. These models, primarily referred to as *higher-order GNNs*, take inspiration from a generalisation of 1-WL to node k-tuples, k-WL [29]. By considering tuples of nodes and computing hashes based on their different types, k-WL improves on the expressive power of 1-WL. However, this improvement comes at a prohibitive computational cost, as the number of tuples considered grows expo-

nentially with respect to k. Therefore, k-WL, and as a result higher-order models, are computationally very demanding.

Alternatively to higher-order GNNs, recent works have improved the power of MPNNs while persisting with the standard message passing framework, using *random node features* [45, 148]. In this setting, initial node features are fully or partially set at random, and this *random node initialisation* (RNI) is shown to improve the power of MPNNs beyond 1-WL. In particular, RNI enables MPNNs to detect *fixed* substructures, such as triangles, and also allows MPNNs to better approximate solutions for a class of combinatorial problems [148]. However, these findings do not provide a holistic understanding of the expressiveness gain stemming from RNI. Hence, the overall theoretical impact of RNI on MPNNs, both from an expressiveness and generalisation perspective, remains *unknown*.

In this chapter, we study the impact of RNI on the expressive power of MPNNs from a theoretical perspective, and show that MPNNs enhanced with RNI are *universal*, and thus can approximate every invariant function defined on graphs of any fixed order. This universality result follows from an earlier logical characterisation of the expressiveness of MPNNs [13] combined with an argument on order-invariant definability. This result is surprising, as it implies that simply adding random features, which themselves are not informative, is sufficient for MPNNs to overcome their expressiveness limitations at a minimal computational cost. Importantly, the introduction of RNI preserves the inductive biases of MPNNs. More concretely, the standard message passing protocol is unaffected and agnostic to random features, and permutation invariance is preserved in expectation when RNI is uniform across all nodes. Hence, our result provides a strong foundation for developing expressive and memory-efficient MPNNs with strong inductive bias.

Parallel to the main theoretical result, we propose a rigorous empirical evaluation to validate the effect of RNI. In particular, we propose EXP, a synthetic dataset that cannot be solved by standard MPNNs, and which can be solved by a model with 2-WL expressive power, and run MPNNs with RNI on it to observe *how well* and *how easily* these models learn and generalise given the added randomness. Then, we propose a more challenging setup by modifying EXP with 1-WL distinguishable data, yielding the dataset CEXP, and evaluate the same questions in this setting.

The rest of this chapter is organised as follows: We present some necessary logic preliminaries in Section 5.2, and state our main result, prove it, and discuss the main properties of MPNNs with RNI in Section 5.3. We then discuss our dataset design for EXP and CEXP in Section 5.4, and conduct an empirical analysis of MPNNs

with RNI in Section 5.5. Finally, we present related work on the expressive power of various GNN approaches and place our work within this context in Section 5.6, and conclude this chapter in Section 5.7.

5.2 The Logical Characterisation of MPNNs

The main result of this chapter builds on the recently established logical characterisation of MPNNs with global readout [13]. Therefore, we present the key concepts needed for this characterisation, namely the logic C^2, and then discuss the characterisation result.

The logic C^2. We consider the logic C, an extension of first-order logic (cf. Chapter 2) that introduces counting quantifiers of the form $\exists^{\geq k} x$ for $k \geq 0$, where $\exists^{\geq k} x\, \varphi(x)$ means that there are at least k elements satisfying φ. Moreover, we are interested in C formulas in the *language of graphs*. More concretely, the language of graphs consists of a signature with a single binary relation $E(x, y)$, indicating the presence of an edge between two graph nodes interpreting x and y. To illustrate, consider the formula:

$$\varphi(x) := \neg\exists^{\geq 3} y\big(E(x, y) \wedge \exists^{\geq 5} z E(y, z)\big). \tag{5.1}$$

This formula is defined in the language of graphs with respect to a variable x, interpreted by a node, and holds true when this node has at most 2 neighbors of degree 5 or higher in the graph. More formally, given a graph $G(V, E)$ and a vertex $v \in V$, we say that G *satisfies* φ when x is interpreted by v, denoted $G \models \varphi(v)$.

The language of graphs can be extended to set of unary relations to describe node features, which can be also be seen as node colours. This addition yields the language of *coloured* graphs. For example, the formula:

$$\psi(x) := \exists^{\geq 4} y\big(E(x, y) \wedge \mathit{Red}(y)\big) \tag{5.2}$$

is true when the node interpreting x has at least 4 red neighbours. Formally, the language of coloured graphs assumes a fixed, potentially infinite list $R_1, R_2, \ldots$ of unary colour symbols in its signature. These symbols can then be used to define a *coloured graph* by mapping each vertex to a finite set of (one or more) colours. In our setting, we consider a finite set of r colours $R_1, \ldots, R_r$, and represent colour membership through an r-dimensional one-hot initial feature vector for every node. That is, for a node $v \in V$, we initialise the vector $\boldsymbol{x}_v = (x_{v1}, \ldots, x_{vd})$ such that $x_{vi} = 1$ if v has colour R_i and $x_{vi} = 0$ otherwise.

As in standard (first-order) logic, a *sentence* in C is a formula without a free variable. In the language of (coloured) graphs, sentences therefore describe graph-level properties, and thus can be used to express Boolean functions over graphs. In particular, given a sentence φ describing a graph-level property, we denote the corresponding 0/1 function by $[\![\varphi]\!]$, such that, for an input graph G, $[\![\varphi]\!](G) = 1$ if $G \models \varphi$ and $[\![\varphi]\!](G) = 0$ otherwise.

In terms of expressiveness, C does not improve on standard first-order logic, and instead only offers a syntactic extension. Indeed, every C formula can equivalently be written in standard FO. To see this, note that counting quantifiers in C, namely $\exists^{\geq k} x$, can be exactly captured using k standard existential quantifiers. More concretely:

$$\exists^{\geq k} x \Leftrightarrow \exists x_1 \ldots \exists x_k \Big(\bigwedge_{1 \leq i < j \leq k} x_i \neq x_j \wedge \bigwedge_{1 \leq i \leq k} \varphi(x_i) \Big). \tag{5.3}$$

However, this scenario changes when the number of variables is *bounded*, i.e., the number of variables in a formula is limited to a finite value. In this bounded setting, counting quantifiers no longer act as just syntactic extensions, but in fact improve on the expressiveness on the corresponding bounded variable first-order fragment, as using, e.g., $\exists^{\geq k} x$ avoids the use of k variables in the formula, and cannot be simulated by any FO-formula using less than k variables. For conciseness, we denote the fragment of C consisting of all formulas with at most k variables by C^k, and the corresponding FO formula by FO^k. For instance, $\varphi(x)$ in Equation (5.1) is in C^3, but can be written in C^2, as:

$$\varphi'(x) := \neg \exists^{\geq 3} y \big(E(x, y) \wedge \exists^{\geq 5} x \; E(y, x) \big). \tag{5.4}$$

The bounded-variable fragments C^k are very relevant in the context of GNNs, as they correspond directly to the $(k-1)$-WL graph isomorphism heuristic, and thus to k-GNNs [127]. In particular, it is known that two graphs G and H can be distinguished by $(k-1)$-WL if and only if there exists a C^k sentence with different outputs on G and H [29]. Hence, in the case of standard MPNNs, with 1-WL discriminating power [127, 183], this result implies that two graphs are indistinguishable by all MPNNs if and only if they satisfy exactly the same C^2 sentences.

Logical characterisation. The correspondence between 1-WL and C^2 has recently been strengthened into a full logical characterisation that describes the expressive power of MPNNs in terms of the Boolean *classifiers* they can learn. In particular, this characterisation shows that MPNNs can accurately learn any classifiers which can be expressed in the logic C^2 [13]. In this context, a *logical node classifier* $\varphi(x)$ is

a first-order formula (in C^2 for this result) with one free variable. Moreover, we say that a model $\mathcal{M}$ *captures* a logical classifier $\varphi(x)$ when both $\mathcal{M}$ and $\varphi(x)$ coincide over every possible input. That is, for any graph G and node $u \in G$, $\mathcal{M}$ evaluates to True if and only if $\varphi(x)$ evaluates to True. Finally, An MPNN model $\mathcal{M}$ captures a logic $\mathcal{L}$ if for every $\varphi(x) \in \mathcal{L}$, there exists an MPNN that $\mathcal{M}$ captures $\varphi(x)$.

In what follows, we say that a function $\mathcal{X}$ over graphs is an ϵ-approximation of a target function f if, for any input graph G, it holds that $|f(G) - \mathcal{X}(G)| \leq \epsilon$. We now formally state the characterisation result with a reformulation using ϵ- approximation:

Theorem 5.2.1 ([13]). *For every C^2-sentence φ and every $\epsilon > 0$ there is an MPNN with global readout that ϵ-approximates $[\![\varphi]\!]$.*

Unlike previous results relating MPNNs to 1-WL, and which describe the pairs of graphs MPNNs can *distinguish*, this logical characterisation describes the Boolean functions that MPNN can *learn*. Indeed, this characterisation shows that any Boolean function that can be expressed as a formula in C^2 can be approximated to an arbitrary error ϵ by an MPNN. However, this characterisation relies on a *global readout* as part of message passing: Global readout computes a graph-level aggregate of all node representations, and passes it to nodes as part of the message passing process. More formally, the MPNN with global readout used in this result, known as ACR-GNN, uses l-dimensional embedding representations and computes representation updates using the following update layer:

$$\mathbf{h}_u^{(t+1)} = f(\mathbf{W}_C\ \mathbf{h}_u^{(t)} + \mathbf{W}_A(\sum_{v \in N(u)} \mathbf{h}_u^{(t)}) + \mathbf{W}_R(\sum_{\mathbf{v} \in V} \mathbf{h}_v^{(t)}) + \mathbf{b}), \tag{5.5}$$

where f is the truncated ReLU non-linearity $f(x) = \max(0, \min(x, 1))$, $\mathbf{W}_C, \mathbf{W}_A, \mathbf{W}_R \in \mathbb{R}^{l \times l}$ are learnable matrices, $\mathbf{h}_u^{(t)} \in \mathbb{R}^l$ is the current state of node u, and $\mathbf{b} \in \mathbb{R}^l$ is a learnable bias vector. In this equation, $\mathbf{W}_C$ transforms the current node representation, $\mathbf{W}_A$ acts on the neighbourhood of every node, and $\mathbf{W}_R$ transforms the global readout, computed as a sum of all current node representations.

At a high level, the logical characterisation of MPNNs with global readout to C^2 is a *constructive* proof, which sets values for $\mathbf{W}_C, \mathbf{W}_A, \mathbf{W}_R$ and $\mathbf{b}$ so as to exactly learn the target Boolean formula φ. This construction is *homogeneous*, in that the same layer is used multiple times across all message passing iterations. Furthermore, this construction is *adaptive*, as the size of the MPNN depends exactly on the complexity of the formula φ. More specifically, the embedding dimensionality of the ACR-GNN, l, exactly corresponds to the number of *sub-formulas* in φ, and the depth of the

model depends on the *quantifier depth* q of φ, which is the maximum nesting level of existential counting quantifiers in φ. For example, the formula

$$\varphi := \exists^{\geq 2} x \big(\exists^{\geq 3} y \; E(x, y) \big)$$

has a quantifier depth of 2.

At a high level, the proof construction represents every *sub-formula* in the target formula φ with a dedicated embedding dimension, which is set to 1 when the corresponding sub-formula holds, and 0 otherwise. Sub-formulas are then traversed recursively up to the top-level formula φ, based on the different logical operands ($\wedge, \vee, \exists$, etc) in φ, and entries of $\mathbf{W}_C, \mathbf{W}_A, \mathbf{W}_R$ and $\mathbf{b}$, are assigned values to fully align with the semantics of the corresponding operands and traversal.

We illustrate this construction with a simple example. Consider the formula:

$$\varphi(x) = \mathrm{Red}(x) \wedge \mathrm{Blue}(x).$$

This formula has 3 sub-formulas, namely (i) the Red atom $\mathrm{Red}(x)$, (ii) the Blue atom $\mathrm{Blue}(x)$, and (iii) their conjunction $\varphi(x)$. We therefore use 3-dimensional embeddings, i.e., $l = 3$, and denote the corresponding dimensions for each sub-formula as x_1, x_2, and x_3 respectively for simplicity.

To represent the conjunction between Red and Blue (sub-formulas 1 and 2), the construction sets $\mathbf{W}_{C,13} = \mathbf{W}_{C,23} = 1$ and $\mathbf{b}_3 = -1$, where $\mathbf{W}_{C,ij}$ denotes the scalar at row i and column j of matrix $\mathbf{W}_C$. This way, an ACR-GNN update only yields an output of 1 at dimension 3, i.e., x_3, if dimensions 1 and 2, i.e. x_1 and x_2, are both 1, in line with conjunction semantics. Similar constructions are defined for all other operands in C^2 so as to enable a full traversal of φ, and the resulting ACR-GNN provably captures φ. For further details about the proof, we refer the reader to the original paper [13].

5.3 MPNNs with Random Node Initialisation

In this section, we show that random node initialisation makes MPNNs universal approximators for invariant functions in the graph domain. This result paints a positive picture regarding the discriminating power of MPNNs using continuous random features, and offers a new theoretical perspective on the role played by randomisation when learning functions over graphs. Importantly, this universality result is not obtained by modifying the MPNN framework, but instead by only introducing random

features, an increasingly popular approach in practice. Therefore, this result also serves as a theoretical justification for the recent empirical success of RNI.

At first inspection, it seems somewhat counter-intuitive and surprising that randomly initialised node features would deliver a universality result for MPNNs, particularly as these features are not informative and do not relate to graph structure in any way. Moreover, these features, on the surface, cost MPNNs their permutation invariance, as different initialisations for the same node neighbourhoods, or even an inversion of random features across this neighbourhood, can yield distinct outputs. Hence, MPNN outputs with RNI no longer primarily rely on the graph structure, but also build on an external random input.

Looking more holistically, however, the contribution of RNI is rather subtle. Indeed, one can see node representations initialised with RNI as representing a random variable, where features correspond to a sample of the random distribution. Hence, one can view MPNNs with RNI as computing a random variable, or generating an output distribution, which itself would remain invariant *in expectation*. With this probabilistic perspective, it is clear that the output of MPNNs with RNI would not be tied to specific instantiations of random features. In fact, RNI fundamentally maintains invariance in expectation, as all random node features are independently and identically distributed, and thus the mean of these features is identical across nodes. From an expressiveness perspective, the variability across different feature samples enables graph discrimination and improves expressiveness. Therefore, RNI achieves an interesting duality at the sample and distribution level, where individual samples improve expressiveness, while distribution-level invariances maintain the desirable inductive biases of MPNNs in expectation.

5.3.1 Universality Result

For our result, we consider $\mathcal{G}_n$, the class of graphs with at most n vertices, as our input domain, and wish to learn real-valued deterministic functions $f : \mathcal{G}_n \to \mathbb{R}$. However, MPNNs with RNI, as mentioned earlier, can be seen as learning output distributions and computing random variables. Therefore, in this setting, we consider learning *randomised functions*, which associate every graph $G \in \mathcal{G}_n$ with a random variable $\mathcal{X}(G)$. In particular, we would like to learn randomised functions that closely approximate a deterministic target function f over $\mathcal{G}_n$. More formally, we would like $\mathcal{X}(G)$ to yield an (ϵ, δ)-*approximation* of f, such that for all $G \in \mathcal{G}_n$, $\Pr\left(|f(G) - \mathcal{X}(G)| \leq \epsilon\right) \geq 1 - \delta$. If an MPNN with RNI $\mathcal{N}$ satisfies these conditions relative to a desired error ϵ and confidence δ, we say that $\mathcal{N}$ (ϵ, δ)-*approximates* f.

Given this setup, we prove the following result for MPNNs with RNI:

Theorem 5.3.1 (Universal approximation)**.** *Let $n \geq 1$, and let $f : \mathcal{G}_n \to \mathbb{R}$ be an invariant function. Then, for all error and confidence values $\epsilon, \delta > 0$, there exists an MPNN with RNI that (ϵ, δ)-approximates f.*

For ease of presentation, we state the theorem only for real-valued functions, but note that it can be easily extended to equivariant functions.

To prove Theorem 5.3.1, we first prove an analogous lemma for Boolean functions over $\mathcal{G}_n$, and subsequently extend the result to the real-valued setting. More concretely, we first prove the following Lemma.

Lemma 5.3.1. *Let $n \geq 1$, and let $f : \mathcal{G}_n \to \{0, 1\}$ be an invariant Boolean function. Then, for all $\epsilon, \delta > 0$ there is a MPNN with RNI that (ϵ, δ)-approximates f.*

To prove this lemma, we use the logical characterisation of the expressiveness of MPNNs with global readouts, from Barcelo et al. [13] and apply it to *individualised* coloured graphs G, where, for any two distinct vertices $u, v \in V$ the sets $\rho(u), \rho(v)$ of colours they have are distinct. In other words, individualised coloured graphs have colour allocations that are unique to each node.

As individualised coloured graphs have unique colour combinations at every node, they can easily be described using logic, by identifying nodes using their colours and subsequently enforcing edges between them. In fact, individualised coloured graphs can directly be mapped to a corresponding logical sentence in C^2 which only holds true for the input graph (up to isomorphism).

In what follows, we say that a sentence χ *identifies* a (coloured) graph G if for all (coloured) graphs H we have $H \models \chi$ if and only if H is isomorphic to G. As a first step towards proving Lemma 5.3.1, we now show that every individualised coloured graph can be identified by a sentence in C^2:

Lemma 5.3.2. *For every individualised coloured graph G, there exists a C^2-sentence χ_G that identifies G.*

Proof. To see this, consider an individualised coloured graph $G(V, E)$, and define, for every vertex $v \in V$, the formula:

$$\alpha_v(x) := \bigwedge_{R \in \rho(v)} R(x) \wedge \bigwedge_{R \in \{R_1, \ldots, R_k\} \setminus \rho(x)} \neg R(x), \tag{5.6}$$

where $R_1, \ldots, R_k$ are colour predicates in the language of coloured graphs, and $\rho(v)$ is the set of true colours of node v. This α_v formula uniquely identifies every v in G, as

the colour combination $\rho(v)$ is unique to v given that G is individualised. Therefore, v is the unique vertex of G such that $G \models \alpha_v(v)$. Using these α_v formulas, we can encode edges between pairs of nodes $v, w \in V$, as follows:

$$\beta_{vw}(x,y) := \begin{cases} \alpha_v(x) \wedge \alpha_w(y) \wedge E(x,y) & \text{if } (v,w) \in E(G), \\ \alpha_v(x) \wedge \alpha_w(y) \wedge \neg E(x,y) & \text{if } (v,w) \notin E(G). \end{cases} \tag{5.7}$$

Hence, for every pair of nodes $v, w \in G$, we can define β_{vw} based on the existence of an edge between v and w in G, such that β_{vw} for all v, w pairs all hold true exclusively on G. Finally, we combine α and β into an overall formula χ_G, so as to (i) impose uniqueness of each node using α_v, and (ii) impose the edge structure in G using β_{vw}. This yields the formula:

$$\chi_G := \bigwedge_{v \in V} \left(\exists x \; \alpha_v(x) \wedge \neg \exists^{\geq 2} x \; \alpha_v(x) \right) \; \wedge \bigwedge_{v,w \in V} \exists x \exists y \beta_{vw}(x,y).$$

As G is individualised, χ_G is sufficient to uniquely identify G, proving the lemma. □

As individualised coloured graphs can directly be mapped to identifying sentences in C^2, we can easily extend Lemma 5.3.2 to *finite* sets of graphs by considering the disjunction of all χ_G formulas for all formulas in the set. This implies that Boolean functions over individualised coloured graphs (up to a size n) can also be learned by simply identifying the set of "True" graphs using the aforementioned disjunction.

Formally, given $n, k \in \mathcal{N}$, let $\mathcal{G}_{n,k}$ be the (finite) class of individualised coloured graphs using colour predicates $R_1, \ldots, R_k$. We obtain the following lemma:

Lemma 5.3.3. *Let $h : \mathcal{G}_{n,k} \to \{0,1\}$ be an invariant Boolean function. Then there exists a C^2-sentence ψ_h such that for all $G \in \mathcal{G}_{n,k}$ it holds that $[\![\psi_h]\!](G) = h(G)$.*

Proof. Consider $\mathcal{H} \subseteq \mathcal{G}_{n,k}$, the subset of all graphs H with $h(H) = 1$. We now define the disjunction:

$$\psi_h := \bigvee_{H \in \mathcal{H}} \chi_H,$$

to capture h. This disjunction is well-defined, as $\mathcal{G}_{n,k}$, up to isomorphism, is finite, and therefore any $\mathcal{H}$ is also finite. □

Hence, the logic C^2 is sufficient to describe any Boolean function over $\mathcal{G}_{n,k}$. Therefore, MPNNs with global readout can learn any (invariant) Boolean sentence over $\mathcal{G}_{n,k}$ due to the established logical characterisation. However, this result does *not* apply to arbitrary graphs, as required to prove Lemma 5.3.1, as nodes in $\mathcal{G}_n$ are not necessarily

uniquely coloured. Therefore, as the next step towards proving Lemma 5.3.1, we use RNI to stochastically map input graphs $G \in \mathcal{G}_n$ with high probability to its (finitely many) possible individualised colourings, denoted $G^{\wedge} \in \mathcal{G}_{n,k}$. Using these coloured versions of G, we then apply Lemma 5.3.3 and learn a function $f' : \mathcal{G}_{n,k} \to \{0,1\}$, which is equivalent modulo colouring to the target function $f : \mathcal{G}_n \to \{0,1\}$. That is, for any input graph $G \in \mathcal{G}_n$ and a given colouring $G^{\wedge} \in \mathcal{G}_{n,k}$, $f'(G^{)} = f(G)$.

First, we show that functions $f : \mathcal{G}_n \to \{0,1\}$ can be captured in the individualised coloured graph setting. To this end, we define the *restriction* of a coloured graph G, denoted $G^{\vee}$, as the underlying plain graph, that is, the graph obtained from G by omitting all colours. Conversely, we formally define the *expansion* of a plain graph G, denoted $G^{\wedge}$, as a coloured graph resulting from colouring the nodes of G. Therefore, $G^{\wedge}$ satisfies $G = (G^{\wedge})^{\vee}$. Note that, while the restriction operation is deterministic, the expansion operation is not, as exponentially (but finitely) many colourings with respect to n and k are possible for a given plain graph G.

Based on this idea of transforming input graphs to the individualised coloured graph domain, we map a standard non-coloured input graph $G \in \mathcal{G}_n$ to all its (finitely many) possible individualised colourings $G^{\wedge}$ in $\mathcal{G}_{n,k}$, and analogously map the Boolean target function f over $\mathcal{G}_n$ to an equivalent function f' over $\mathcal{G}_{n,k}$. This is done by considering all individualised expansions $G^{\wedge}$ of G, and mapping these to the same value $f(G)$ through f'. The function f' can thus be defined as a disjunction over all individualised graphs $G^{\wedge} \in \mathcal{G}_{n,k}$ such that $f(G) = 1$, and thus, by Lemma 5.3.3, f' can be written in C^2, as the disjunction over all possible individualised colourings of any $G \in \mathcal{G}_n$ is also finite. We denote the corresponding C^2 disjunction by $\varphi_f^{\wedge}$. Hence, we obtain the following corollary:

Corollary 5.3.1. *Let $f : \mathcal{G}_n \to \{0,1\}$ be an invariant Boolean function. Then there exists a C^2-sentence $\varphi_f^{\wedge}$ (in the language of coloured graphs) such that for all $G^{\wedge} \in \mathcal{G}_{n,k}$ it holds that $[\![\psi_f^{\wedge}]\!](G^{\wedge}) = f(G)$.*

We now show that RNI maps an input graph $G \in \mathcal{G}_n$ to a corresponding coloured expansion $G^{\wedge} \in \mathcal{G}_{n,k}$ with high probability, and thus transforms the problem to learning $\varphi_f^{\wedge}$, which can be learned by an MPNN with global readout, thus yielding universality and bringing us closer to proving Lemma 5.3.1. To this end, we fix $n \geq 1$ and $\epsilon, \delta > 0$. We let

$$c := \left\lceil \frac{2}{\delta} \right\rceil \quad \text{and} \quad k := c^2 \cdot n^3,$$

where $\lceil\ \rceil$ denotes the ceiling operation. Moreover, we assume that we initialise the feature vectors $\boldsymbol{x}_v = (x_{v1}, \ldots, x_{vd})$ of all vertices to $(r_v, 0, \ldots, 0)$, where all r_v for $v \in V$ are chosen independently uniformly at random from $[0, 1]$ and d is the embedding dimensionality. For our MPNN activation function σ, we choose the linearised sigmoid function defined by $\sigma(x) = 0$ for $x < 0$, $\sigma(x) = x$ for $0 \leq x < 1$, and $\sigma(x) = 1$ for $x \geq 1$, analogously to the logical characterisation proof for MPNNs [13]. We now prove the following lemma:

Lemma 5.3.4. *Let $r_1, \ldots, r_n$ be chosen independently uniformly at random from the interval $[0, 1]$. For $1 \leq i \leq n$ and $1 \leq j \leq c \cdot n^2$, let*

$$s_{ij} := k \cdot r_i - (j-1) \cdot \frac{k}{c \cdot n^2}.$$

Then with probability greater than $1 - \delta$, the following conditions are satisfied.

(i) For all $i \in \{1, \ldots, n\}, j \in \{1, \ldots, c \cdot n^2\}$ it holds that $\sigma(s_{ij}) \in \{0, 1\}$.

(ii) For all distinct $i, i' \in \{1, \ldots, n\}$ there exists a $j \in \{1, \ldots, c \cdot n^2\}$ such that $\sigma(s_{ij}) \neq \sigma(s_{i'j})$.

Proof. For every i, let $p_i := \lfloor k \cdot r_i \rfloor$, where $\lfloor\ \rfloor$ denotes the flooring operation. Since $k \cdot r_i$ is uniformly random from the interval $[0, k]$, the integer p_i is uniformly random from $\{0, \ldots, k-1\}$. We are interested in the scenario where $0 < \sigma(s_{ij}) < 1$: Observe that this holds only if $p_i - (j-1) \cdot \frac{k}{c \cdot n^2} = 0$, as this implies that $s_{ij} \in [0, 1]$. Considering that k is divisible by $c \cdot n^2$, this implies that p_i must equal $(j-1) \cdot n$. This is possible with probability $\frac{1}{k}$, as $(j-1) \cdot n$ is an element of $\{0, \ldots, k-1\}$ (since $(j-1)n < c \cdot n^3 < k$). Therefore, we can estimate an upper bound for the probability of *any* s_{ij} yielding intermediate values using the union bound:

$$\Pr\left(\exists i, j :\ 0 < \sigma(s_{ij}) < 1\right) \leq \frac{c \cdot n^3}{k} = \frac{1}{c} \leq \frac{\delta}{2}. \tag{5.8}$$

This is the complement of condition (i) in the lemma, and therefore this condition is satisfied with probability at least $1 - \frac{\delta}{2}$. We now consider condition (ii) and seek to upper bound the probability of failure. Concretely, we let $i, i' \in \{1, \ldots, n\}, i \neq i'$ and suppose that $\sigma(s_{ij}) = \sigma(s_{i'j})$ for all j. This implies that, for all j, $s_{ij} \leq 0 \iff s_{i'j} \leq 0$ due to the linearised sigmoid activation. Therefore, we can apply the floor function to obtain $\lfloor s_{ij} \rfloor \leq 0 \iff \lfloor s_{i'j} \rfloor \leq 0$. Replacing s_{ij} with $k \cdot r_i - (j-1)\frac{k}{c \cdot n^2}$ on both sides and applying the floor function yields:

$$\forall j \in \{1, \ldots, c \cdot n^2\} : \quad p_i \leq (j-1) \cdot \frac{k}{c \cdot n^2} \iff p_{i'} \leq (j-1) \cdot \frac{k}{c \cdot n^2}. \tag{5.9}$$

Notice that $\lfloor k \cdot r_i - (j-1)\frac{k}{c \cdot n^2} \rfloor$ (resp., $k \cdot r_{i'}$) yields the above inequalities as $(j-1)\frac{k}{c \cdot n^2}$ is an integer since $c \cdot n^2$ divides k, whereas $k \cdot r_i$ is not. We now let $j^* \in \{1, \ldots, c \cdot n^2\}$ such that:

$$p_i \in \Big\{(j^* - 1) \cdot \frac{k}{c \cdot n^2}, \ldots, j^* \cdot \frac{k}{c \cdot n^2} - 1\Big\}.$$

In other words, we select the j^* value at the boundary of Equation (5.9) relative to the randomly selected p_i. As we assume that all j satisfy $\sigma(s_{ij}) = \sigma(s_{i'j})$, we can apply Equation (5.9) to obtain:

$$p'_i \in \Big\{(j^* - 1) \cdot \frac{k}{c \cdot n^2}, \ldots, j^* \cdot \frac{k}{c \cdot n^2} - 1\Big\}.$$

First, we note that $p_{i'}$ is independent of p_i, and thus of j^*, as they are based on independent random variables $r_{i'}$ and r_i respectively. Therefore, the probability of getting $\sigma(s_{ij}) = \sigma(s_{i'j})$ for all j, and thus violating condition (ii) corresponds to sampling the same value for p_i and p'_i relative to all possible j^*. Recall that both p_i and p'_i are uniformly distributed. Thus, the probability of violating condition (ii) at a given pair i, i' for any possible j^* is at most $\frac{1}{k} \cdot \frac{k}{c \cdot n^2} = \frac{1}{c \cdot n^2}$. This proves that for all distinct i, i', the probability that $\sigma(s_{ij}) = \sigma(s_{i'j})$ is at most $\frac{1}{c \cdot n^2}$. Hence, we again can apply the union bound to upper-bound the probability across all i, i' pairs:

$$\Pr\left(\exists i \neq i' \forall j : \ \sigma(s_{ij}) = \sigma(s_{i'j})\right) \leq \frac{1}{c}. \tag{5.10}$$

Finally, Equation (5.8) and C5.9 together imply that the probability that either condition (i) or (ii) is violated is at most

$$\frac{1}{c} + \frac{1}{c} = \frac{2}{c} \leq \delta,$$

as required. □

Using all earlier lemmas, we can now complete the proof of Lemma 5.3.1.

Proof of Lemma 5.3.1. For a given function $f : \mathcal{G}_n \to \{0, 1\}$ mapping uncoloured graphs to a Boolean value, we use the sentence $\psi_f^\wedge$ according to Corollary 5.3.1, so as to produce analogous outputs to f for all possible colouring expansions $G^\wedge \in \mathcal{G}_{n,k}$ of $G \in \mathcal{G}_n$. Applying Lemma 5.2.1 (the logical characterisation) to this sentence, we obtain an MPNN $\mathcal{N}_f$ that computes, for an input expansion coloured graph $G^\wedge \in \mathcal{G}_{n,k}$, an ϵ-approximation of $f(G)$, i.e., the target output over the original graph.

Without loss of generality, we assume that the vertex set of the input graph to our MPNN is $\{1, \ldots, n\}$. We now choose the embedding dimensionality d such that

$d \geq c \cdot n^2$ and d is at least as large as the dimension l of the state vectors of $\mathcal{N}_f$ (based on the number of sub-formulas in $\psi_f^\wedge$, as in the construction of the MPNN from the characterisation proof [13]). We then randomly initialise state vectors for all nodes in the input graph as $\boldsymbol{x}_i^{(0)} = (r_i, 0, \ldots, 0)$ for values r_i chosen independently and uniformly at random from the interval $[0, 1]$.

As a first step, our MPNN computes a purely local update (no messages need to be passed) that maps $\boldsymbol{x}_i^{(0)}$ to $\boldsymbol{x}_i^{(1)} = (x_{i1}^{(1)}, \ldots, x_{id}^{(1)})$ with

$$x_{ij}^{(1)} = \begin{cases} \sigma\Big(k \cdot r_i - (j-1) \cdot \frac{k}{c \cdot n^2}\Big) & \text{for } 1 \leq j \leq c \cdot n^2, \\ 0 & \text{for } c \cdot n^2 + 1 \leq j \leq d. \end{cases}$$

Since we treat k, c, n as constants, the mapping

$$r_i \mapsto k \cdot r_i - (j-1) \cdot \frac{k}{c \cdot n^2}$$

is just a linear mapping applied to $r_i = x_{i1}^{(0)}$.

By Lemma 5.3.4, with probability at least $1 - \delta$, the vectors $\boldsymbol{x}_i^{(1)}$ are mutually different $\{0, 1\}$-vectors, which we can view as representing a colouring of all input graph nodes with colours from the predicate set $R_1, \ldots, R_k$. We refer to this resulting coloured graph as $G^\wedge$. Since the vectors $\boldsymbol{x}_i^{(0)}$ are mutually distinct, $G^\wedge$ is an individualised coloured graph, and thus belongs to the class $\mathcal{G}_{n,k}$. We can now thus apply the MPNN $\mathcal{N}_f$ to $G^\wedge$ and compute a value ϵ-close to

$$[\![\psi_f^\wedge]\!](G^\wedge) = f((G^\wedge)^\vee) = f(G),$$

with probability at least $1 - \delta$, thus completing the proof. □

Having now proven Lemma 5.3.1, we can now extend the result to the real-valued setting and prove Theorem 5.3.1.

Proof of Theorem 5.3.1. Let $f : \mathcal{G}_n \to \mathbb{R}$ be an invariant, real-valued function. Since the input domain $\mathcal{G}_n$ is finite, the output range $Y := f(\mathcal{G}_n)$ is also finite. More precisely, we have $N := |Y| \leq |\mathcal{G}_n|$. Without loss of generality, we write the output set Y as $Y = \{y_1, \ldots, y_N\}$. Then, for $i = \{1, \ldots, N\}$, we define a Boolean sub-function $g_i : \mathcal{G}_n \to \{0, 1\}$ as:

$$g_i(G) = \begin{cases} 1 & \text{if } f(G) = y_i, \\ 0 & \text{otherwise.} \end{cases}$$

Note that all Boolean functions g_i are invariant by design, and these functions are linearly combined to yield the original f. More specifically, f can be written as:

$$f(G) = \sum_{i=1}^{N} y_i \cdot g_i(G).$$

Let $\epsilon, \delta > 0$ be our target error and confidence thresholds respectively for the real-valued target function f. To achieve an (ϵ, δ) approximation of f, we must approximate each g_i with error $\epsilon' = \frac{\epsilon}{\max Y^+}$, where Y^+ denotes the absolute value set $\{|y_1|, \ldots, |y_N|\}$, so that linear multiplication by y_i recovers the error ϵ. In terms of confidence, we set $\delta' = \frac{\delta}{N}$ so that the overall error probability across all Y values is limited to δ by the union bound.

By Lemma 5.3.1, for every $i \in \{1, \ldots, N\}$ there exists an MPNN (with global readout) with RNI $\mathcal{N}_i$ that (ϵ', δ')-approximates each g_i. These MPNNs can be put together, e.g., through concatenation, to obtain an invariant MPNN $\mathcal{N}$ that computes a function $g : \mathcal{G}_n \to \{0, 1\}^N$. We then only need to apply the linear transformation

$$\boldsymbol{x} \mapsto \sum_{i=1}^{N} x_i \cdot y_i$$

to the N outputs of $\mathcal{N}$ to yield an (ϵ, δ) approximation of f. □

5.3.2 Discussion

The implications of Theorem 5.3.1 can be summarised as follows. First, our result shows that MPNNs with RNI can distinguish individual graphs with an embedding dimensionality polynomial in n and $\frac{1}{\delta}$, namely, $O(n^2\delta^{-1})$. Second, our universality result does not require fully randomised node features, but only needs a single randomised dimension. Hence, the theorem also allows for the study of *partial* RNI, where a fraction of node features is randomised, leading to reduced noise introduction and better preservation of input features.

In terms of size, our construction yields MPNNs with an exponentially large embedding dimensionality. This is especially true for the final step mapping from $|\mathcal{G}_n|$ Boolean functions to a single arbitrary real-valued function. Nonetheless, we doubt that much more compact models can be defined, as we make no assumption about the function f. However, the approximation of Boolean functions using MPNNs is more delicate and interesting. Although it may still be the case that MPNNs need a dimensionality exponentially large in n, particularly for highly complicated target sentences, the construction itself, following from the logical characterisation of MPNNs

[13], is very adaptive and its size is tightly linked to the descriptive complexity of the function we want to approximate. That is, for a more restricted class of functions, there may be more efficient constructions than the disjunction of individualised graph sentences, and our proof does not rely on a particular construction.

Unlike other universality results for GNNs, e.g, higher-order invariant/equivariant networks [88], our construction does not use intractable higher-order tensors defined on tuples of nodes. Instead, the complexity of our construction is entirely delegated to embedding dimensionality, which can be treated as a hyper-parameter in practice. Moreover, the required embedding dimensionality is usually very small to achieve strong performance on most real-world applications. Instead, our construction provides a logical characterisation for MPNNs with RNI, and theoretically substantiates how randomisation improves expressiveness. This construction therefore also enables a more logically grounded theoretical study of randomised MPNN models, based on particular architectural or parametric choices.

Finally, it is interesting to think about the value of using RNI, when its main contribution in our proof is to produce identifying node features. Put differently, one can ask the question: Why not simply use explicitly defined unique node features or sample randomly from a fixed set of discrete features to reduce the uncertainty δ stemming from RNI? Indeed, several works have proposed MPNN extensions to introduce auxiliary identifying node features [107], yielding universality. However, these models are problematic in practice, as adding deterministic features typically hinders model generalisation, as models simply learn to fit these uninformative features. Furthermore, random pre-set (discrete) colour features have also been used to disambiguate between nodes [45]. This approach, known as CLIP, introduces randomness to node representations, and explicitly makes graphs distinguishable, thus yielding universality by construction. However, this approach is designed a priori to distinguish nodes, and explicitly introduces an arbitrary fixed structure, which makes it vulnerable to overfitting. Therefore, RNI is a promising approach for jointly leveraging the power of identifying features and maintaining model generalisation, and this has motivated RNI-based approaches like rGNN [148]. Within rGNN, an MPNN trains and runs with partially randomised initial (continuous) node features. However, rGNNs are only shown near-optimally approximate specific combinatorial optimisation problems, and can distinguish between 1-WL indistinguishable graph pairs based on fixed local substructures, leaving the precise impact of RNI on GNNs for learning arbitrary functions over graphs *unknown*.

All in all, our theorem considers the more challenging RNI setting, where no explicit structure is introduced to node features, and improves on Theorem 1 of Sato et al. [148] by showing MPNN *universality* with RNI, based on an adaptive logical construction. Therefore, our work shows that arbitrary real-valued functions over graphs can be learned by MPNNs with RNI, and also links the required depth and embedding dimensionality of the MPNN with the descriptive complexity of the target function.

5.4 Datasets for Expressiveness Evaluation

Standard evaluation protocols for GNNs typically use real-world benchmarks [89], based on e.g., molecular graphs, social networks. However, these benchmarks are not suited for GNN evaluation from the expressive of expressive power, as they usually do not contain instances indistinguishable by 1-WL. In fact, higher-order models, which have expressiveness strictly higher than 1-WL, only marginally outperform MPNNs, if at all, on these datasets [53], which further highlights their unsuitability for expressiveness evaluation. Thus, we propose the synthetic datasets EXP and CEXP, to explicitly evaluate GNN expressiveness. The EXP dataset consists of graph instances $\{G_1, \ldots, G_n, H_1, \ldots, H_n\}$, which each encode a propositional formula, and the classification objective, yielding the label for each graph, is to determine whether the underlying propositional formula represented by each graph is satisfiable (SAT). Fundamentally, each graph pair (G_i, H_i) in EXP satisfies the following properties:

1. G_i and H_i are non-isomorphic.
2. G_i and H_i yield different SAT outcomes. More precisely, G_i encodes an unsatisfiable propositional formula, and H_i encodes a satisfiable formula,
3. G_i and H_i are 1-WL indistinguishable, so are *guaranteed* to be classified in the same way, i.e., both True or both False, by standard MPNNs, and
4. G_i and H_i are 2-WL distinguishable, so *can* be classified differently by higher-order GNNs.

Building on EXP, CEXP additionally introduces instances not requiring expressiveness beyond 1-WL by corrupting standard EXP graph pairs. Specifically, CEXP is based on a standard EXP dataset, but modifies 50% of its graph pairs such that the satisfiable graph G_i is made 1-WL distinguishable from its unsatisfiable counterpart H_i using a small number of added edges. Hence, CEXP consists of 50% "corrupted"

data, distinguishable by MPNNs and 1-WL, which we refer to as CORRUPT, and 50% standard EXP data, requiring expressive power beyond 1-WL, referred to as $\overline{\text{EXP}}$. Thus, CEXP contains the same graph structures as EXP, but maps these to different output values in $\overline{\text{EXP}}$ and CORRUPT, which makes the overall learning task on both CEXP subsets more challenging than learning $\overline{\text{EXP}}$ or CORRUPT in isolation.

We now present the detailed generation procedure for both datasets.

5.4.1 Construction of EXP

To generate the graph pairs of EXP, we build on two main components:

1. A *core pair* of graphs, which are non-isomorphic, planar[1], 1-WL indistinguishable, 2-WL distinguishable, and decide the satisfiability of every instance. That is, the core pair encodes the propositional components that yield satisfiability and unsatisfiability, respectively.

2. An additional randomly generated and satisfiable *planar component*, identically added to both graphs in the aforementioned core pair, to add variability to EXP and make learning more challenging.

We therefore present both components, and then provide further details about graph encoding and planar embeddings.

Core pair. In EXP, a core pair consists of two formulas in conjunctive normal form (CNF) φ_1, φ_2 with $2n$ variables each, $n \in \mathbb{N}^+$, such that φ_1 is unsatisfiable and φ_2 is satisfiable, and such that their graph representations are 1-WL indistinguishable and planar. φ_1 and φ_2 are constructed using two structures which we refer to as *variable chains* and *variable bridges* respectively.

A *variable chain* φ_{chain} is defined over a set of $n \geq 2$ Boolean variables, and imposes that all variables be identically set to the same Boolean value for the chain to be satisfied. More specifically, given $x_i, ..., x_j$, the variable chain can be defined in increasing or decreasing order over these variables, yielding the two formulas:

$$\text{Chain}_{\text{Inc[i,j]}} = \bigwedge_{k=i}^{j-1} (\bar{x_k} \vee x_{i+(k+1)\%(j-i+1)}), \text{ and} \tag{5.11}$$

$$\text{Chain}_{\text{Dec[i,j]}} = \bigwedge_{k=i}^{j-1} (x_k \vee \bar{x}_{i+(k+1)\%(j-i+1)}). \tag{5.12}$$

[1]A main motivation behind using planar graphs is a recent result [90], showing that 3-WL is sufficient to distinguish any pair of planar graphs. Hence, we start from planar graphs, and apply further constraints to yield our desired 2-WL boundaries. Beyond this result, planar graph instances are easy to visualise and depict, and thus are a good choice for a benchmark like EXP.

Furthermore, a *variable bridge* is defined on an even number of variables $x_0, ..., x_{2n-1}$ using the formula:

$$\text{Bridge}_{[n]} = \bigwedge_{i=0}^{n-1} \big((x_i \vee x_{2n-1-i}) \wedge (\bar{x}_i \vee \bar{x}_{2n-1-i})\big). \tag{5.13}$$

A variable bridge constrains the variables in its conjuncts, e.g., x_0 and x_{2n-1}, to have opposite truth assignments for the bridge to be satisfied. That is, in our earlier example, the bridge can only be satisfied if $x_0 = \bar{x}_{2n-1}$.

Using variable chains and bridges, we define φ_1 and φ_2. More specifically, we define φ_1 as a variable chain and bridge on variables $x_0, \ldots, x_{2n-1}$, yielding contrasting and unsatisfiable constraints. To define φ_2, we "cut" the chain at x_n, replacing it with two chains of size n, such that the first n variables can differ in truth value from the latter n, and thus allowing the bridge constraints to be satisfied. To maintain planarity, we use a decrementing chain on variables $x_n, \ldots, x_{2n-1}$. Hence, φ_1 and φ_2, relative to $2n$ propositional variables, can be written as:

$$\varphi_1 = \text{Chain}_{\text{Inc}[0,\, 2n]} \wedge \text{Bridge}_{[2n]}, \text{ and} \tag{5.14}$$

$$\varphi_2 = \text{Chain}_{\text{Inc}[0,\, n]} \wedge \text{Chain}_{\text{Dec}[n,\, 2n]} \wedge \text{Bridge}_{[2n]}. \tag{5.15}$$

Planar component. Building on φ_1 and φ_2, a disjoint satisfiable planar graph component φ_{planar} is added. The planar component φ_{planar} shares no variables or disjunctions with the core pairs, and is introduced to create noise and make learning more challenging. We generate φ_{planar} as a random biconnected and bipartite planar graph using Plantri [26], and then map this to a propositional formula as follows:

1. The larger set of nodes in the bipartite graph is selected as the variable set[2].
2. Highly-connected disjunctions, where the threshold is 5 disjuncts, are randomly split in a planarity-preserving fashion (using the planar embedding of the generated planar component).
3. Literal signs for variables are assigned uniformly at random.
4. Redundant disjunctions, i.e., disjunctions over the same literals, are removed to obtain φ_{planar}.
5. If φ_{planar} is satisfiable, then generation is successful. Otherwise, the formula is discarded and a new φ_{planar} is analogously generated from Step 1.

[2]Ties are broken arbitrarily if the two sets are equally sized.

Since the core pair and φ_{planar} are disjoint, it clearly follows that the graph encodings of $\Phi_1 = \varphi_{\text{planar}} \wedge \varphi_1$ and $\Phi_2 = \varphi_{\text{planar}} \wedge \varphi_2$, namely the disjoint union of both conjunct graph representations in both conjunctions, are planar and 1-WL indistinguishable. Furthermore, Φ_1 is unsatisfiable, and Φ_2 is satisfiable. Hence, the introduction of φ_{planar} maintains all the desirable core properties, all while making any generated EXP dataset more challenging.

The structural properties of the cores, combined with the combinatorial difficulty of SAT, make EXP a challenging dataset. For example, even minor formula changes, such as flipping a literal, can lead to a change in the SAT outcome, which enables the creation of near-identical, yet semantically different instances. Moreover, SAT is NP-complete [42], and remains so on planar instances [79]. Hence, EXP is cast to be challenging, both from an expressiveness and computational perspective.

Remark. Intuitively, Φ_1 and Φ_2 can be distinguished by 2-WL, as 2-WL can detect the break in cycles resulting from the "cut" in the variable chain. In other words, 2-WL can identify that the chain has been broken when comparing φ_1 and φ_2, and thus can return distinct node hashes, and therefore, distinct graph outputs.

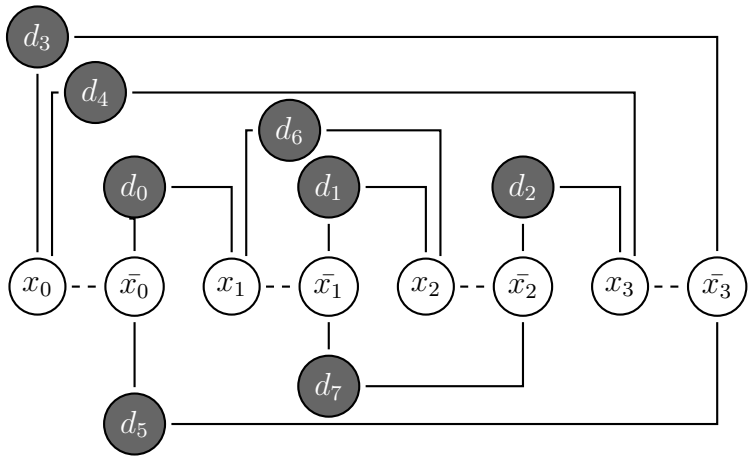
Representing propositional formulas as graphs. To encode Φ_1 and Φ_2 as graphs, we use the following graph encoding, denoted by *Enc*:

1. Every variable is encoded by two nodes, representing its positive and negative literals, and both literals are connected by an edge.

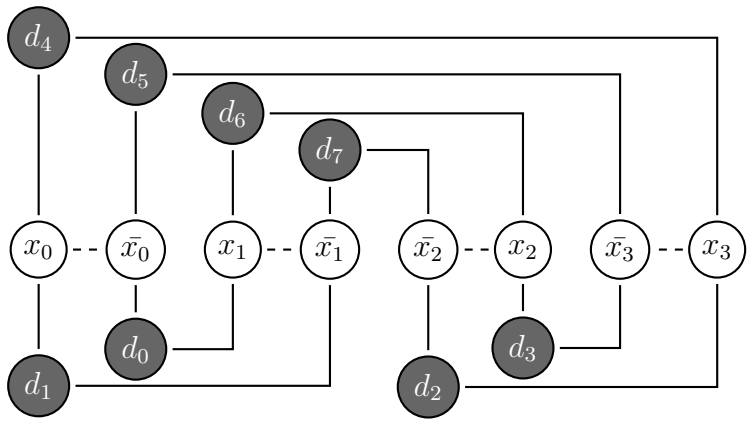
2. Every disjunction is represented by a node, and these disjunction nodes are connected by an edge to literal nodes if the literal appears in the corresponding disjunction.

3. Literal and disjunction nodes are encoded with their respective types, i.e., colours. We encode types into the graph to maintain the semantic distinction and align our encoding with those commonly used in the literature [152].

Planar embeddings for core pair. To better illustrate the core pair, we show planar embeddings for $Enc(\varphi_1)$ and $Enc(\varphi_2)$ for $n = 2$ in Figure 5.1. Note that $Enc(\varphi_1)$ and $Enc(\varphi_2)$ can also be shown to be 1-WL indistinguishable. Intuitively, we note that the node neighbourhoods for both graphs are identical and regular, both for disjunction and literal nodes. Indeed, all literal nodes are connected to exactly one other literal node and two disjunction nodes, and all disjunction nodes are connected to exactly two literal nodes.



(a) The encoding of the formula φ_1.



(b) The encoding of the formula φ_2.

Figure 5.1: Illustration of planar embeddings for the core pair formulas φ_1 and φ_2 for $n = 2$. Nodes labelled d_i indicate disjunctions in the original formulas.

5.4.2 Construction of CExp

Given a generated Exp dataset with N pairs of graphs, we create CExp by selecting $\frac{N}{2}$ graph pairs uniformly at random and modify these to yield Corrupt. Hence, the remaining $\frac{N}{2}$ unselected graph pairs are identically generated as standard Exp graph pairs, and we refer to these instances within CExp as $\overline{\text{Exp}}$.

For a selected graph pair in Corrupt, we discard the satisfiable graph and construct a new graph, starting from a copy of the unsatisfiable graph, as follows:

1. We randomly introduce new (non-redundant) literals to existing disjunctions (with less than 5 disjuncts) of the unsatisfiable graph , until both a) 3 literals have been added *and* b) the formula becomes satisfiable. On the graph representation level, this can be seen as creating new edges in the graph encoding

between the relevant disjunction node and the added literal nodes.

2. Once a satisfiable formula is reached from Step 1, we sequentially revisit all added edges and drop any edge whose removal does not restore unsatisfiability. This step ensures that a minimal number of new edges, relative to the original unsatisfiable graph, is added.

Within Step 1, we perform edge addition in a sequential manner: We first sample uniformly at random from the set of disjunctions with less than 5 literals, and then uniformly randomly sample a literal other than existing literals and their negations.

Observe that this corruption process achieves multiple challenging objectives within CEXP. First, it preserves the existing unsatisfiable core φ_1, and provides a new formula based on φ_1 which is satisfiable. Therefore, identifying the φ_1 structure on CORRUPT is no longer sufficient to achieve good performance. Second, the corruption introduces significant variability to CEXP, as the planar component and cores can now share edges, and could no longer be disjoint. Finally, corruption achieves the main goal of making graph pairs 1-WL distinguishable, unlike in EXP. Hence, CORRUPT graph pairs provide standard MPNNs with an opportunity to perform strongly, despite their inherent 1-WL limitation on the remaining instances in $\overline{\text{EXP}}$.

5.5 Experimental Evaluation

In this section, we evaluate the effect of RNI on MPNN expressiveness using EXP and CEXP, and compare against established higher-order GNNs. In particular, we are interested in studying (i) whether RNI improves MPNN expressiveness in practice, and (ii) how RNI affects learning in general, including on simpler data, as is partly the case in CEXP. We first describe the dataset setup for EXP and CEXP.

Data generation. We generate an EXP dataset of 600 core pairs, where n (cf. Section 5.4.1) is uniformly randomly set between 2 and 4 inclusive. For the planar component, we generate 600 graphs, such that 500 φ_{planar} formulas are generated from 12-node planar bipartite planar graphs, and the remaining 100 from planar bipartite graphs with 15 nodes. Hence, the formulas in EXP have a number of variables ranging from 10 (4 core variables when $n = 2$ plus a minimum 6 variables from the 12-node planar components) to 22 (8 core variables for $n = 4$ plus a maximally-sized variable subset of 14 nodes for the planar component), and a number of disjunctions from 10 (8 core disjunctions for $n = 2$ plus a minimum 2 disjunctions from φ_{planar}) to 30 (16 core disjunctions for $n = 4$ plus at most 14 from φ_{planar}). Finally, we generate CEXP

independently, based on another 600 EXP graph pairs, and corrupt 300 of these to obtain the CORRUPT subset.

Experimental setup. We experiment with the following models:

- **1-WL GCN (1-GCN):** A graph convolutional network (GCN) consisting of 8 distinct, i.e., heterogeneous, message passing iterations, each followed by an exponential linear unit (ELU) activation function, [41], 64-dimensional embeddings, and deterministic learnable initial node embeddings for both disjunction and literal features. This is a standard MPNN baseline, so is limited by 1-WL in terms of expressive power. Therefore, 1-GCN is guaranteed to achieve 50% accuracy on EXP.

- **GCN - Random node initialisation (GCN-RNI):** A GCN enhanced with RNI. We evaluate this model with four initialisation distributions, namely, the standard normal distribution (mean 0, variance 1) (N), the uniform distribution over $[-1, 1]$ (U), Xavier normal (XN), and the Xavier uniform distribution (XU) [63]. We denote the respective models GCN-RNI(D), where $D \in \{\mathrm{N}, \mathrm{U}, \mathrm{XN}, \mathrm{XU}\}$.

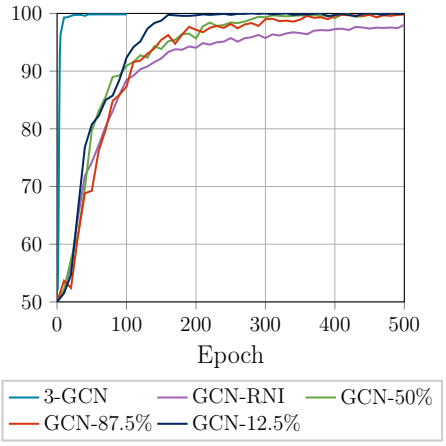
- **GCN - Partial RNI (GCN-x%RNI):** A GCN-RNI model, where $\lfloor \frac{64x}{100} \rfloor$ dimensions are initially randomised, and all remaining dimensions are set deterministically for literal and disjunction types. We set x to the extreme values 0 and 100%, 50%, as well as near-edge cases of 87.5% and 12.5%, respectively.

- **PPGN:** A higher-order GNN with 2-WL expressive power [114]. We set up the provably powerful graph network (PPGN) using its original implementation, and use its default configuration of eight 400-dimensional computational blocks.

- **1-2-3-GCN-L:** A higher-order GNN [127] emulating 2-WL on 3-node tuples. 1-2-3-GCN-L operates at increasingly coarse granularity, starting with single nodes, as in 1-WL, and rising to 3-tuples. This model uses a *connected* relaxation of 2-WL, which slightly reduces space requirements as it discards a significant proportion of all possible 3-tuples. However, this comes at the cost of some theoretical guarantees. We set up 1-2-3-GCN-L with 64-dimensional embeddings, 3 message passing iterations at level 1, 2 at level 2, and 8 at level 3.

Model	Test Accuracy (%)
GCN-RNI(U)	97.3 ± 2.55
GCN-RNI(N)	**98.0 ± 1.85**
GCN-RNI(XU)	97.0 ± 1.43
GCN-RNI(XN)	96.6 ± 2.20
PPGN	50.0
1-2-3-GCN-L	50.0
3-GCN	**99.7 ± 0.004**

(a) Accuracy results on EXP.



(b) Learning curves on EXP.

Figure 5.2: Accuracy results (a) and learning curves (b) on the EXP dataset for GCN-RNI and competing baseline models.

- **3-GCN:** A GCN analog of the *full* 2-WL procedure over 3-node tuples, i.e., 8 message passing layers, 64-dimensional embeddings. This models preserves all 2-WL theoretical guarantees.

Further details about hyper-parameter tuning for all models can be found in the appendix of this thesis.

5.5.1 How Does RNI Improve Expressiveness?

In this experiment, we evaluate GCN-RNI with different RNI settings on EXP, and compare with standard MPNNs and higher-order models. We evaluate all baseline models on EXP using 10-fold cross-validation. Due to computational limitations, we train the more computationally demanding 3-GCN for 100 epochs per fold, whereas we train all other systems for 500 epochs. Finally, we report *mean test accuracy* across all folds as our target metric.

Results. Accuracy results for all models are reported in Table 5.2a, and learning curves, showing the evolution of model performance over training, for 3-GCN and all GCN-RNI models are shown in Figure 5.2b. In keeping with Theorem 5.3.1, GCN-RNI achieves near-perfect performance on EXP, substantially surpassing the 50% threshold due to 1-WL. Indeed, GCN-RNI models achieve above 95% accuracy with all four initial distributions N, U, XN and XU. These findings confirm concurrent empirical evidence from rGNNs [148], and suggest that RNI yields expressiveness improvements beyond the limited scope of fixed sub-structure detection. However,

we must note that RNI, though enabling universality in theory, is delicate when included as part of models in practice. In particular, we observed that GCN-RNI is highly sensitive to changes in learning rate, activation function, and/or randomisation distribution in our experiments, and had to perform careful tuning to reach a setup showcasing the full potential of RNI. To demonstrate this, we provide substantially worse results (except with normal distribution RNI) for GCN-RNI where ELU is replaced by the hyperbolic tangent activation in Table 5.3.

Table 5.3: Test Accuracy results of GCN-RNI models on EXP using the hyperbolic tangent (*tanh*) activation function.

Model	Testing Accuracy (%)
GCN-RNI(U)	92.7 ± 5.61
GCN-RNI(N)	**96.0 ± 2.11**
GCN-RNI(XU)	64.6 ± 19.9
GCN-RNI(XN)	63.0 ± 20.9

Aside from standard MPNNs, PPGN, a higher-order model, does not break the 50% performance barrier, despite theoretically being 2-WL expressive. In fact, this result stands even with extensive training. An explanation for this unexpectedly poor performance could lie in the theoretical underpinning of PPGN: Essentially, PPGN learns an approximation of 2-WL, based on power-sum multi-symmetric polynomials (PMP), which are very carefully set in the derivation of the PPGN expressiveness result. Furthermore, PPGN has recently been shown to require exponentially many data samples in the size of the input graph domain [139] for learning. These factors together suggest that PPGN struggles to learn the required PMPs on EXP, and this behaviour does not change, neither on the training and testing sets, across any of the extensive hyperparameter tuning setups we attempted on the official implementation. Hence, PPGN is likely struggling to discern between EXP graph pairs due to the smaller sample size and variability of the dataset.

Parallel to PPGN, 1-2-3-GCN-L, the 2-WL-inspired model with connected relaxation among node 3-tuples, also fails to surpass 50% accuracy. This, however, can be fully attributed to the aforementioned relaxation. Indeed, only considering 3-tuples of nodes that form a connected sub-graph ignores disconnected 3-tuples by definition, and the latter tuples are where the difference between EXP core pair graphs lies. To see this, consider the variable chain structure discussed in Section 5.4.1, which is "cut" between the unsatisfiable and satisfiable pair in the core pair. To detect this "cut", one naturally must compare the connected 3-tuple in the full chain with its now disconnected analog in the separated chains. However, this is not possible in 1-2-3-GCN-L. This further highlights the difficulty of EXP, as even relaxing 2-WL

reduces the model to random performance. By contrast, 3-GCN achieves near-perfect performance, as it explicitly has the necessary theoretical power to solve Exp.

In terms of model convergence, we observe that 3-GCN converges significantly faster than GCN-RNI models at all randomisation percentages. Indeed, 3-GCN converges around an accuracy of approximately 99% after only 10 epochs, whereas GCN-RNI models require over 100 epochs. Intuitively, this slower convergence of GCN-RNI can be partly explained by the more difficult nature of the learning task for GCN-RNI compared to 3-GCN: Whereas 3-GCN learns from deterministic embeddings, and can naturally discern between dataset cores, GCN-RNI relies on RNI-induced node features to make distinctions between Exp graph pairs. Moreover, it must learn to do this *in expectation*, and with the highest possible probability, i.e., it must learn a robust function that correctly maps Exp graph pairs irrespective of specific randomisations. Hence, GCN-RNI must build on RNI to detect overall graph structure implicitly, and simultaneously learn a robust function that overcomes RNI variability. Therefore, the learning task in the RNI setting, though feasible, is practically challenging and requires more training to converge.

All in all, our findings indicate that RNI indeed improves the expressive power of MPNNs, and even makes them competitive with higher-order models at a fraction of the computational cost. To be precise, both 3-GCN and GCN-RNI achieve accuracies of above 95% on Exp, where graphs include no more than 70 nodes. Yet, for, e.g., a 50-node graph in Exp, GCN-RNI only requires 3200 node parameters/features (64-dimensional embeddings), whereas 3-GCN requires 1,254,400 node parameters. Therefore, GCN-RNI, unlike 3-GCN, can easily scale to larger instances. However, this increase in expressive power comes at the expense of convergence, as models must train for longer periods to reach strong performance levels.

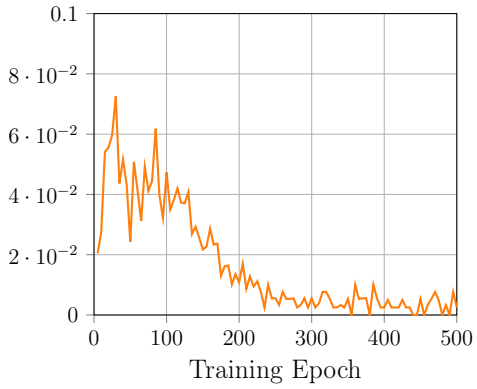


Figure 5.3: Average standard deviation of test accuracy over all 10 validation splits for GCN-50%RNI on Exp.

To more formally study this convergence and quantify the variability of GCN-RNI, we also measured the standard deviation across the 10 splits for test accuracy for the semi-randomised GCN-50%RNI model while training on Exp, and show this evolution in Figure 5.3. This figure shows that standard deviation spikes sharply

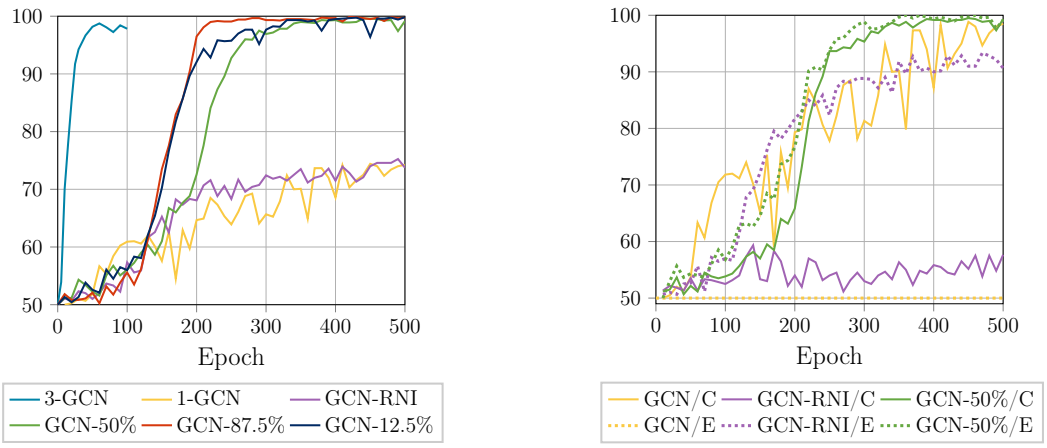
at the start of training, and only begins dropping after 100 epochs. This suggests that the learning behaviour of GCN-50% RNI is highly variable across splits, i.e., convergence speed and start differ substantially across runs. This phenomenon only subsides once sufficiently many training epochs have passed, so as to allow all training loops to achieve near-perfect test performance with high probability. Hence, RNI introduces volatility to GCN training, and this manifests in variable convergence rates across the different splits of EXP. Nonetheless, this volatility does not hinder performance, as all folds eventually reach satisfactory performance within a reasonable amount of epochs, and subsequently stabilise.

5.5.2 How Does RNI Behave on Variable Data?

We have shown that RNI practically improves the expressive power of MPNNs and enables them to achieve near-perfect performance on EXP. However, this empirical result solely demonstrates the impact of RNI from an expressiveness perspective, but does not address other practical concerns about learning functions over graphs with RNI. In particular, multiple questions remain open: How does RNI impact learning when data contains instances with varying expressiveness requirements, e.g., 1-WL distinguishable and 1-WL indistinguishable data, and how does RNI affect model generalisation on these more variable datasets? To address these questions, we run analogous experiments on the richer, more variable CEXP dataset.

At a high level, CEXP is well-suited for holistically evaluating the efficacy of RNI on more variable data, as it evaluates the expressiveness contribution of RNI on $\overline{\text{EXP}}$, but in a joint setting with a second, highly different learning task on CORRUPT involving very similar core structures. Moreover, CEXP enables a direct assessment of the effect of different randomisation proportions on overall and subset-specific, i.e., CORRUPT, $\overline{\text{EXP}}$, model performance. In particular, randomisation is not exclusively beneficial on CEXP, as random features introduce noise that can hurt performance on CORRUPT.

In this experiment, we train GCN-RNI with varying randomisation degrees, as well as 3-GCN, the only strongly performing higher-order GNN from the earlier experiment, on CEXP. Across all GCN-RNI models, we only use the normal distribution for RNI, given its strong performance in the earlier experiment. The learning curves of all GCN-RNI and 3-GCN on CEXP are shown in Figure 5.4a, whereas the same curves for the $\overline{\text{EXP}}$ and CORRUPT subsets are shown in Figure 5.4b. As on EXP, 3-GCN converges very quickly, exceeding 90% test accuracy within 25 epochs on CEXP. By contrast, GCN-RNI, for all randomisation levels, converges much slower, at around



(a) CEXP. (b) $\overline{\text{EXP}}$ (/E) and CORRUPT (/C).

Figure 5.4: Learning curves on the CEXP dataset, as well as its subsets CORRUPT and $\overline{\text{EXP}}$, for GCN-RNI and competing baseline models.

200 epochs, despite the small size of input graphs (∼70 nodes at most). Furthermore, fully randomised GCN-RNI performs worse than partly randomised GCN-RNI on CEXP due to its weak performance on CORRUPT. Across the board, partial randomisation significantly improves performance relative to full randomisation. This can clearly be seen with GCN-12.5%RNI and GCN-87.5%RNI, which by far outperform GCN-RNI. Interestingly, this improvement with partial randomisation does not occur in the previous experiment on EXP.

Looking at CEXP in more detail, we can infer that partial randomisation helps MPNNs far more, as it offers a better inductive bias combining both random features and deterministic features. More concretely, GCN-12.5%RNI has mostly deterministic node features, which greatly simplifies learning over CORRUPT, and includes random features to help on $\overline{\text{EXP}}$. Interestingly, GCN-87.5%RNI, where only 8 embedding dimensions are deterministic, still performs strongly on CORRUPT, which suggests that even this small number of deterministic dimensions is sufficient for learning on this subset. By contrast, fully randomised GCN-RNI loses all node type information, which is key for CORRUPT, and therefore fails to achieve even 60% accuracy on this subset. This model even relatively struggles on $\overline{\text{EXP}}$, only reaching 91% accuracy, and converges slower. Hence, GCN-12.5%RNI and GCN-87.5%RNI effectively achieve a "best of both worlds" on CEXP, leveraging inductive bias from deterministic node embeddings to succeed on CORRUPT while harnessing the power of RNI to perform strongly on $\overline{\text{EXP}}$. This behaviour is best shown in Figure 5.4b, where it is clear that standard GCN fails to learn $\overline{\text{EXP}}$, fully randomised GCN-RNI

struggles to learn CORRUPT, and the semi-randomised GCN-50%RNI achieves perfect performance on both subsets.

In terms of convergence, all GCN-RNI models, at all randomisation levels, converge significantly slower than 3-GCN. Moreover, an interesting phenomenon occurs in the first 100 epochs of training, where all GCN-RNI models fluctuate around 55% accuracy, before ultimately recovering and reaching better performance levels. This fluctuation is very interesting, and suggests a "struggle" by all models to reconcile fitting $\overline{\text{EXP}}$ (expressiveness) with fitting CORRUPT (based on features), particularly given the variability of RNI, which makes separating the two objectives especially challenging. Indeed, models must decide on how to use random features, and eventually learn to (i) ignore them to achieve good accuracy on CORRUPT, and (ii) use them effectively to surpass the 1-WL limit and perform well on $\overline{\text{EXP}}$. This fluctuation is therefore inherent to RNI, and naturally does not affect higher-order deterministic models such as 3-GCN. Overall, these findings consolidate the earlier observations made on EXP regarding convergence, and highlight that the variability and convergence speed for RNI-based models also hinges on the complexity of the input dataset.

Surprisingly, 1-GCN struggles to learn on CORRUPT, despite it theoretically being the best-suited model for the task. In fact, this model converges slower than partially randomised models, but does not exhibit the aforementioned training "fluctuation". In parallel, 1-GCN only achieves 50% accuracy on $\overline{\text{EXP}}$, as expected. This performance suggests an interesting training dynamic with 1-GCN, where $\overline{\text{EXP}}$ instances, which themselves cannot be distinguished by 1-GCN, hinder training on the learnable CORRUPT subset. More specifically, the identical generation of unsatisfiable instances across both subsets, as well as the minimal difference between the satisfiable graph and the unsatisfiable graph in CORRUPT graph pairs, make the learning objectives highly intertwined, and offer the model little means (without RNI) to differentiate between the two subsets, particularly as it provably cannot fit the $\overline{\text{EXP}}$ graph pairs. Hence, 1-GCN, in attempting to fit graphs from both subsets, struggles to learn the simpler difference in CORRUPT pairs that it otherwise can easily fit in isolation. All in all, this behaviour further confirms the promise of partial RNI as a means to combine the strengths of both deterministic and random features in a unified model framework.

5.5.3 How Does RNI Affect Learning over Sparse Datasets?

In addition to the experiments on EXP and CEXP, we additionally consider the impact on RNI on sparser datasets, i.e., datasets with very few training instances.

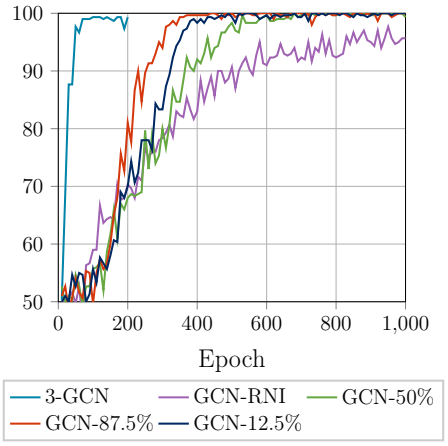
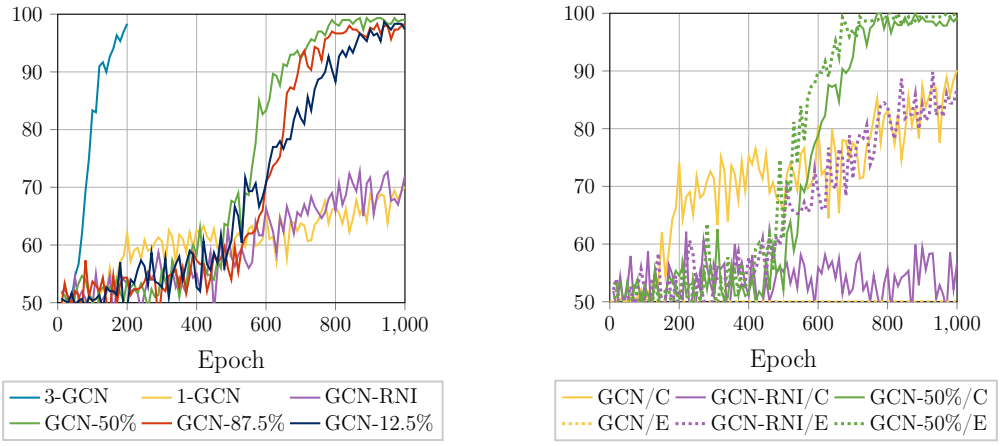


Figure 5.5: Learning curves on SparseExp for GCN-RNI and competing models.

To this end, we propose sparse analogs to Exp and CExp, namely SparseExp and SparseCExp, that only contain 25% of the number of instances (150 pairs) of their original counterparts, and subsequently conduct analogous training and hyper-parameter tuning on these datasets.

SparseExp. We generate SparseExp analogously to Exp, except that this dataset only consists of 150 graph pairs. To alleviate for the sparsity of SparseExp, we double the training time of all models, and thus train 3-GCN for 200 epochs, and all other systems for 1000 epochs, but otherwise use an identical setup to the earlier experiments. We show the learning curves for all models on SparseExp in Figure 5.5.

This figure shows that all models converge slower on SparseExp compared to Exp. This aligns with expectations, as a lower data density makes learning a well-performing function more challenging. In particular, sparsity implies that (i) fewer weight updates are made per epoch, and (ii) these updates are of lower quality, as they are computed from a less complete dataset. Nonetheless, the same convergence patterns for GCN-RNI models and 3-GCN are also present in this sparse setting. Moreover, all GCN-RNI models, though eventually converging, do so in a more volatile fashion compared to the standard Exp dataset. Indeed, GCN-RNI models suffer from the sparseness of the dataset, which makes them more sensitive to RNI. As a result, these models require more training to effectively learn robustness against RNI values, increasing their variability further. Moreover, the nature of SparseExp makes learning more difficult, as, like Exp, it necessitates RNI for MPNNs have a chance of achieving above-random performance. Hence, the volatility and variability introduced by RNI are particularly evident in this sparser data setting.



(a) SparseCExp. (b) Sparse $\overline{\text{Exp}}$ (/E) and Corrupt (/C).

Figure 5.6: Learning curves on the SparseCExp dataset, as well as its sparse subsets Corrupt and $\overline{\text{Exp}}$, for GCN-RNI and competing baseline models.

SparseCExp. Analogously to the previous experiment, we generate a SparseCExp dataset with only 150 graph pairs. We report the learning curves for all models on SparseCExp in Figure 5.6. As with the difference between Exp and SparseExp, the model converges slower on SparseCExp compared with the denser CExp. However, the "fluctuation" phase, which only occurs during the first 100 epochs over CExp due to conflicting learning requirements stemming from Corrupt and $\overline{\text{Exp}}$, lasts for around 500 epochs on SparseExp. This further highlights the increased volatility of RNI with higher data sparsity, as sparsity makes that further samples are needed in expectation to learn a high-quality solution function, leading to a lengthy "fluctuation" phase, in which both Corrupt and $\overline{\text{Exp}}$ data points conflict with one another during model optimisation.

5.6 Summary and Outlook

In this chapter, we studied the expressive power of MPNNs with RNI, and showed that these models are universal and preserve MPNN invariance in expectation. Empirically, we evaluated MPNNs with RNI on carefully designed datasets, and observed that RNI improves their learning ability, but slows their convergence.

All in all, this chapter proposes a surprising but powerful theoretical result, supported by practical insights, to quantify the effect of RNI on GNNs and unambiguously describe the increase in expressive power that RNI delivers. It also proposes EXP and CEXP as challenging benchmarks through which future graph neural network models can be efficiently and effectively evaluated from both an expressiveness

and learning perspective. Building on these findings, an interesting research direction is to study the power of RNI in other contexts beyond GNNs. In fact, this result has already been used to show the university of equivariant quantum graph circuits [120]. Moreover, another interesting topic for future work is to study the classes of functions that can be captured by MPNN using efficient constructions, both with and without RNI, see, e.g., [69] for related open problems.

Chapter 6

Learning to Reason with Graph Neural Networks

6.1 Introduction

Propositional *model counting (MC)*, or *#SAT*, is the task of counting the number of satisfying assignments for a given propositional formula [66]. *Weighted model counting (WMC), or weighted #SAT,* additionally incorporates a *weight function* over the set of all possible assignments. Offering an elegant formalism for encoding various probabilistic inference problems, WMC is a unifying approach for inference in a wide range of probabilistic models [95, 50, 46, 158, 23].

Two important special cases of WMC are WMC(CNF) and WMC(DNF), which require the input formula to be in conjunctive normal form (CNF) and disjunctive normal form (DNF), respectively. Inference in probabilistic graphical models typically reduces to solving WMC(CNF), while query evaluation in probabilistic databases reduces to solving WMC(DNF) instances [158]. However, both WMC(CNF) and WMC(DNF) are known to be #P-hard [168], and this computational complexity is a major bottleneck for solving large-scale WMC instances.

To overcome this problem, two main paradigms have been developed. The first paradigm is *knowledge compilation* [28, 151], which solves computationally difficult problems by compiling them into a new representation, i.e., a target language, where they can subsequently be solved efficiently. Following compilation, exact inference in WMC can be done in linear time [44]. However, the compilation process can produce exponentially-sized problem representations, i.e., arithmetic circuits, leading to intractable computational requirements at the compilation step. Furthermore, knowledge compilation is not robust to changes. Indeed, for every change in the underlying model, the computationally demanding knowledge compilation process needs to be

repeated. As a result, approaches based on knowledge compilation struggle to scale to large and varying problem instances.

The second paradigm is approximate solving [55, 33, 119], which provides approximations of the (weighted) model count as opposed to an exact solution, potentially with probabilistic guarantees. Loosening the requirement for exact solutions renders WMC more tractable, especially for WMC(DNF) counting, where approximate solving admits a fully polynomial randomised approximation scheme (FPRAS) due to Karp, Luby, and Madras [84], which we denote KLM. KLM allows for faster estimation of (weighted) model counts, while also providing probabilistic guarantees on its approximations relative to a desired error and confidence, and is the state of the art approach for WMC(DNF). However, KLM runs in $O(mk)$, where m denotes the number of Boolean variables and k the number of clauses of the input DNF formula. Hence, KLM struggles to scale to real-world DNF formulas.

In this chapter, we propose Neural#DNF, an approach that combines deep learning, more specifically message passing neural networks (MPNNs), and approximate model counting and enables fast WMC(DNF) approximation. To this end, we first generate random WMC(DNF) instances for training and evaluation, and obtain labels for these using KLM. This data is carefully encoded as a graph structure, so as to accurately represent the variables and clauses of the original DNF formula. Then, we propose a custom MPNN with a dedicated message passing protocol to exploit the symbolic DNF structure and make WMC(DNF) predictions. More specifically, Neural#DNF uses a sequential message passing structure, in which the layers of the input formula communicate in sequence, rather than the standard parallel message passing structure typically used in MPNNs, and this offers the model a more faithful computational flow to represent WMC(DNF) computations.

By construction, Neural#DNF produces approximations in $O(k\bar{W})$ time, where $\bar{W}$ denotes the average clause width, i.e., the number of literals per clause. Furthermore, when width is bounded, this asymptotic runtime bound reduces to just $O(m + k)$. As this approach is based on a neural network, Neural#DNF does not provide probabilistic guarantees, but in turn enables a speed-up of multiple orders of magnitude in the average case, thus offering much better scalability and a viable, efficient alternative for computing informed WMC(DNF) estimates. Empirically, we show that Neural#DNF learns to accurately estimate weighted model counts and generalises to novel formulas. Indeed, our model computes solutions to unseen WMC(DNF) instances with 99% accuracy relative to an additive error threshold of 0.1 with respect to tight KLM approximations. It also generalises to larger problem instances

involving up to 15 thousand variables remarkably well, despite only seeing formulas with at most 5 thousand variables during training. All in all, our findings suggest that the Neural#DNF MPNN can effectively and efficiently perform large-scale model counting through training on dense and reliable data.

The rest of this chapter is organised as follows. Section 6.2 introduces the weighted model counting problem, Section 6.3 introduces the Neural#DNF approach, and Section 6.4 presents an empirical evaluation of this model. Section 6.5 presents related work for approximately and exactly solving weighted model counting, and Section 6.6 concludes this chapter.

6.2 Weighted Model Counting

Given a propositional formula φ (cf. Chapter 2), its *model count*, denoted $\#\varphi$, is the number of assignments ν satisfying φ. The *weighted model count* (WMC) of φ is defined as:

$$\mathrm{WMC}(\varphi) = \sum_{\nu \models \varphi} w(\nu),$$

where $w : \mathfrak{A} \mapsto \mathbb{R}$ is a *weight function*, and $\mathfrak{A}$ is the set of all possible assignments. We denote by WMC(CNF) and WMC(DNF) the special cases of WMC, where the class of input formulas is restricted to CNF and DNF, respectively. We now illustrate WMC with a simple example:

Example 6.2.1. *Consider the disjunction $p_1 \vee p_2$. This disjunction has a model count of 3, corresponding to all possible assignments of p_1, p_2 except $(False, False)$. Now consider the weight function*

$$w(\nu) = \begin{cases} 0.4 & \text{if both } p_1, p_2 \text{ are true,} \\ 0.25 & \text{if either } p_1 \text{ or } p_2 \text{ is true,} \\ 0.1 & \text{otherwise.} \end{cases}$$

Then, the weighted model count of $x_1 \vee x_2$ is $0.4 + 0.25 + 0.25 = 0.9$.

In this chapter, we set $w : \mathfrak{A} \mapsto [0, 1] \cap \mathbb{Q}$ such that every assignment is mapped to a rational probability and $\sum_{\nu \in \mathfrak{A}} w(\nu) = 1$. As common in the literature, we view every propositional variable as an independent Bernoulli random variable and assign probabilities to literals. That is, every variable p_i is assigned a rational truth weight w_i, and the weight of an assignment ν is computed as $\prod_{i=1}^{|S|} w_i$, where $\prod$ denotes the product operation.

Approximate WMC over DNF structures. Model counting (resp., weighted model counting) problems are #P-hard [168] to solve exactly, and thus are intractable for exact computation. As a result, techniques for efficient *approximations* to model counting have been devised, a special class of which being *fully polynomial randomised approximation schemes* (FPRAS). Given a target error $0 < \epsilon < 1$ and confidence $0 < \delta < 1$, an FPRAS computes an approximation $\hat{\mu}$ of the actual solution μ, in polynomial time w.r.t. the input, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, such that

$$\Pr\big(\mu(1-\epsilon) \leq \hat{\mu} \leq \mu(1+\epsilon)\big) \geq 1 - \delta.$$

For WMC(DNF), the Karp, Luby, and Madras [84] algorithm (KLM), a special case of the Linear-Time Coverage (LTC) [111] algorithm, is an FPRAS. KLM is an iterative random sampling procedure, which samples assignments satisfying a random clause in the input DNF, and then verifies this assignment against another uniformly randomly selected clause, to compute an unbiased estimator of the overall model count. More specifically, for a DNF φ with m propositional variables and k clauses $c_1, \ldots, c_k$, KLM runs $T = 8(1+\epsilon)k\log(\frac{2}{\delta})\frac{1}{\epsilon^2}$ sampling iterations, to compute a successful trial count N, from which the WMC estimator is produced. At every trial, KLM performs the following steps:

1. (**Sampling**) If no current sample assignment τ exists, then a random clause c_i is selected with probability $\Pr(c_i)/\sum_{j=1}^{k}\Pr(c_j)$ (i.e., proportionally to its probability), where $\Pr(c_i) = \prod_{p \in c_i} w_b(p)$. Afterwards, τ is sampled uniformly from the set of satisfying assignments for c_i. More concretely, the variables appearing in c_i are all set to make c_i true, and all remaining variables are sampled using the weight function w. If τ already exists, proceed to Step 2.

2. (**Verification**) Another clause c' (which could be identical to c_i) is *uniformly* randomly sampled with probability $\frac{1}{k}$, and τ is checked against c'. If $\tau \models c'$, N is incremented and τ is re-sampled. Otherwise, the sampled assignment τ is preserved, and re-used in the next trial.

KLM returns $T\sum_{j=1}^{k}\Pr(c_j)/kN$ as an estimate for WMC(DNF). Informally, this estimator is designed to count every distinct satisfying assignment exactly once in expectation, and proportionally to its weight. This is done by making assignments simultaneously satisfying multiple clauses, which are likelier to be sampled in Step 1, increase N more frequently. This lowers the overall estimator, and levels out the contribution of such assignments relative to that of less prominent assignments. In terms of running time, assignment checking in Step 2, as well as sampling in Step 1, run in $O(m)$. Therefore, KLM runs in time $O(mT) = O\big(mk\epsilon^{-2}\log(\frac{1}{\delta})\big)$.

6.3 Approach Overview

In this section, we propose Neural#DNF, a custom MPNN for approximately solving WMC(DNF). At a high level, Neural#DNF applies to graph representations of DNF formulas and performs a dedicated message passing procedure to compute an approximate weighted model count. Therefore, we first discuss the graph representation for DNF formulas, and show how this representation preserves semantic properties of propositional formulas. Then, we discuss the Neural#DNF MPNN, namely its sequential message passing and type-specific update functions.

6.3.1 Representing Propositional Formulas as Graphs

To encode a propositional DNF formula, we use a 3-layer graph representation, where each layer captures different levels of abstraction within the DNF. First, we introduce a *literal* layer, which contains nodes corresponding to literals in the input formula. That is, for a formula with m Boolean variables, the literal layer contains $2m$ nodes, corresponding to the 2 possible literals (positive and negated) for every variable. Furthermore, for any variable p, the two corresponding literal nodes are connected by an edge to highlight their complementarity. Hence, the literal layer contains m connected components, each of which consists of 2 nodes and a connecting edge between them.

On top of the literal layer, conjunctions in the DNF are represented using a *conjunction* layer. Concretely, for a DNF with k clauses, the conjunction layer introduces k corresponding conjunction nodes. These nodes are connected to literal layer nodes to map exactly with the literals appearing in the corresponding conjunction. In other words, an edge is placed between a literal node and a conjunction node if and only if the literal appears in the corresponding DNF conjunction. Finally, the graph representation introduces a single-node *disjunction* layer, such that this node, denoted s, is connected to all conjunction nodes. Therefore, for a DNF with m variables and k clauses, the corresponding graph representation includes $2m + k + 1$ nodes, as well as at least $m + k$ edges. An illustration of the graph representation for an example DNF formula is shown in Figure 6.1.

Observe that each layer represents information at different layers of granularity within the DNF. Indeed, the literal layer captures information at the local variable and literal level, and contributes directly to conjunction nodes. In turn, these nodes capture conjunction-level information, and are designed to model the interactions between literals within a given conjunction. Finally, the disjunction node s acts as a

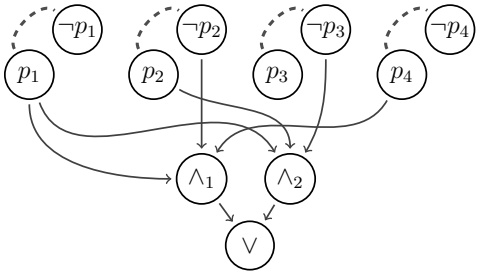


Figure 6.1: A graph encoding of the formula $\varphi = (p_1 \wedge \neg p_2 \wedge p_4) \vee (p_1 \wedge p_2 \wedge \neg p_3)$.

global super-node, which is designed to capture global interactions among conjunctions, and ultimately yield approximations for WMC(DNF).

In addition to its hierarchical design, this graph representation offers strong representational properties that align with propositional semantics, namely:

1. **Naming invariance.** Graph representations are produced independently from variable names, and thus semantically identical formulas with otherwise distinct naming conventions are mapped to isomorphic graphs.

2. **Permutation invariance.** DNF graph representations do not make any attribution to the order of literals or conjunctions in the formula.

3. **Negation invariance.** For unweighted DNF counting, replacing all literal occurrences with their negation (and vice-versa) does not affect the overall model count. This is preserved in our graph representation, as the representations of both the original and the "negated" formula are isomorphic.

Remark. Note that negation invariance does not necessarily hold in the weighted setting, as required. Specifically, for a variable with weight $w_i \neq 0.5$, negating all variable appearances changes the model count by a factor of $\frac{1-w_i}{w_i}$. This is appropriately handled in our approach, as we compute initial representations for literal nodes which are fully determined by w_i, and which can be interpreted as initial *node types*. Hence, performing negation changes the initial type of literal nodes in the weighted setting if and only if $w_i \neq 0.5$, and yields non-isomorphic graphs, as required.

6.3.2 Message Passing Approach

To approximate the model count of a DNF formula, we use a specialised MPNN model Neural#DNF that sequentially passes messages over the layers of the DNF graph and ultimately returns an approximate Gaussian distribution for WMC(DNF)

via the disjunction node. We now present all components of this MPNN, starting with the initialisation of node representations across the input graph.

Initialisation. An initial vector representation is computed differently for every node in the input graph based on its corresponding DNF component (literal, conjunction, disjunction). For nodes in the literal layer, the initial representation of the node for any literal p_i is computed based on its weight w_i, using a multi-layer perceptron (MLP) f_{enc}. Analogously, the representation of $\neg p_i$ is determined by $1 - w_i$. More formally, the d-dimensional representation $\mathbf{h}_{p_i}^{(0)}$ of a literal p_i with weight w_i is given by $\mathbf{h}_{p_i}^{(0)} = f_{\text{enc}}(w_i)$ (resp., $\mathbf{h}_{\neg p_i}^{(0)} = f_{\text{enc}}(1 - w_i)$). For *conjunction* and *disjunction* layer nodes, we use two initialisation vectors $\mathbf{h}_c$ and $\mathbf{h}_d$ respectively, and these vectors are learned over the course of training.

Message passing protocol. To approximate WMC(DNF), we use a *sequential* message passing protocol, such that (i) messages propagate in steps across the layers of the graph, and (ii) nodes use distinct update functions, namely distinct type-specific layer-norm LSTMs. In particular, a message passing iteration in Neural#DNF consists of the following four steps:

(a) Literal layer nodes compute messages based on current node representations using an MLP $f_{\text{litt}} : \mathbb{R}^d \to \mathbb{R}^d$. More concretely, given a literal p in the input DNF, the corresponding node for literal p prepares the message $f_{\text{litt}}(\mathbf{h}_p^{(t)})$. Then, all literal layer nodes pass their messages to conjunction layer nodes. In turn, these conjunction nodes then aggregate messages using the sum function and update their representations using a layer-norm LSTM $L_{\text{conj},1}$. The updated conjunction node representations, denoted $\hat{\mathbf{v}}_{c_i}^{(t+1)}$ for a given conjunction c, is given formally as:

$$\hat{\mathbf{v}}_c^{(t+1)} = L_{\text{conj},1}\Big(\mathbf{h}_c^{(t)}, \sum_{p \in N(c)} f_{\text{litt}}(\mathbf{h}_p^{(t)})\Big).$$

(b) Conjunction layer nodes compute and send messages to the disjunction node s via an MLP $f_{\text{conj}} : \mathbb{R}^d \to \mathbb{R}^d$, i.e., for a conjunction c, the message computed is $f_{\text{conj}}(\hat{\mathbf{v}}_c^{(t+1)})$. This disjunction node then aggregates these messages and updates its own representation using a layer-norm LSTM L_{disj}, i.e.,

$$\mathbf{h}_s^{(t+1)} = L_{\text{disj}}\Big(\mathbf{h}_s^{(t)}, \sum_{c \in N(s)} f_{\text{conj}}(\hat{\mathbf{v}}_c^{(t+1)})\Big).$$

(c) The disjunction node computes a message using an MLP $f_{\text{disj}} : \mathbb{R}^d \to \mathbb{R}^d$ and sends it to all conjunction layer nodes, which update their representations a second and final time. However, for this update, we use a different LSTM cell $_{\text{conj},2}$ to allow more flexibility and enable Neural#DNF to learn separate update procedures for the

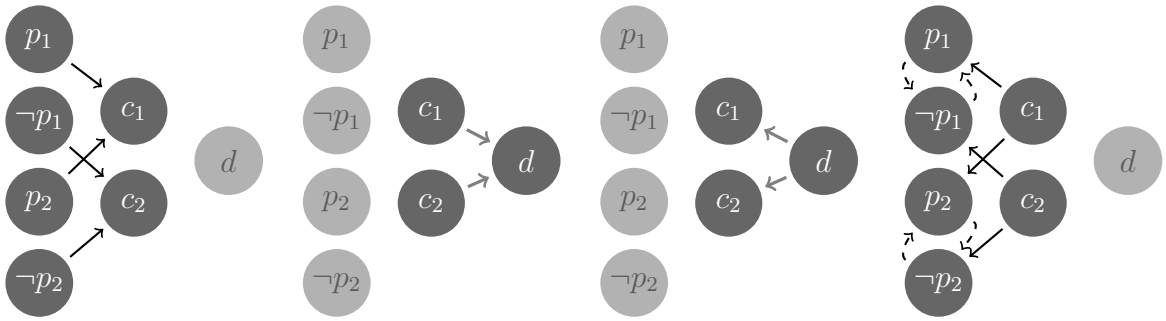


Figure 6.2: Message passing protocol on the DNF formula $\psi = (p_1 \wedge p_2) \vee (\neg p_1 \wedge \neg p_2)$.

local information propagation (Step (a)) and global information propagation (this step). For a conjunction c, this step is therefore written as:

$$\mathbf{h}_c^{(t+1)} = L_{\text{conj},2}\Big(\hat{\mathbf{v}}_c^{(t+1)}, f_{\text{disj}}(\mathbf{h}_s^{(t+1)})\Big).$$

(d) Using their latest representations, conjunction layer nodes send messages to neighbouring nodes in the literal layer. In parallel, literal layer nodes send messages to their negated counterparts. Thereafter, all literal layer nodes separately aggregate conjunction layer messages and *concatenate* this aggregate with the message from its corresponding negated literal. That is, a literal p sums all conjunction messages from $c \in N(p)$, i.e, it computes $\Sigma_{c \in N(p)} f_{\text{conj}}(\mathbf{h}_c^{(t+1)})$, and concatenates this with $f_{\text{litt}}(\mathbf{h}_{\neg p}^{(t)})$, yielding a $2d$-dimensional vector. Finally, this vector is used to update literal layer node representations using a layer-norm LSTM L_{litt}, yielding the equation:

$$\mathbf{h}_p^{(t+1)} = L_{\text{litt}}\Big(\mathbf{h}_p^{(t)}, \big(\sum_{c \in N(p)} f_{\text{conj}}(\mathbf{h}_c^{(t+1)}) || f_{\text{litt}}(\mathbf{h}_{\neg p}^{(t)})\big)\Big),$$

where $||$ denotes the concatenation operation.

To better illustrate this protocol, a visual representation of the four message passing steps over an example DNF formula is provided in Figure 6.2.

Making approximations. Following message passing, the final disjunction node representation $\mathbf{h}_s^{(T)}$ is passed through an MLP $f_{\text{out}} : \mathbb{R}^d \to \mathbb{R}^- \times \mathbb{R}^+$, where the two neurons of the layer return the log mean, a negative real number, and standard deviation, a positive number. To enforce these output ranges, we use the exponential linear unit (ELU) [41] for the last layer of f_{out}. More specifically, we use:

$$ELU + 1(x) = \begin{cases} e^{-x} & \text{if } x \leq 0 \\ x + 1 & \text{otherwise,} \end{cases}$$

which returns a number in $\mathbb{R}^+$, such that the mean neuron uses $-[ELU + 1](x)$, and the standard deviation neuron uses $ELU + 1(x)$. Note that we opt for predicting the log mean (or the log probability, generally speaking), rather than the mean directly, as predicting a number in $\mathbb{R}^-$ offers the model more flexibility to make fine-grained differences as opposed to a simple map to $[0, 1]$ with flattening at both extremes, e.g., using a sigmoid. The final output of the model is therefore a predicted Gaussian distribution for the log probability for WMC(DNF).

Loss computation. To train Neural#DNF, DNF formulas are approximately solved using the KLM algorithm, producing an (ϵ, δ) approximation which we subsequently use as our ground-truth target. For the purposes of training, we first map the aforementioned error and confidence bounds into a Gaussian distribution. This allows for tractable model training, as computing the divergence between probability distributions admits an efficient closed-form equation for Gaussian distributions.

We first recall the output of KLM. Given ϵ and δ, KLM returns an estimate $\hat{\mu}$ of the true weighted model count μ within a multiplicative bound with respect to ϵ, and this bound holds with probability $1 - \delta$. By identifying different configurations of ϵ and δ that lead to an identical number of KLM iterations, one can notice that the probability mass of the distribution over μ is concentrated around $\hat{\mu}$ and decays (non-uniformly) away from it. We formalise this by rewriting the probabilistic guarantee of KLM with respect to μ, the true weight model count. That is, we rewrite the bound $(1 - \epsilon)\mu \leq \hat{\mu} \leq (1 + \epsilon)\mu$ such that μ is at the centre of the inequalities. This yields the following inequality, which holds with probability $1 - \delta$:

$$\frac{\hat{\mu}}{1 + \epsilon} \leq \mu \leq \frac{\hat{\mu}}{1 - \epsilon}.$$

This bound is multiplicative relative to $\hat{\mu}$ with respect to ϵ, and this makes it hard to fit standard distributions on it. Furthermore, it uses different multiplicative factors $(1 - \epsilon)$ and $(1 + \epsilon)$ and is thus not centred at μ. Hence, we first apply a natural logarithm to get an additive bound on $\log \mu$:

$$\log \hat{\mu} - \log (1 + \epsilon) \leq \log \mu \leq \log \hat{\mu} - \log (1 - \epsilon).$$

Then, we fit a Gaussian $\mathcal{N}(\mu', \sigma)$ setting $\mu' = \hat{\mu}$ and $\sigma = {\log (1 + \epsilon)}/{F^{-1}(1 - \frac{\delta}{2})}$, where F^{-1} denotes the inverse cumulative distribution function of the standard Gaussian distribution. This Gaussian *approximates* the earlier bound over a tighter interval by replacing $\log(1 - \epsilon)$ on the right-hand side with the uniform $-\log(1 + \epsilon)$[1]. By design,

[1]For $0 < \epsilon < 1$, $-\log(1 - \epsilon) > \log(1 + \epsilon)$, as $-\log(1 - \epsilon) = \log(\frac{1}{1-\epsilon})$ and $\frac{1}{1-x} > 1 + x$ for $0 < x < 1$.

this Gaussian ensures that the interval

$$\log \hat{\mu} - \log (1 + \epsilon) \leq \log \mu \leq \log \hat{\mu} + \log (1 + \epsilon),$$

is assigned a probability of $1 - \delta$. Therefore, this distribution strictly satisfies the earlier KLM-induced bound, and offers a tractable means to compute distribution divergence. Finally, using this Gaussian, and using the model output Gaussian, we compute Kullback-Leibler (KL) divergence, which between two Gaussians $\mathcal{N}_1(\mu_1, \sigma_1)$ and $\mathcal{N}_2(\mu_2, \sigma_2)$ is given by:

$$KL(\mathcal{N}_1, \mathcal{N}_2) = \log \frac{\sigma_2}{\sigma_1} - \frac{1}{2} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2}.$$

We set $\mathcal{N}_1$ to be the prediction returned by the network and $\mathcal{N}_2$ to be the KLM approximation. This choice is critical in order to avoid the system minimising the training loss by learning to produce arbitrarily large values of σ_2.

6.3.3 Model Complexity

We now study the runtime complexity of Neural#DNF in the average case, best case, and worst case settings. Given an input DNF formula with m variables, k clauses, and an average clause width of $\bar{W}$. We show that our tool has average-case complexity $O(k\bar{W})$. This complexity is therefore more efficient than KLM, which has complexity $O\big(mk\epsilon^{-2}\log(\frac{1}{\delta})\big)$ as $\bar{W} << m$ in practice.

Average case. In a message passing iteration, $2\sum_{i=1}^{k} W_i$ messages are passed between literal layer nodes and conjunction layer nodes, where W_i denotes the width of clause c_i. This sum can be rewritten as $2k\bar{W}$, where $\bar{W} = \frac{1}{k}\sum_{i=1}^{k} W_i$. Furthermore, $2m$ messages pass between literal nodes and their negations, and $2k$ messages pass between conjunction nodes and the disjunction node s. All messages are summed at their respective destinations, thus yielding as many additions as messages passed. Following this, $2m + k + 1$ constant-time (since d is fixed) state updates are made. Hence, a total of $2k\bar{W} + 2k + 2m$ messages are passed. This term is $O(k\bar{W})$, as $k\bar{W} \geq m$ given that all variables appear in at least one clause. Hence, the average-case complexity of a message passing iteration is $O(k\bar{W})$. Therefore, since the number of message passing iterations T is fixed, the average-case complexity of Neural#DNF is also $O(k\bar{W})$.

Worst case. The worst-case non-redundant scenario is where all clauses have width m, and thus $\bar{W} = m$. In this scenario, $2mk$ messages are passed between literal layer

nodes and conjunction layer nodes, which implies a worst-case complexity $O(mk)$, analogously to KLM.

Best case. In the best-case scenario, clause width is upper-bounded by a constant. Therefore, the number of messages passed becomes $O(m + k)$. This is not superseded by any other computation, as representation updates are also $O(m + k)$. The complexity of a message passing iteration therefore reduces to $O(m + k)$, allowing for *linear-time* estimation of WMC(DNF).

6.4 Experimental Evaluation

In this section, we train Neural#DNF on large sets of DNF formulas and measure its generalisation relative to new DNF formulas. To this end, we study the performance of the model relative to two main criteria. First, we study model generalisation to distinct *structures*, where DNF formulas differ in terms of the configuration of clauses and variables relative to training instances. Second, we study the generalisation of Neural#DNF relative to *larger* formulas including more variables and clauses. Parallel to these objectives, we also evaluate the practical running time of our model, and study its performance relative to the number of message passing iterations as an ablation experiment.

For training and evaluation across all experiments, we use a novel random data generation procedure to produce DNF formulas, owing to the lack of DNF benchmarks in the literature [118]. This procedure is carefully designed to produce strong dependencies between clauses, and avoid trivial statistical behaviours stemming from unconstrained random generation, such as mutual clause exclusion and uniform literal appearances. Therefore, we present this procedure in detail next, and conduct additional experiments to validate its importance later in this section.

6.4.1 Data Generation

To generate DNF instances for our experiments, we propose a novel randomised generation procedure. This procedure takes as input a target number of Boolean variables m, a target number of clauses k, and a fixed clause width W for all clauses in the formula[2]. Our procedure divides generation into two parts. First, all variables are randomly assigned appearance counts in the generated DNF, indicating the number of non-redundant appearances they make. Second, these appearances are placed into

[2]Note that this can easily be generalised to an arbitrary width range, which can be uniformly sampled during generation, but we omit this for simplicity.

concrete clause allocations that are non-redundant and that create large dependencies between clauses. Finally, probabilities for all variables are randomly generated.

Appearances allocation. To compute appearances for variables in a DNF, our procedure adopts an abstraction where all possible positions in the DNF are viewed as "slots", shares of which are distributed across variables. Indeed, for a DNF with fixed clause width W and k clauses, the procedure considers kW "slots", corresponding to all possible positions that literals can occupy across all k clauses. Then, it produces allocation *quotas* across these kW slots to the m variables, such that every variable is promised at least one slot, i.e., is not redundant. Notice that this allocation quota simply specifies *how many* times a variable appears, and not *where* it appears, as randomly sampling slots directly could lead to redundancies within clauses.

Computing allocation quotas is first done uniformly, by corresponding to the combinatorial problem of putting kW indistinguishable balls into m boxes, such that no boxes are empty, and by additionally imposing the constraint that no variable is allocated more than k slots[3](as this necessarily creates redundancies). Furthermore, we subsequently set the literal sign for every appearance uniformly at random, i.e., for every slot, a variable p can appear as the positive literal p with probability 0.5. In this case, we refer to this mechanism as the *uniform allocation* mechanism. However, this type of procedure is unlikely to produce highly variable allocations. Indeed, every variable is expected to appear $\frac{kW}{m}$ times in the generated formula, and this number of appearances corresponds to the sum of independent and identically distributed Bernoulli trials. Therefore, by a probabilistic argument using the Chernoff bound, one can observe that it is highly unlikely to produce formulas with variable slot distributions, and this makes individual variables have little impact on the model count. This is undesirable, as it undermines the contributions of individual variables, and encourages overfitting to higher-level parameters, namely W, k, and n.

To tackle this, we propose a more refined allocation mechanism, which we refer to as *privileged allocation*. In this mechanism, we introduce two new variables $q, r \in [0, 1]$, where q helps specify the size of a subset of "privileged" variables, to be allocated more slots, and r specifies the proportion of all excess slots to be used for these privileged variables. Concretely, we define a set of $q \cdot m$ "privileged" variables, and exclusively assign these variables $r \cdot (kW - m)$ slots at random, where $kW - m$ is the *excess* number of slots above the minimum. The remaining

[3]This constraint requires that $W \leq m$, by the pigeonhole principle. In practice, we impose this constraint using rejection sampling: We randomly sample an allocation from the $\binom{kW-1}{m-1}$ options until a satisfying one is found. This is practically efficient as the probability of a given box having k or more balls is extremely small in our setting.

$kW - r(kW - m) = (1 - r)kW + rm$ slots are then subsequently uniformly allocated to all variables (including privileged ones) as before. Therefore, in this privileged mechanism, the expected number of allocations for a privileged variable given non-zero values of q and r is:

$$\frac{r(kW - m)}{qm} + \frac{(1 - r)(kW) + rm}{m}.$$

By contrast, all other (non-privileged) variables have a smaller expectation, namely the right-hand term in the above equation.

To further enforce dependence on privileged variables, all corresponding literals for a privileged variable are *jointly* set to an identical sign, i.e., all privileged variable literals are either positive with probability 0.5 or negative otherwise. By unifying literal signs, this generation procedure minimises the chance of creating mutually exclusive clauses, and thus simplifying the weighted model counting problem. Hence, privileged variables are designed to make a more substantial contribution to the weighted model count than their uniform counterparts.

Literal placement. Once all variable allocations are determined, all variables are sorted in decreasing allocation order and then assigned to clauses in that order. This ensures that more prominent variables, which appear more in the formula, are accommodated first, when more empty slots are available, thus maximising the likelihood of generation success. In this placement phase, a variable with a ($\leq k$) slot allocations will be assigned to a clauses by sampling from all k clauses without replacement. Further heuristics are also added to this mechanism to prioritise selecting clauses with more empty slots during this sampling, so that all clauses are "filled" uniformly.

Variable probability generation. Probabilities for variables are first chosen uniformly at random. More specifically, we use four distinct distributions for every formula in our training and evaluation sets, such that (i) one distribution is randomly generated (by uniformly sampling from $]0, 1[$), and (ii) the other 3 distributions are quarter increments of the random distribution modulo 1. For example, if a variable p_1 is assigned probability 0.1 for the random distribution, it will have probabilities 0.35, 0.6, and 0.85, respectively, in the 3 other distributions. This ensures that we produce formulas with model counts covering the entire $[0, 1]$ range as evenly as possible, so as to have more representative training and testing data.

6.4.2 Generalisation Experiments

We first train Neural#DNF on a large set of DNF formulas and evaluate its performance on distinct formulas, as well as formulas of larger size than its training set.

Table 6.1: Distribution of DNF formula sizes in the training set.

Size (m)	**50**	**100**	**250**	**500**	**750**	**1000**	**2500**	**5000**
Count	30000	20000	16000	12000	10000	8000	6000	3000

We first present the selected data generation configuration, evaluation metrics, and computational setup for Neural#DNF in our experiments.

Data generation. In our experiments, we generate DNF instances split evenly between the uniform and privileged allocation mechanisms. Specifically, we set $q = 0$ and $r = 0$, i.e., we use uniform allocation, with probability 0.5. For the remaining 50%, we sample q from the set of multiples of $\frac{1}{m}$ in the range $[0, \frac{\log(m)}{m}]$ using the standard exponential distribution ($\lambda = 1$). Hence, we select at most $\log(m)$ privileged variables, and give higher likelihood to smaller values for q through the exponential distribution, and thus this process is highly selective. Furthermore, we set r to the largest value which ensures that no privileged variable gets allocated more than k slots with probability at least 0.5 (by a one-sided Chebyshev bound).

Using these settings for q and r, we generate 100 thousand distinct formulas to train Neural#DNF, following the distribution over variable count m shown in Table 6.1. For every m, formulas are generated with fixed clause width $W \in \{3, 5, 8, 13, 21, 34\}$ and number of clauses k from $\{0.25, 0.375, 0.5, 0.625, 0.75\} \cdot m$, such that every valid setting, i.e., all configurations except $W = 3$ and $k = 0.25 \cdot m$, is represented equally, and such that each formula is generated with 4 variable probability distributions, as mentioned earlier. To label these formulas, we use KLM with $\epsilon = 0.1$ and $\delta = 0.05$ as a reasonable trade-off between label accuracy and generation tractability, and run this on a compute node with a single Haswell E5-2640v3 CPU and 64 GB of memory.

Parallel to this training dataset, we generate two evaluation datasets using a largely analogous setup, for structure generalisation and size generalisation respectively. The structure generalisation dataset is generated identically to the training set, but uses roughly 12.5% of the formula counts for every value m given in Table 6.1. It thus includes around 13,000 distinct, structurally different formulas. On the other hand, the *size* generalisation testing dataset contains 464 formulas: 348 formulas with $m = 10,000$ variables and 116 formulas with $m = 15,000$. Specifically, we generate 12 distinct formulas for each of the 29 valid configurations for $m = 10,000$, and 4 distinct formulas per configuration for $m = 15,000$. Finally, given the computational constraints of running KLM on large formulas, we only compute

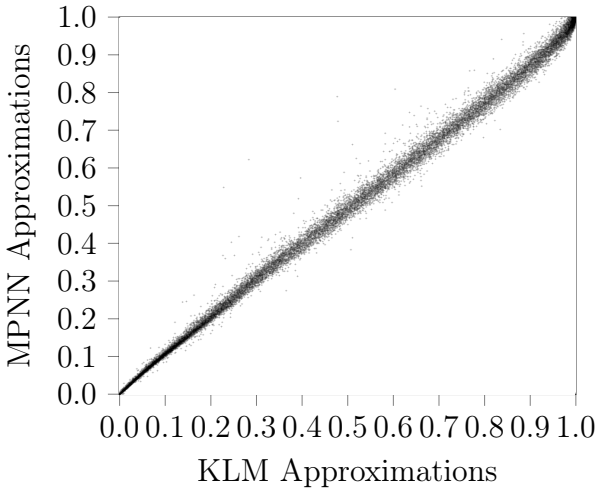


Figure 6.3: A gray-scale heat map representing the distribution of model predictions compared to KLM approximations.

Table 6.2: Model accuracy (%) with respect to all additive thresholds.

Evaluation Data	**Thresholds**			
	0.02	0.05	0.10	0.15
Training Set	87.14	98.80	99.97	99.99
Test Set	87.37	98.76	99.95	99.98

an approximate WMC for the initial random distribution for every formula, rather than the 4 distributions used in the other datasets.

Evaluation metric. In our experiments, we compare Neural#DNF predictions $\hat{\mu}$ with those of KLM and check whether their *absolute difference* falls within pre-defined additive thresholds. We opt for additive error, as opposed to multiplicative error, as the former produces an absolute distance metric, whereas the latter is relative to the target model count.

Model setup. For all experiments, we use $d = 128$-dimensional vector representations. We define f_{enc} as a 3-layer MLP with layer sizes 8, 32, and 128, message-generating MLPs (f_{litt}, f_{conj}, and f_{disj}) as 4-layer MLPs with 128-sized hidden layers, and f_{out} as a 3-layer MLP with layers of size 32, 8, and 2. We use the rectified linear unit (ReLU) as the activation function at MLP hidden layers, and linear activation at the output layer for all MLPs except f_{out}, which uses the ELU function as explained in Section 6.3. Finally, we train the system for 4 epochs on a P100 GPU using KL divergence loss, the Adam optimiser [91], a learning rate of $\lambda = 10^{-5}$, a gradient clipping norm threshold of 0.5, and $T = 8$ message passing iterations.

Table 6.3: Model accuracy (%) for structure generalisation by threshold versus number of formula variables (m).

m	**Thresholds**			
	0.02	0.05	0.10	0.15
50	85.58	98.58	99.98	100.0
100	87.87	98.87	100.0	100.0
250	87.93	99.24	100.0	100.0
500	87.67	99.40	99.99	100.0
750	87.56	99.15	100.0	100.0
1000	86.79	99.01	99.98	100.0
2500	90.06	98.17	99.85	99.94
5000	88.15	95.86	99.48	99.74

Table 6.4: Model accuracy (%) over test set by threshold versus formula clause widths (W).

W	**Thresholds**			
	0.02	0.05	0.10	0.15
3	80.42	98.66	99.87	99.93
5	68.04	96.56	99.90	99.98
8	79.10	97.77	99.96	99.98
13	99.70	99.98	100.0	100.0
21	100.0	100.0	100.0	100.0
34	100.0	100.0	100.0	100.0

Results. On the structure generalisation test, Neural#DNF predictions align very closely with those of KLM, as shown in the heatmap in Figure 6.3. In fact, the model is within 0.02 of the KLM WMC estimate over 87.37% of the test set, and this rises to 99.95% for a threshold of 0.1. Moreover, model performance does not drop when switching between training and testing, which shows that the network has not memorised its training formulas, but rather learned a more general WMC procedure over the dataset. Overall test results, categorised by m, are given in Table 6.2.

Relative to the variable count m, the model maintains consistent and strong performance, with accuracy varying by at most 4.5% between any two different m values for all four test thresholds, as shown in Table 6.3. For example, Neural#DNF achieves at least 95.5% accuracy for threshold 0.05, across all m values. Hence, the model does not rely on a particular m to achieve its high overall performance. Notably, Neural#DNF is also robust against variation in W. As shown in Table 6.4, the model

Table 6.5: Accuracy (%) by threshold with respect to additive thresholds on size generalisation test formulas.

m	**Thresholds**			
	0.02	0.05	0.10	0.15
10,000	79.89	89.94	97.13	99.71
15,000	72.41	81.90	94.83	97.41

Table 6.6: Accuracy (%) by threshold over size generalisation test formulas versus W.

W	**Thresholds**			
	0.02	0.05	0.10	0.15
3	78.13	90.63	98.44	100.0
5	73.75	90.0	100.0	100.0
8	76.25	91.25	98.75	100.0
13	40.0	56.25	82.5	95.0
21,34	100.0	100.0	100.0	100.0

scores above 96% and 99% across all widths for thresholds 0.05 and 0.1, respectively. Interestingly, it has near-perfect performance for larger widths 13, 21, and 34, where weighted model counts are near-zero, and has relatively higher accuracy at threshold 0.02 when $W = 3$, where counts are almost one. Simultaneously performing well in both extreme cases, coupled with high accuracy on intermediate widths, further highlights the robustness of the model.

On the size generalisation task, Neural#DNF maintains accuracies of 97.13% and 94.83% with a threshold of 0.1 on 10,000 and 15,000-variable formulas, respectively, despite having as many as triple the variables as the corresponding formulas in the training set. The full results for this experiment are given in Table 6.5, and the same results parametrised by width W are also given in Table 6.6. In the latter table, we see that the model performs consistently across widths 3, 5, and 8, but performs less well at $W = 13$. This is due to formulas with $W = 13$ exhibiting a "phase transition" at this m and k. Indeed, model counts fluctuate dramatically in this phase transition, since k is in the same order of magnitude as the inverse expected clause satisfaction probability. In the training set, phase transition occurs at smaller widths, but never for $W = 13$, so this is an entirely new situation for the model, at a much larger scale. Nonetheless, the model maintains reasonable approximation performance, achieving an encouraging accuracy of 82.5% for the threshold 0.1 despite the variability of the target WMC in this situation.

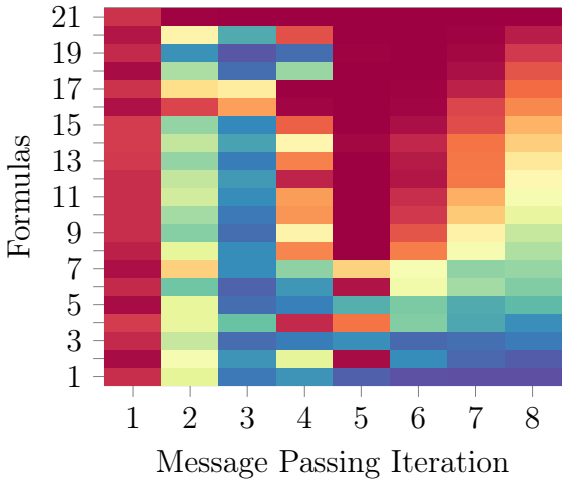


Figure 6.4: Model estimates over message passing iterations. Red denotes small probability and blue denotes high probability.

These results show that reliable approximate model counting on large-scale formulas can be achieved, even with training restricted to smaller, similarly generated and structured formulas. From a practical perspective, this provides further evidence that large-scale solvers can be trained using smaller formulas that can be tractably labelled with existing solvers.

All in all, our model performs remarkably both in terms of structure and size generalisation. These results highlight the potential and scalability of neural message passing (NMP) methods to perform reasoning tasks, and therefore justify further consideration of NMP for approximate reasoning tasks.

6.4.3 Performance Analysis

To examine how our model makes predictions, we selected 21 formulas $\text{DNF}_i : i \in [1, 21]$ from the structure generalisation dataset with weighted KLM model counts of roughly $\frac{21-i}{20}$. We then ran Neural#DNF on these formulas and computed the predicted probability at the end of every message passing iteration. Results for this experiment are visualised in Figure 6.4.

Initially, the model starts with a low estimate across all formulas. Then, within the first 3 iterations, it accumulates probabilities and hits a "spike". This phenomenon directly corresponds to messages from literal nodes reaching the disjunction node. Following these first 3 iterations, the model then "corrects" and lowers its estimates, before adjusting and refining its probability predictions in the final iterations.

Unlike NeuroSAT [152], where the estimate of satisfiability increases mostly monotonically, our network estimates fluctuate aggressively: A large initial estimate is made following the first two iterations, and this is then reduced and refined with more iterations. In doing so, the network seems to be initially estimating the naive sum of conjunction probabilities, and subsequently revisiting its estimates as it better captures intersections between conjunctions. Indeed, in the first two iterations, the model is unable to model intersections between different conjunctions, as it would still be learning more naive representations based only on the existing edges in the DNF. However, as more iterations occur, the message passing protocol, primarily through the DNF supernode, enables the sharing of more refined, global information, which better represents conjunction intersections. In turn, this more informed message passing at later iterations subsequently allows the model to adjust its predictions and better converge to the correct weighted model count.

6.4.4 Message Passing Study

Experimental setup. To evaluate the role of message passing in our model, we run the same experimental protocol as in the generalisation experiments, but with a varying number of message passing iterations T. First, we conduct an ablation study, where we only use $T = 2$ iterations for both training and testing. With 2 iterations, only one full pass through the 3 network layers is possible, and this pass can only capture limited information, as discussed in the performance analysis. Hence, we perform this ablation study to confirm the importance of using a sufficient amount of message passing iterations, and quantitatively observe the gap in performance that results from the reduction in T. Second, we consider the complementary question, namely: Does model performance continue to improve with more iterations? To this end, we run the model with $T = 32$ iterations and report our findings.

Results. In the ablation study ($T = 2$), we observe a large drop in system performance, both in terms of structure and size generalisation, and shown in Tables 6.7 and 6.8. These results further confirm that message passing is essential to model performance. Indeed, the model cannot deliver a reliable estimate with just 2 iterations, as it can only capture limited information about intersections between conjunctions. It is therefore limited to more naive guesses, which will very likely overestimate the target weighted model count, as shown in Figure 6.4.

With more iterations ($T = 32$), we see that the system surprisingly loses performance. In terms of structure generalisation, shown in Table 6.9, Neural#DNF achieves similar accuracy results compared with $T = 8$ with respect to both the

Table 6.7: Accuracy (%) on all thresholds on training and structure evaluation datasets for ablation study ($T = 2$).

Evaluation Data	**Thresholds**			
	0.02	0.05	0.10	0.15
Training Set	71.84	82.91	91.62	95.18
Test Set	72.08	82.80	91.36	94.97

Table 6.8: Accuracy (%) on all thresholds on size generalisation datasets for ablation study ($T = 2$).

m	**Thresholds**			
	0.02	0.05	0.10	0.15
10,000	71.84	76.72	84.20	89.94
15,000	66.37	69.83	79.31	87.93

training set and test set. However, when considering size generalisation (Table 6.10), the model performs substantially worse. This shows that, with more iterations, the model *overfits* its training set, and thus learns a message passing procedure which generalises poorly to larger formulas.

All in all, both earlier experiments highlight a trade-off between generalisation ability and representation capacity based on the selected number of message passing iterations T. Hence, this hyper-parameter must be carefully selected to achieve a compromise between both requirements. In particular, T must both be large enough to sufficiently iterate over the input graph and propagate necessary information, but also be small enough to prevent overfitting to the training set and allow for better size generalisation.

6.4.5 Runtime Analysis

To highlight the efficiency of Neural#DNF, we report the running time required by this model and compare with the time taken by KLM during data generation to approximately solve the same DNF instance. In particular, we show running times for Neural#DNF versus KLM ($\epsilon = 0.1, \delta = 0.05$) for formulas with ($W = \{3, 34\}$, $k = 0.75m$) at every m in Table 6.11. We note, however, that KLM and Neural#DNF ran over different hardware (Haswell E5-2640v3 CPU vs. P100 GPU, resp.), since they are best suited to their respective devices: CPUs more efficiently handle multiple operations, like sampling, slicing, and comparison, whereas GPUs are efficient for

Table 6.9: Accuracy (%) on all thresholds on training and structure evaluation datasets for $T = 32$.

Evaluation Data	**Thresholds**			
	0.02	0.05	0.10	0.15
Training Set	87.68	98.22	99.87	99.99
Test Set	87.32	98.02	99.83	99.95

Table 6.10: Accuracy (%) on all thresholds on size generalisation datasets for $T = 32$.

n	**Thresholds**			
	0.02	0.05	0.10	0.15
10,000	74.43	83.91	93.67	96.26
15,000	68.97	80.17	87.93	92.24

repetitive floating point operations. Therefore, these running times are only provided to highlight the running time *trends* of both approaches with respect to increasing formula size, in keeping with their respective running time expectations.

For $W = 34, m = 1,000$, we see that KLM requires 7.62 seconds to compute an approximation, and this rises rapidly to 305.61s for $W = 34, m = 15,000$. By contrast, Neural#DNF only needs 0.02 and 0.223 seconds, respectively. This aligns with theoretical expectations, as the model takes advantage of limited width and scales linearly with m, whereas KLM scales quadratically relative to m and the number of clauses k. Moreover, our model does not perform slower at smaller widths as with KLM, as it does not rely on sampling. By contrast, KLM relies on sampling iterations, where random assignments are replaced when they satisfy a clause. This implies that, in expectation, more replacements are made with smaller clause widths, as clause satisfaction is more likely, and thus that a heavy computational overhead is imposed for small W. This can be clearly observed in our results. For instance, KLM needs 2375 seconds to run on a formula with $m = 15,000$ and $W = 3$, but only requires 306s when $W = 34$. This limitation is not problematic for Neural#DNF, as it does not rely on such sampling procedures.

In general, Neural#DNF presents multiple properties that enable it to efficiently compute WMC(DNF) approximations. First, it has a smaller model complexity, which only relies on average clause width. Second, it can naturally run on high-performance hardware such as GPUs, and does not require computationally costly operations such as sampling. However, as a neural model, Neural#DNF does not

Table 6.11: Runtimes (seconds) for KLM, Neural#DNF by number of variables m and width W for $k = 0.75m$.

W	**Algorithm**	m			
		1,000	5,000	10,000	15,000
3	KLM	22.59	270.77	1151.86	2375.56
	Neural#DNF	0.017	0.040	0.073	0.104
5	KLM	15.32	145.47	608.13	1298.38
	Neural#DNF	0.017	0.042	0.077	0.110
8	KLM	9.32	81.39	322.56	689.50
	Neural#DNF	0.017	0.045	0.083	0.121
13	KLM	7.07	40.58	158.09	299.26
	Neural#DNF	0.017	0.050	0.094	0.138
21	KLM	7.26	41.28	157.85	293.83
	Neural#DNF	0.018	0.058	0.113	0.167
34	KLM	7.62	43.57	164.46	305.61
	Neural#DNF	0.020	0.074	0.145	0.223

offer theoretical guarantees, unlike classical algorithms such as KLM. Nonetheless, our earlier experiments validate the strong performance of this model, as well as its robustness and generalisability on our generated datasets. Therefore, our results highlight the promise of neural methods for efficient WMC(DNF) approximation, and provide a compelling alternative to formal methods in settings with very restricted computational resources.

6.4.6 Data Generation Procedure Validation

Experimental setup. To evaluate the impact of the privileged allocation mechanism, and of our proposed data generation procedure in general, we additionally generate two new synthetic datasets of identical size and composition to the original structure generalisation evaluation set. The first dataset is *entirely* based on our privileged mechanism generator, as opposed to 50% in the earlier experiments. We refer to this dataset as *"Fully Privileged"*. On the other hand, the second dataset is generated uniformly, using a random generator similar to earlier work [118], and to our uniform mechanism. We refer to this dataset as the *"Uniformly Random"* dataset.
Results. Results on both datasets are shown in Table 6.12. On the "Fully Privileged" dataset, model performance drops slightly relative to Section 6.4.2, especially for the

Table 6.12: Model accuracy (%) results over the "Fully Privileged" and "Uniformly Random" synthetic datasets.

Dataset	**Thresholds**			
	0.02	0.05	0.10	0.15
Fully Privileged	86.33	98.88	99.96	99.99
Uniformly Random	88.26	98.67	99.98	100.0

tight 0.02 threshold, but otherwise remains very strong. By contrast, model performance slightly *improves* on the "Uniformly Random" dataset at this same threshold. This aligns with our expectations, as uniform randomness produces formulas with model counts that better statistically correlate with choices of m, k, and W. Hence, these results show that our generation procedure helps alleviate the randomness of purely uniform generation, as it offers a good means to produce dependencies and obtain challenging, uncorrelated DNF data instances.

6.5 Related Work

WMC(DNF) belongs to the wider family of weighted model counting problems, which have been extensively studied due to their connection with probabilistic inference. Weighted #DNF is #P-hard [168], so is highly intractable. In fact, Toda proved that the class $\mathrm{P}^{\#\mathrm{P}}$ (the complexity class of problems solvable in polynomial time given access to a #P oracle) contains the entire polynomial hierarchy [163]. Even weighted #DNF counting on positive, partitioned DNF formulas with clause width at most 2 [138] remains #P-hard. The latter result is surprising, as the satisfiability (SAT) problem on this same DNF fragment, as well as any CNFs with width 2, can be solved in polynomial time.

Given the complexity of WMC, many methods and optimisations have been proposed to obtain exact or approximate WMC solutions. In terms of exact solving, one popular method is *knowledge compilation (KC)* [44]. In knowledge compilation, WMC problems are *compiled* into a new representation in which they are solved efficiently and exactly. This compilation acts as a pre-processing phase, and introduces a potentially exponential computational overhead. However, following compilation, the resulting representation can be directly queried, and allows for efficient, linear-time probabilistic inference [44]. Hence, KC is particularly effective when a same formula structure is repeatedly used, as compilation only occurs once per formula. However,

compiled representations can be of exponential size in the worst case, leading to intractable memory requirements. Moreover, compiled representations are typically not robust to input changes, as any change in the input formula requires compilation to run again from scratch. These limitations of KC have motivated some research towards *approximate* KC. In particular, this research performs approximate compilation either by pruning or simplifying the input formula so as to reduce its complexity [110], or alternatively by performing sampling over structural components of the formula [110, 61].

In addition to exact solvers, another important paradigm is to produce approximate solutions to circumvent the intractability of WMC [156]. For the unweighted case (MC), *hashing-based methods* [55, 34] (see also [33]) produce an approximation with probabilistic guarantees. Intuitively, these hashing techniques partition the solution space into smaller sub-spaces with similar satisfying assignment density, and continue to do so until reaching a subspace where exact computation is tractable. From there, the sub-space exact model count is computed and used to yield an approximate overall model count. For example, ApproxMC [34] applies to model counting for CNF formulas, but also yields an FPRAS when restricted to unweighted DNF; see, e.g., [119]. For CNF formulas, hashing methods for model counting can be applied for WMC, as approximation-preserving reductions are known. However, the existence of such a reduction remains open for DNF formulas [32]. Hence, *none* of these hashing methods are known to apply to WMC (DNF).

Beyond hashing techniques, *loopy belief propagation (LBP)* [134, 128] has been applied to approximate WMC. LBP is based on message passing, such that nodes exchange formally defined pairwise messages and update their local beliefs iteratively. However, LBP does not provide any guarantees for general classes of input graphs [178]. By contrast, the message passing used in this approach is learned from data, such that the model learns messages and states that best capture the necessary information to relay. Moreover, messages from a given node in Neural#DNF are restricted to be identical, which makes the total number of message computations in the model linear in the number of nodes.

Broadly speaking, our model builds on recent applications of graph neural networks [149] to a variety of reasoning tasks, such as the NeuroSAT system for solving SAT [152] and the proposed GNN for the travelling salesman problem (TSP) [137]. In particular, NeuroSAT applies a similar graph encoding, but does not use a supernode to aggregate global information. Instead, it aggregates literal node embeddings following message passing to yield a final satisfiability prediction. There has also been

work towards outright learning of inference in probabilistic graphical models using GNNs [187].

All aforementioned GNN works achieve encouraging results, and further motivate the study of GNN applications for reasoning. However, they are very limited in scope, as they can only train on very small instances (i.e., ∼40 variables) of their respective problems, and struggle to generalise to larger (but still small) instances. This is expected, as these models are learning to solve computationally hard problems: SAT and TSP are NP-complete and are hard to approximate with strong guarantees, whereas probabilistic inference in graphical models is #P-hard and remains NP-hard to approximate (as is WMC(CNF)). Thus, these approaches are of limited practical relevance in their current form.

By contrast, we propose an MPNN-based approach whose setup avoids these limitations. First, we tackle WMC(DNF), a problem which is #P-hard to solve exactly, but which admits a known polynomial-time approximation. This allows us to generate large-scale datasets to appropriately train the model, and also enables the generation of larger instances (with $m \geq 10,000$), to better evaluate and study size generalisation at practically relevant instance sizes. Furthermore, we aim for a subtly different objective in Neural#DNF relative to the earlier works. Indeed, the task being learned in our setting is *not* solving computationally hard problems, but rather searching for *more efficient* methods, based on neural message passing, for a problem with known, but practically slow, FPRAS algorithms. In this respect, the approximability of WMC(DNF) is a key distinguishing factor motivating our problem setup, study, and objectives. Hence, our model, to our knowledge, is the first proposal combining approximate reasoning and deep learning, while also scaling and strongly generalising to realistic and practically relevant problem instance sizes.

6.6 Summary and Outlook

In this chapter, we presented Neural#DNF, an approach that leverages the traditional KLM approximate weighted model counter for DNF formulas and uses it to train a custom message passing neural network to efficiently produce approximations to the WMC(DNF) problem. Our results show that neural models can be effectively and efficiently applied to large-scale weighted #DNF, given sufficiently dense and reliable training data. Therefore, it is particularly useful for query evaluation on large online probabilistic databases, where queries have computational limitations [30].

Beyond WMC it is interesting to analyse the viability of MPNNs for other reasoning problems admitting efficient approximations, and consider alternative approaches to avoid the training data bottleneck that naturally occurs with computationally intractable reasoning tasks. Parallel to this, it is interesting to jointly study the learning capacity of MPNNs in settings where their limited 1-WL expressive power [183, 127] could prove problematic. This is not the case in this chapter, as literal nodes effectively have random node initialisation stemming from their probabilities, and thus our MPNNs will likely not struggle to distinguish pairs of graphs (cf. Chapter 5). However, the limited expressive power of MPNNs could affect the learning of several tasks more generally, particularly tasks requiring, e.g., sub-structure detection.

Beyond expressive power, it is also important to research means to add probabilistic guarantees and improve the reliability of neural networks over reasoning tasks. The MPNN proposed in this chapter performs strongly in practice, and is a good candidate for learning advanced functions over graphs. However, it does not provide guarantees, unlike the KLM algorithm it is trained on. Therefore, we hope that our approach helps motivate further research, leading to less data-dependent methods, better problem-specific model designs with powerful inductive bias, and more principled future architectures that can incorporate the reliability guarantees of formal algorithms.

Chapter 7

Approximate Weighted Model Integration on DNF Structures

7.1 Introduction

Model counting (MC) is the task of counting the number of satisfying truth assignments (i.e., models) of a given propositional formula, and *weighted model counting* (WMC) generalises this task by incorporating a *weight distribution* over the set of truth assignments. Specifically, given a propositional formula, and a weight function that assigns every truth assignment a weight, WMC is the problem of computing the total weight of the models of the input formula [66]. Typically, WMC is studied with weight functions, where the weight of each assignment factorises into the weights of the individual variable assignments. WMC plays a very fundamental role in artificial intelligence, as it provides a unifying approach for encoding and solving different probabilistic inference problems that arise in various contexts, including *probabilistic graphical models* [95], *probabilistic planning* [50], *probabilistic logic programming* [46], *probabilistic databases* [158], and *probabilistic knowledge bases* [23].

Despite its wide applicability in artificial intelligence, WMC can only capture problems in discrete domains, since the whole formulation is based on *Boolean variables*. As a result, WMC cannot directly capture domains involving *real variables*, which puts even simple inference tasks in partly (or, fully) continuous domains, such as analysing the probability of discrete signals under Gaussian noise, beyond the scope of WMC. This observation has motivated the study of a generalisation of WMC, known as *weighted model integration* (WMI) [17, 125], which supports probabilistic inference in *hybrid domains* that consist of a mixture of discrete and real variables, and interactions among them.

The generalisation of WMC to WMI is inspired by the generalisation of the classical *satisfiability problem* (SAT) to *satisfiability modulo theories* (SMT) [14]. While SAT is limited to discrete domains, SMT allows to reason about the satisfiability of formulas involving, e.g., linear constraints over reals. This is realised through an SMT formula, which additionally contains real variables, and allows arithmetic statements (e.g., arithmetic (in)equalities) over these variables. Building on the foundations of SMT [14], WMI can be defined in a natural way: Given an SMT formula, and a weight function that defines a *density* for every truth assignment of the formula, WMI is the problem of computing the sum of integrals over the densities of all the satisfying assignments of the SMT formula [125]. In other words, WMI replaces the propositional formula from WMC with an SMT formula, and accordingly asks to compute a sum of integrals over weight functions defined over the satisfying assignments of the SMT formula.

One way of obtaining special classes of WMI is through appropriately restricting the class of input formulas, i.e., by only allowing formulas in *conjunctive normal form* (CNF), or by only allowing formulas in *disjunctive normal form* (DNF). In fact, WMC is widely studied in the literature relative to both CNF and DNF structures. For example, marginal inference in Bayesian networks [135] can be reduced to WMC, by encoding the conditional dependencies from the network through a propositional formula in CNF [146, 37], while, e.g., conjunctive query evaluation on probabilistic databases can be reduced to WMC, by computing the lineage representation of the input query, which yields a propositional formula in DNF [158]. The standard formulation of WMI assumes a formula in CNF as input, and to date, there is no study of WMI which is specifically tailored to formulas in DNF. This is surprising, because both variants relate to different inference problems which occur in various probabilistic data models.

Analogously to Chapter 6, we write WMI(CNF) and WMI(DNF) to distinguish between special cases of WMI, and follow a similar convention for WMC. These problems are obviously #P-hard for exact solving, as is model counting over CNF and DNF structures [168]. For approximate solving, however, there is a strong contrast in computational complexity between variants of weighted model counting problems: WMC(DNF) has a *fully polynomial randomised approximation scheme* (FPRAS) due to Karp, Luby and Madras [84], producing polynomial-time approximations with guarantees, whereas WMC(CNF) is known to be NP-hard to approximate, provided that RP $\neq$ NP [144]. The latter polynomial-time inapproximability result immediately propagates to WMI(CNF), while the approximability status of WMI(DNF)

remains *open*. In this chapter, we are interested in answering the following question: Does WMI(DNF) admit an FPRAS?

We answer this question in the affirmative for *concave* weight functions, and provide an FPRAS for WMI(DNF) with probabilistic accuracy guarantees. This result implies that WMI over DNF can be tractably approximated for a rich class of input problems. The intuition behind this result is based on two observations, corresponding to special cases of WMI(DNF). First, WMC (DNF), a special case of WMI(DNF) with no continuous variables, admits an FPRAS [84]. Second, the special case of WMI(DNF) with constant weight functions, and without any Boolean variables, corresponds to computing the *volume of unions of convex bodies*, and also admits an FPRAS [25]. Both of these approaches are based on the linear time coverage algorithm [111] (LTC, cf. Section 6.2), and are instrumental for our result.

Our algorithm, called ApproxWMI, builds on these classical algorithms, and extends them, by allowing extra constructs essential for WMI, while preserving the approximation guarantees. More concretely, our algorithm jointly handles Boolean and real variables, and uses a natural encoding of SMT constraints and a concave weight function. Essentially, our algorithm views DNF clauses analogously to convex bodies in volume computation [25], and handles Boolean variables analogously to Karp, Luby and Madras [84], all while incorporating the weight function via weighted sampling techniques [38]. Overall, this algorithm preserves the approximation guarantees provided by both approaches, and enables tractable approximations for WMI(DNF) over concave weight functions.

The rest of the chapter is organised as follows. Section 7.2 formally presents the problem of weighted model integration (WMI). In Section 7.3, we introduce ApproxWMI, an FPRAS for WMI(DNF), as well as some extensions, and prove their correctness. We empirically verify ApproxWMI in Section 7.4. We review the related work in Section 7.5, and conclude with a summary and discussions for future work in Section 7.6.

7.2 Weighted Model Integration

In this section, we introduce the weighted model integration problem. Hence, we first present satisfiability modulo theories (SMT).

Satisfiability modulo theories. The research field concerned with the satisfiability of formulas with respect to a background theory is called *satisfiability modulo theories*

(SMT) [66]. In this chapter, we are especially interested in SMT on quantifier-free formulas in the theory of *linear real arithmetic* (LRA).

Let $\mathbb{R}$ denote the real domain, and $\mathbb{B}$ denote the Boolean domain $\{0, 1\}$. Furthermore, let $\mathbf{X}$ denote a set of n real variables, and $\mathbf{V}$ denote a set of m Boolean variables. An *LRA atom* is of the form:

$$\sum_i \gamma_i x_i \bowtie \gamma,$$

where γ and γ_i are constant rational values, $x_i \in \mathbf{X}$, and $\bowtie \in \{<, \leq, >, \geq, =, \neq\}$ with their usual semantics. These atoms define constraints over the real domain.

For example, the LRA atoms $x_1 + x_2 \leq 4$, $x_1 \geq 0$, $x_2 \geq 0$ together restrict the real variables x_1 and x_2 to a two-dimensional triangle lying in the fully positive quadrant of $\mathbb{R}^2$. We write $\mathsf{atoms}(\mathbf{X}, \mathbf{V})$ to denote the set of atoms over $\mathbf{X} \cup \mathbf{V}$. As in propositional logic, a literal is either an atom or its negation. We sometimes write *LRA literal*, or a *Boolean literal*, to distinguish the literals depending on the domain of their corresponding variables.

An SMT formula φ is a propositional formula over $\mathbf{X}$ and $\mathbf{V}$, which is defined as a Boolean combination of literals via the logical connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ in the usual sense. We sometimes write $\varphi(\mathbf{X}, \mathbf{V})$ to denote the class of formulas over $\mathbf{X}$ and $\mathbf{V}$. Note that the CNF or DNF fragments of such formulas are defined as standard. If $\mathbf{V} = \emptyset$, then the domain is fully continuous, and we say that φ is an *LRA formula*. Otherwise, if $\mathbf{X} = \emptyset$ then φ corresponds to a standard (Boolean) propositional formula.

Example 7.2.1. *Consider the following formula:*

$$\varphi_{ex} \;=\; ((0 \leq x_1 \leq 5) \vee \neg p_1) \wedge (p_2 \vee \neg(10 \leq x_1 + x_2 \leq 15)).$$

This formula is defined using 2 Boolean and 2 LRA literals. If we drop p_1 and p_2, we obtain a formula over only LRA constraints.

Analogously to the Boolean setting, a *truth assignment* $\nu : \mathsf{atoms}(\mathbf{X}, \mathbf{V}) \mapsto \mathbb{B}$ for a propositional formula φ maps every atom to either 0 (false), or 1 (true), and *satisfies* φ in the usual sense. We sometimes say ν *propositionally satisfies* φ to make the underlying entailment relation explicit. However, truth assignments over $\mathsf{atoms}(\mathbf{X}, \mathbf{V})$ introduce a subtle difference relative to the standard Boolean definition. In particular, a propositional formula over $\mathbf{X}$ and $\mathbf{V}$ may have a propositionally satisfying truth assignment, but this assignment may yield inconsistency (i.e., infeasibility) with respect to the LRA constraints in the formula. We illustrate this with an example:

Example 7.2.2. *Recall the formula φ_{ex} from Example 7.2.1, and consider the assignment ν, where*

$$\nu(p_1) = 1, \quad \nu(p_2) = 0, \quad \nu(0 \leq x_1 \leq 5) = 1, \quad \nu(10 \leq x_1 + x_2 \leq 15) = 0.$$

It is easy to see that φ_{ex} is propositionally satisfiable. Observe that ν also satisfies the LRA constraints. Indeed, it implies that $0 \leq x_1 \leq 5$, and that either $x_1 + x_2 > 15$ or $x_1 + x_2 < 10$ holds. These constraints admit a solution, e.g., $x_1 = 2$, and $x_2 = 30$, so can be satisfied given ν. In contrast, consider the following formula:

$$\varphi_{ex'} = (1 \leq x_1 + x_2 \leq 4) \wedge \neg p_1 \wedge (x_1 \leq -2) \wedge (x_2 \leq 2).$$

This formula is trivially propositionally satisfiable by setting all atoms to true, and this satisfying assignment is unique. However, this assignment yields inconsistent LRA constraints, since $(x_1 \leq -2)$, $(x_2 \leq 2)$ and $(1 \leq x_1 + x_2 \leq 4)$ cannot be satisfied simultaneously, i.e., they yield an empty solution set over $\mathbb{R}^2$. Hence, the LRA constraints of $\varphi_{ex'}$ cannot be satisfied by any propositionally satisfying truth assignment.

We can now formally define the satisfiability of LRA atoms relative to a propositional truth assignment. A truth assignment ν is *LRA-satisfiable* if the solution space to the set of linear inequalities induced by ν is non-empty. Therefore, propositional formulas defined over $\mathbf{X}$ and $\mathbf{V}$ have 2 satisfiability criteria, namely conventional propositional satisfiability, and LRA satisfiability. In fact, the classical SMT problem over LRA is to determine, for a given propositional formula φ over $\mathbf{X}$ and $\mathbf{V}$, whether there exists an LRA-satisfiable assignment ν such that $\nu \models \varphi$.

Observe that for any $\varphi(\mathbf{X}, \mathbf{V})$, any choice of a real assignment $\mathbf{x}$ to $\mathbf{X}$ and a Boolean assignment $\mathbf{v}$ to $\mathbf{V}$, together yields an assignment to φ (since this choice sets every atom in the universe to true or false). Hence, throughout the paper, we consider propositional formulas $\varphi(\mathbf{X}, \mathbf{V})$ such that $\varphi : (\mathbb{R}^{|\mathbf{X}|} \times \mathbb{B}^{|\mathbf{V}|}) \mapsto \mathbb{B}$, and write $\varphi(\mathbf{x}, \mathbf{v}) \to 1$ to denote a satisfying assignment of φ.

Weighted model integration. As before, we assume a set of n real variables $\mathbf{X}$, and a set of m Boolean variables $\mathbf{V}$. We consider propositional formulas $\varphi(\mathbf{X}, \mathbf{V})$ such that $\varphi : (\mathbb{R}^n \times \mathbb{B}^m) \mapsto \mathbb{B}$, and weight functions $w(\mathbf{X}, \mathbf{V})$ such that $w : (\mathbb{R}^n \times \mathbb{B}^m) \mapsto \mathbb{R}^+$. In this setting, $\mathbf{x} \in \mathbb{R}^n$ denotes an assignment to real variables $\mathbf{X}$, and $\mathbf{v} \in \mathbb{B}^m$ denotes an assignment to Boolean variables $\mathbf{V}$.

Given a propositional formula φ and a weight function w over $\mathbf{X}$ and $\mathbf{V}$, the *weighted model integral* of φ over w is defined as:

$$\mathrm{WMI}(\varphi, w \mid \mathbf{X}, \mathbf{V}) = \sum_{\mathbf{v} \in \mathbb{B}^m} \int_{\mathbf{x} \in \mathcal{X}_{\varphi,\mathbf{v}}} w(\mathbf{x}, \mathbf{v}) d\mathbf{x}, \tag{7.1}$$

where $\mathcal{X}_{\varphi,\mathbf{v}} = \{\mathbf{x} \in \mathbb{R}^n \mid \varphi(\mathbf{x}, \mathbf{v}) \to 1\}$ denotes the set of *all* real valuations of $\mathbf{X}$ satisfying $\varphi(\mathbf{x}, \mathbf{v})$.

In this definition, the weight function w is very general, and can contain arbitrary dependencies between (and among) Boolean and real variables. Functions having such dependencies theoretically fall within the scope of WMI, but in practice are highly intractable, and thus are rarely adopted. Therefore, it is common to employ simplifying assumptions over w, so as to decompose and factorise it into more practically viable fragments. For WMC, a commonly used factorisation is to assign every Boolean variable its own independent weight, and formulate $w(\mathbf{v})$ as:

$$w(\mathbf{v}) = \prod_{p \in \mathbf{v}} w_b(p),$$

where $w_b(p)$ is the weight of Boolean literal p. An analogous factorisation is also used for WMI (c.f. Section 2.1 of [117]), namely:

$$w(\mathbf{x}, \mathbf{v}) = w_x(\mathbf{x}) \prod_{p \in \mathbf{v}} w_b(p), \tag{7.2}$$

where $w_x : \mathbb{R}^n \to \mathbb{R}$. Thus, the weight function in Equation (7.1) is typically factorised into a product of a real variable weight function w_x and a product of individual Boolean literal weights, and we refer to this as the *factorisation assumption*. In this chapter, we additionally assume that w_x and w_b all return *positive* values, that is $w_x : \mathbb{R}^n \to \mathbb{R}^+$ and $w_b : \mathbb{B} \to \mathbb{R}^+$. We make this restriction so as to align with volume computation (which necessarily returns a positive quantity) over real bodies, and to easily map the weight function over Boolean variables to a probability distribution to be used for sampling. Indeed, if the (positive) Boolean weights w_b assigned to each variable do not define a distribution, then a simple normalisation of literal weights for each Boolean variable p is sufficient to recover a distribution. We now illustrate WMI computation under this factorisation with an example.

Example 7.2.3. *Consider the formula:*

$$\varphi_{\mathrm{DNF}} \;=\; (p_1 \wedge (0 \leq x_1 \leq 5) \wedge \neg p_2) \vee (p_2 \wedge \neg(2 \leq x_1 \leq 4)).$$

In this example, the range of x_1 *is* $0 \leq x_1 \leq 10$*, and the weight function* $w(\mathbf{x}, \mathbf{v}) = x_1 \cdot w_b(p_1) \cdot w_b(p_2)$ *with* $w_b(p_1) = 0.6$*, and* $w_b(p_2) = 0.1$*. We can compute the weighted model integral over this formula and weight function as:*

$$\begin{aligned}\mathrm{WMI}(\varphi_{\mathrm{DNF}}, w \mid \mathbf{X}, \mathbf{V}) = 0.6 \cdot (1 - 0.1) \cdot \int_0^5 x_1 \; dx_1 \; + \\ 0.6 \cdot 0.1 \cdot \Big(\int_0^2 x_1 \; dx_1 + \int_4^{10} x_1 \; dx_1 \Big) \; + \\ (1 - 0.6) \cdot 0.1 \cdot \Big(\int_0^2 x_1 \; dx_1 + \int_4^{10} x_1 \; dx_1 \Big) = \; 11.15.\end{aligned}$$

Finally, weighted model integration is defined on fragments of propositional logic in the obvious way, i.e., WMI over DNF formulas (resp. CNF formulas) is the weighted model integration problem where the class of input formulas is restricted to formulas in DNF (resp., CNF).

Notice that *volume computation* of bodies over a high-dimensional space $\mathbb{R}^n$ is captured by weighted model integration as a special case. Indeed, by considering a uniform weight function defined exclusively over continuous LRA variables across $\mathbb{R}^n$, WMI reduces exactly to volume computation over *convex polytopes*. In fact, this problem is intractable to solve exactly, but, similarly to WMC(DNF), admits an FPRAS, which we build on to propose our approach for WMI. Therefore, we briefly present approaches for volume computation next.

Computing volumes of convex bodies. In this chapter, we build on a fully polynomial randomised approximation scheme (FPRAS) for volume computation over convex bodies [109]. Generally speaking, this problem is highly intractable to solve. Indeed, given an n-dimensional real domain $\mathbb{R}^n$, it is shown that no deterministic algorithm with a running time of $O(n^{cn}), c < \frac{1}{2}$, can even approximate the volume of convex bodies [12]. This intractability has led to the development of randomised alternatives, which have proven key for producing approximations in polynomial time. Following an initial breakthrough FPRAS [9], which has prohibitive practical computational complexity, several more efficient FPRAS algorithms have been proposed to compute volume of convex bodies. For this chapter, we will primarily refer to the FPRAS by Lovàsz and Vempala [109], which currently has the lowest computational complexity for this task. This FPRAS is highly involved, and we only use it as a sub-routine in our approach. Therefore, we limit ourselves to a high-level overview of the algorithm, and refer readers to the original paper for a detailed presentation.

The Lovàsz and Vempala FPRAS for convex body volume computation is based on multi-phase Monte Carlo. At every phase, the FPRAS approximates a ratio be-

tween the volume of two convex bodies, where one body is contained in the other. Phases start with large, simpler bodies, and in subsequent phases, volume ratios are approximately computed between ever smaller, more complex bodies, until the target convex body is reached in the final phase. For an n-dimensional space, the Lovàsz and Vempala FPRAS uses $O^*(n)$ phases, where the asterisk denotes suppressed logarithmic factors, and at every phase, random walk algorithms such as *hit-and-run* [38] are called over the convex bodies to approximate the ratio between their volumes. Since hit-and-run runs in $O^*(n^3)$, the overall volume computation FPRAS runs in time $O^*(n^4)$.

7.3 Approximating Weighted Model Integration over DNFs

In this section, we propose ApproxWMI, an algorithm for WMI(DNF), and prove that it is an FPRAS given a concave weight function w formulated using the factorisation assumption. ApproxWMI builds on ApproxUnion, an FPRAS for approximately computing the volume of unions of convex bodies [25] which is based on linear-time coverage (LTC) [111]. We first introduce ApproxUnion.

7.3.1 Computing the Volume of Unions of Convex Bodies

ApproxUnion is an FPRAS for computing the volume of the union of convex bodies. More formally, given k convex bodies $B_1, ..., B_k$, ApproxUnion returns an approximation of the volume of $\bigcup_{i=1}^{k} B_i$, denoted $\mathsf{Vol}(\bigcup_{i=1}^{k} B_i)$ with confidence $1-\delta$ and error ϵ. Fundamentally, ApproxUnion is based on LTC, similarly to KLM: it uses an iterative sampling procedure, consisting of (i) sampling points and (ii) verifying them against a uniformly sampled convex body. However, ApproxUnion additionally introduces an initial volume computation step, to estimate the volumes of all individual bodies, and allows *errors* in its computations. That is, volume computation, sampling and verification can return results with errors, so long as these errors can be made arbitrarily small. More specifically, ApproxUnion is an FPRAS if volume computation, sampling and verification are each an FPRAS. Hence, ApproxUnion tolerates the use of "weak" oracles producing approximate results, while maintaining probabilistic guarantees on its outputs. We now present ApproxUnion in detail.

Initially, ApproxUnion approximately computes the volume of every convex body, yielding estimates with multiplicative error ϵ_V relative to the exact body volume, using an oracle VolumeQuery. Unlike KLM, where clause probabilities can be

trivially computed, volume computation is a costly operation, which requires a dedicated FPRAS, e.g., Lovàsz and Vempala, to be approximated in polynomial time. Once all approximate convex body volumes are computed, ApproxUnion then performs T iterative sampling procedure steps to compute an estimator for the volume of the union of convex bodies. An ApproxUnion sampling iteration is as follows:

1. (**Sampling**) If no sample point τ exists, sample a body B_i with probability $\mathsf{Vol}(B_i)/\sum_{j=1}^{k}\mathsf{Vol}(B_j)$, and then approximately uniformly sample τ from B_i using an oracle SampleQuery with error ϵ_S. Otherwise, if τ already exists, proceed to Step 2.

2. (**Verification**) Check whether τ belongs to another uniformly randomly chosen body B' using another approximate oracle, PointQuery, with error ϵ_P. If $\tau \in B'$, a new point τ is sampled in Step 1, and N is incremented. Otherwise, τ is re-used in the next trial.

We observe two main differences relative to KLM. First, we note that sampling points in a convex body differs from assignment sampling for KLM clauses, in that (i) it is uniform across points in convex bodies, so does not assign distinct weights to different assignments and (ii) requires a dedicated query SampleQuery with its own error ϵ_S to perform sampling. Uniformity stems from the nature of volume computation, since all regions of a body contribute identically to its overall volume, whereas the need for a dedicated SampleQuery is due to the complexity of point sampling in an arbitrary convex body. Indeed, given the factorisation assumption, assignment sampling in KLM can be trivially performed through Bernoulli trials, whereas uniform point sampling in a convex body requires advanced subroutines in the general case.

The second main difference in ApproxUnion relative to KLM is the complexity of membership verification. More specifically, ApproxUnion verifies membership of a point relative to a convex body, which is non-trivial for general convex bodies, whereas KLM simply matches Boolean variables to target values.

Following T sampling iterations, ApproxUnion returns $T\sum_{j=1}^{k}\mathsf{Vol}(B_j)/kN$ as an estimate of $\mathsf{Vol}(\bigcup_{i=1}^{k}B_i)$. ApproxUnion is an FPRAS for the volume computation of a union of convex bodies under certain conditions, as stated in Theorem 2 of the original work [25], and these conditions are satisfied with closed-form bounds from Lemma 3 of the same work. More concretely, the following result holds:

Theorem 7.3.1 (Theorem 2 and Lemma 3, [25]). ApproxUnion *relative to oracles* VolumeQuery, SampleQuery, PointQuery *with errors* ϵ_V, ϵ_S, *and* ϵ_P *respectively, is an FPRAS for* $\mathsf{Vol}(\bigcup_{i=1}^{k} B_i)$ *with error* ϵ *and confidence* $\frac{1}{4}$ *using* $T = \frac{24\ln(2)(1+\tilde{\epsilon})k}{\tilde{\epsilon}^2-8(\tilde{C}-1)k}$ *iterations, with* $\tilde{\epsilon} = \frac{\epsilon-\epsilon_V}{1+\epsilon_V}$ *and* $\tilde{C} = \frac{(1+\epsilon_S)(1+\epsilon_V)(1+k\epsilon_P)}{(1-\epsilon_V)(1-\epsilon_P)}$, *for* ϵ_V, $\epsilon_S \leq \frac{\epsilon^2}{47k}$, *and* $\epsilon_P \leq \frac{\epsilon^2}{47k^2}$.

Furthermore, when this theorem holds, T is $O(\frac{k}{\epsilon^2})$. In summary, ApproxUnion generalises LTC to allow errors within sampling, membership checking, and volume computation, so long as these can be made arbitrarily small, and computes unions of continuous sets. However, ApproxUnion does *not* allow for confidence parameters in the oracles, but this can be trivially extended to allow for FPRAS oracles having confidence δ, as stated earlier, using standard tools of probability, as we now show in the following corollary.

Corollary 7.3.1. ApproxUnion *relative to FPRAS oracles* VolumeQuery, SampleQuery, *and* PointQuery *with errors* ϵ_V, ϵ_S, ϵ_P *and confidence values* δ_V, δ_S, δ_P, *respectively, is an FPRAS with error* ϵ *and confidence* δ *for* $\mathsf{Vol}(\bigcup_{i=1}^{k} B_i)$ *using* $T = \frac{8\ln(\frac{8}{\delta})(1+\tilde{\epsilon})k}{\tilde{\epsilon}^2-8(\tilde{C}-1)k}$ *iterations, for*

1. $\epsilon_V, \epsilon_S \leq \frac{\epsilon^2}{47k}$, $\epsilon_P \leq \frac{\epsilon^2}{47k^2}$, *and*
2. $\delta_V \leq \frac{\delta}{4k}$, $\delta_S + \delta_P \leq \frac{\delta}{2276\ln(\frac{8}{\delta})\frac{k}{\epsilon^2}}$.

Proof. We use the worst-case assumption that a single failure of any oracle, or a failure of the sampling procedure, implies the failure of ApproxUnion. Hence, we seek to upper-bound the union of these failure probabilities by δ to ensure that ApproxUnion is an FPRAS.

In Lemma 3 of the ApproxUnion paper [25], it is shown that Condition 1 is sufficient for reliable but weak oracles VolumeQuery, SampleQuery, and PointQuery, to ensure ApproxUnion meets the ϵ error requirement with probability $\frac{3}{4}$. Hence, we now need to prove that, given unreliable oracles satisfying Condition 2, and failure probability of the LTC sampling procedure generalised from $\frac{1}{4}$ to a value $\delta_1 < \delta$, ApproxUnion also meets the confidence requirement δ, and is therefore an FPRAS for computing the union of convex bodies.

The generalisation of the failure probability of sampling from $\frac{1}{4}$ to $\delta_1 < \delta$ can be achieved using standard probability amplification techniques, namely by multiplying the required number of trials T specified in Theorem 7.3.1 by $\frac{1}{3\ln(2)}\ln(\frac{2}{\delta_1})$, yielding T as specified in Corollary 7.3.1. We can now upper-bound the failure probability of

the LTC sampling procedure, denoted f_{LTC}, by setting $\delta_1 = \frac{\delta}{4}$, which yields the value of T shown in Corollary 7.3.1.

Given $\delta_1 = \frac{\delta}{4}$, a failure probability of $\frac{3\delta}{4}$ remains, and shall be allocated for all possible oracle failures. We now show that the confidence requirements stated in Condition 2 perform this allocation and indeed upper-bound any oracle failure probability by $\frac{3\delta}{4}$, as required:

Let f_V denote the failure of any call to VolumeQuery, f_S the failure of any call to SampleQuery, and f_P the failure of any call to PointQuery. Furthermore, let f denote the overall failure probability of ApproxUnion. Given that VolumeQuery is called k times within ApproxUnion, setting $\delta_V \leq \frac{\delta}{4k}$ yields, by the union bound:

$$\Pr(f_V) \leq k\delta_V = \frac{\delta}{4}.$$

Within ApproxUnion, PointQuery is called T times, whereas SampleQuery can be called up to T times, depending on the success of PointQuery at the previous iteration. We assume the worst-case and consider that SampleQuery is called T times. Applying the union bound with the bound on δ_P and δ_S as specified in Condition 2 yields:

$$\Pr(f_P \vee f_S) \;\leq\; T(\delta_S + \delta_P) \;\leq\; T\frac{\delta}{2276\ln(\frac{8}{\delta})\frac{k}{\epsilon^2}}.$$

We now use the bound from Lemma 3 [25], which gives that, under Condition 1, $T < 2365\frac{k}{\epsilon^2}$ for failure probability $\frac{1}{4}$. generalising this bound for an arbitrary δ_1 gives $T < \frac{2365}{3\ln(2)}\ln(\frac{2}{\delta_1})\frac{k}{\epsilon^2} < 1138\ln(\frac{8}{\delta})\frac{k}{\epsilon^2}$. Replacing T with this bound in the failure probability yields:

$$\Pr(f_P \vee f_S) \leq \frac{\delta}{2}.$$

Finally, using the union bound to upper-bound the overall ApproxUnion failure probability yields:

$$\Pr(f) \leq \Pr(f_V \vee f_P \vee f_S) + \Pr(f_{\text{LTC}}) \leq \frac{\delta}{4} + \frac{\delta}{2} + \frac{\delta}{4} = \delta,$$

as required. □

7.3.2 ApproxWMI

We now introduce ApproxWMI (Algorithm 1), an FPRAS for WMI(DNF) given a *concave* and *factorised* weight function w. More precisely, a function w is concave if $\forall x, y \in \mathbb{R}^n$, and $\lambda \in [0, 1]$,

$$\lambda w(x) + (1 - \lambda)w(y) \leq w(\lambda x + (1 - \lambda)y),$$

Algorithm 1 ApproxWMI for WMI(DNF).

Input: $\mathbf{X}$: A set of n real variables; $\mathbf{V}$: A set of m Boolean variables; φ: A DNF consisting of k clauses c_i, $i \in \{1, ..., k\}$; w: a concave factorised weight function.
Parameters: ϵ: Error, δ: Confidence.
Output: An (ϵ, δ)-approximation of WMI$(\varphi, w \mid \mathbf{X}, \mathbf{V})$.

1: $T \leftarrow \frac{8 \ln(\frac{8}{\delta})(1+\tilde{\epsilon})k}{\tilde{\epsilon}^2 - 8(\tilde{C}-1)k}$ ▷ T is $O(\frac{k}{\epsilon^2} \ln(\frac{1}{\delta}))$ [25]
2: $\delta_V \leftarrow \frac{\delta}{4k}$, $\delta_S \leftarrow \frac{\delta}{2276 \ln(\frac{8}{\delta}) \frac{k}{\epsilon^2}}$ ▷ Subroutine confidence parameters
3: **for** $a \leftarrow 1$ to k **do**
4: $\quad U_a \leftarrow$ ClauseWeight$(c_a, w, \mathbf{X}, \mathbf{V})[\frac{\epsilon^2}{47k},\ \delta_V]$ ▷ Clause weight computation
5: $U \leftarrow \sum_{a=1}^{k} U_a$
6: time $\leftarrow 0$, $\quad N \leftarrow 0$
7: **while** time $< T$ **do** ▷ Sampling trials
8: $\quad$ Randomly select c_i, $i \in \{1, ..., k\}$ with probability $\frac{U_i}{U}$
9: $\quad (\mathbf{x}, \mathbf{v}) \leftarrow$ Sample$(c_i, w, \mathbf{X}, \mathbf{V})[\frac{\epsilon^2}{47k},\ \delta_S]$
10: $\quad c_{\text{sat}} \leftarrow \mathit{false}$
11: $\quad$ **while** $\neg c_{\text{sat}}$ **do**
12: $\quad\quad$ Uniformly select c_j, where $j \in \{1, ..., k\}$
13: $\quad\quad$ time $\leftarrow$ time $+ 1$
14: $\quad\quad$ **if** time $\geq T$ **then** ▷ If T reached during trial
15: $\quad\quad\quad$ **return** TU/kN
16: $\quad\quad$ **if** Evaluate$(c_j, \mathbf{x}, \mathbf{v})$ **then** ▷ Membership checking
17: $\quad\quad\quad c_{\text{sat}} \leftarrow \mathit{true}$, $N \leftarrow N + 1$
18: **return** TU/kN

and w is factorised when it can be formulated using the factorisation assumption. Fundamentally, the factorisation assumption simplifies the initial computation of clause weights, and subsequent sampling from formula clauses, whereas concavity ensures that the weight function restricts the scope of any volume computation in ApproxWMI only to convex bodies, enabling the use of the corresponding FPRAS algorithms. We now provide an overview of ApproxWMI, and show that it is an FPRAS for WMI(DNF).

Overview of ApproxWMI. Given a DNF φ over $\mathbf{X}$ and $\mathbf{V}$ defined using propositional and LRA atoms, and a concave, factorised weight function w, ApproxWMI computes an (ϵ, δ)-approximation of WMI$(\varphi, w \mid \mathbf{X}, \mathbf{V})$. ApproxWMI is based on ApproxUnion, and extends its oracle functions to allow unreliable oracles, hybrid domains, and factorised concave weight functions over convex bodies.

First, ApproxWMI sets essential parameters, namely the number of sampling iterations T (line 1) and confidence targets for subsequent clause weight computation

Algorithm 2 Subroutines of ApproxWMI as functions ClauseWeight, Sample, and Evaluate.

1: **function** ClauseWeight$(c, w, \mathbf{X}, \mathbf{V})[\epsilon, \delta]$
2: $w_{\text{Bool}} \leftarrow \prod_{p \in \mathbf{v}_c} w_b(p)$ ▷ Weight based on set Boolean literals from c
3: Introduce a new variable d in $\mathbf{X}$ ▷ Include w via auxiliary variable
4: $\mathcal{X}'_c \leftarrow \mathcal{X}_c \wedge (\mathbf{x} \in \mathbb{R}^n, 0 \leq d \leq w_x(\mathbf{x}))$ ▷ $\mathcal{X}_c$ is the c-induced convex polytope
5: $w_{\text{Real}} \leftarrow \text{Volume}(\mathcal{X}'_c)[\epsilon, \delta]$ ▷ This body volume automatically factors in w_x
6: **return** $w_{\text{Bool}} \cdot w_{\text{Real}}$ ▷ Return product as clause weight
7:
8: **function** Sample$(c, w, \mathbf{X}, \mathbf{V})[\epsilon, \delta]$
9: $\mathbf{v} \leftarrow$ sample from $\mathbf{V}$ respecting $\mathbf{v}_c$ and w_b ▷ Bernoulli sampling from $\mathbf{v}_c$
10: Introduce a new variable d in $\mathbf{X}$ ▷ This variable allows for uniform sampling
11: $\mathcal{X}'_c \leftarrow \mathcal{X}_c \wedge (\mathbf{x} \in \mathbb{R}^n, 0 \leq d \leq w_x(\mathbf{x}))$
12: $\mathbf{x} \leftarrow \text{ConvexBodySampler}(\mathcal{X}'_c)[\epsilon, \delta]_{1:n}$ ▷ Drop last dimension
13: **return** $(\mathbf{x}, \mathbf{v})$ ▷ Return the samples as output
14:
15: **function** Evaluate(c, $\mathbf{x}$, $\mathbf{v}$)
16: Compute convex polytope $\mathcal{X}_c$ from c
17: **if** $\mathbf{x} \not\models \mathcal{X}_c$ **then return** *false* ▷ Polytope membership
18: **if** $\mathbf{v} \not\models c$ **then return** *false* ▷ Boolean satisfiability
19: **return** *true*

(i.e., the WMI analog of volume computation in ApproxUnion) and point sampling operations (line 2) based on overall target error ϵ, confidence δ and the number of clauses k. Then, ApproxWMI computes the weighted model integral for every clause in φ given w, using the function ClauseWeight (lines 3-4). ClauseWeight is analogous to VolumeQuery in ApproxUnion, but additionally considers the effect of w on the integral and supports discrete variables in $\mathbf{V}$. ClauseWeight yields estimates of individual clause WMI with error $\frac{\epsilon^2}{47}$ and confidence δ_V.

Following calls to ClauseWeight for all k clauses in φ, T sampling iterations are conducted to compute the LTC estimator (lines 7-18). To this end, the disjoint universe weight is initially computed (line 5), and the corresponding iteration counter *time* and success counter N are initialised (line 6). In a sampling trial, a random clause c_i is selected with probability proportional to its weight U_i, and then a point $(\mathbf{x}, \mathbf{v})$ is sampled from c_i according to w using the function Sample (lines 8-9). Using the factorisation assumption, Sample independently samples Boolean ($\mathbf{v}$) and real ($\mathbf{x}$) variables satisfying c_i. Afterwards, another clause c_j (possibly c_i), is uniformly chosen from φ (line 12), and the point $(\mathbf{x}, \mathbf{v})$ is checked for membership to c_j via the function Evaluate (line 16). Evaluate separately validates Boolean and real

components of the sampled point, and, in case of membership in both domains, the variable N is incremented (line 17).

We now explain the subroutines ClauseWeight, Sample, and Evaluate in detail. The pseudo-codes for these subroutines are shown in Algorithm 2.

ClauseWeight. This function returns an (ϵ, δ)-approximation of $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ for a given clause c and weight function w over the domains $\mathbf{X}, \mathbf{V}$. Owing to the factorisation assumption, ClauseWeight separately processes computation over $\mathbf{X}$ and $\mathbf{V}$. For the Boolean sub-domain, ClauseWeight computes w_{Bool} as the product of probabilities for all Boolean literals p appearing in c (line 2), analogously to the right-hand-side product described in Equation 7.2.

For the real sub-domain, ClauseWeight computes the integral of w_x over the convex polytope defined by the LRA constraints of c. At a high level, this is achieved by introducing the concave weight function over the polytope as an additional convex dimension, which creates a new $n+1$-dimensional convex body (lines 3-4), and subsequently computing the volume of this new body using existing volume computation tools to yield the weighted integral (line 5).

More specifically, let $\mathcal{X}_c$ denote the n-dimensional convex polytope induced by the LRA constraints of c. This body is fully determined by the LRA constraints and literals of c, but additionally covers the full domain of definition for all real variables not appearing in c. To compute the integral of w over $\mathcal{X}_c$, we introduce an additional $n+1^{\text{th}}$ dimension, represented by dummy variable d, and constrain this dimension by limiting its value to the range $0 \leq d \leq w_x(\mathbf{x})$, for $\mathbf{x} \in \mathbb{R}^n$. We denote the resulting body as $\mathcal{X}'_c$ (line 3). Observe that the introduction of d reduces the current weighted integral formulation into a standard volume computation. Indeed, w is incorporated into $\mathcal{X}'_c$ via d and its corresponding constraint, and it is clear that computing the integral of $\mathcal{X}'_c$ corresponds to computing the weighted integral of $\mathcal{X}_c$ given w. Hence, the weighted integral over $\mathcal{X}_c$ can be computed as a standard integral over $\mathcal{X}'_c$.

We also observe that the additional constraint on d is convex, since w_x is concave. Thus, $\mathcal{X}'_c$ is a convex body, since $\mathcal{X}_c$ is a convex polytope. Note however, that $\mathcal{X}'_c$ is not necessarily a polytope, as w_x can be an arbitrary concave function. In fact, $\mathcal{X}'_c$ is a polytope if and only if w_x is a linear function. Nonetheless, the general convexity of $\mathcal{X}'_c$ is sufficient, as it enables the computation of its volume using known FPRAS algorithms designed for convex bodies [109, 83], represented by the function Volume. Therefore, the output of Volume, w_{Real}, corresponds to the weighted integral of $\mathcal{X}_c$ given w. Finally, ClauseWeight returns the product of lines 2 and 5 as its estimate for $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ (line 6).

Sample. This function samples a point $(\mathbf{x}, \mathbf{v})$ over the domain $\mathbf{X} \cup \mathbf{V}$ from a given clause c and a weight function w. Based on the factorisation assumption, it separately performs sampling over Boolean and real sub-domains. For the Boolean sub-domain, a Boolean assignment $\mathbf{v}$ satisfying c is sampled by (i) setting all Boolean literals appearing in c to their required values and (ii) randomly sampling all remaining variables according to w_b (line 9), which corresponds to independent Bernoulli sampling for each variable based on its defined truth probability.

For the real sub-domain, an analogous transformation from $\mathcal{X}_c$ to a convex body $\mathcal{X}'_c$ as in ClauseWeight (lines 10-11) is used to incorporate w and reduce sampling to a uniform setting. Afterwards, Sample samples a point approximately uniformly from $\mathcal{X}'_c$, using standard convex body samplers such as hit-and-run [38], denoted by ConvexBodySampler (line 12). It then discards the $n + 1^{th}$ dimension of this point to yield an approximate sample $\mathbf{x}$ from $\mathcal{X}_c$ weighted according to w. Finally, Sample returns the outputs of lines 9 and 12 as a sample from c (line 13).

Evaluate. This function determines the membership of a point $(\mathbf{x}, \mathbf{v}) \in \mathbb{R}^n \times \mathbb{B}^m$ to a clause c. Specifically, it checks the membership of $(\mathbf{x}, \mathbf{v})$ to the polytope $\mathcal{X}_c$ defined by c in two steps: Evaluate first verifies the real component $\mathbf{x}$, i.e., that all the LRA constraints of c are satisfied (line 17), and then verifies the Boolean component $\mathbf{v}$ (line 18). If both conditions are met, then $(\mathbf{x}, \mathbf{v})$ satisfies c. Unlike the earlier two functions, Evaluate is deterministic, and its outputs have no attached uncertainty.

7.3.3 ApproxWMI is an FPRAS

We show the correctness of ApproxWMI, and prove that, under the error and confidence settings presented in Algorithm 1, it is an FPRAS for WMI(DNF) with concave weight functions w respecting the factorisation assumption. As ApproxWMI builds on ApproxUnion, and replaces its oracles with specific WMI analogs, the proof shows that all oracles in ApproxWMI satisfy the conditions of Corollary 7.3.1. Then, it shows that (i) all oracles correctly compute their target values, and that (ii) these oracles each meet the conditions of Corollary 7.3.1.

More concretely, the proof shows that ClauseWeight correctly computes an (ϵ_V, δ_V)-approximation of $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$, and meets the conditions of Corollary 7.3.1. Then, the proof shows that Sample is correct, by showing that uniform sampling from $\mathcal{X}'_c$ and discarding the final dimension is equivalent to sampling from $\mathcal{X}_c$ according to w, and establishes the sufficiency of running ConvexBodySampler with parameters ϵ_S and δ_S to satisfy 7.3.1. Finally, the correctness and satisfaction of

Corollary 7.3.1 for EVALUATE are trivially shown. We now formally state our result, and provide its complete proof.

Theorem 7.3.2. APPROXWMI *relative to FPRAS oracles* CLAUSEWEIGHT, SAMPLE, *and* EVALUATE *having error* ϵ_V, ϵ_S, ϵ_P *and confidence* δ_V, δ_S, δ_P, *respectively, is an FPRAS for* WMI(DNF) *over a concave and factorised weight function* w, *with error* ϵ *and confidence* δ *and using* $T = \frac{8\ln(\frac{8}{\delta})(1+\tilde{\epsilon})k}{\tilde{\epsilon}^2 - 8(\tilde{C}-1)k}$ *iterations, for*

1. $\epsilon_V, \epsilon_S \leq \frac{\epsilon^2}{47k}$, $\epsilon_P \leq \frac{\epsilon^2}{47k^2}$, *and*
2. $\delta_V \leq \frac{\delta}{4k}$, $\delta_S + \delta_P \leq \frac{\delta}{2276\ln(\frac{8}{\delta})\frac{k}{\epsilon^2}}$.

Proof. The difference between APPROXUNION and APPROXWMI lies in the implementation of task-specific oracles for WMI. Therefore, it is sufficient to show that all oracles in APPROXWMI satisfy the requirements of Corollary 7.3.1 to reach the desired result.

CLAUSEWEIGHT. We first show the correctness of CLAUSEWEIGHT for computing $\mathrm{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ given a concave and factorised w. Let $\mathbf{v} \models c$ be a Boolean assignment satisfying the Boolean literals of c, $\mathcal{X}_c$ be the convex polytope defined by the LRA constraints of c, and $\mathbf{v}_c$ be the set of Boolean literals appearing in c. Given a factorised w, WMI over c reduces to:

$$\begin{aligned}
\mathrm{WMI}(c, w \mid \mathbf{X}, \mathbf{V}) &= \sum_{\mathbf{v} \models c} \int_{\mathcal{X}_c} w(\mathbf{x}, \mathbf{v}) \; d\mathbf{x} \\
&= \sum_{\mathbf{v} \models c} \int_{\mathcal{X}_c} w_x(\mathbf{x}) \prod_{p \in \mathbf{v}} w_b(p) \; d\mathbf{x} \\
&= \sum_{\mathbf{v} \models c} \prod_{p \in \mathbf{v}} w_b(p) \int_{\mathcal{X}_c} w_x(\mathbf{x}) \; d\mathbf{x} \\
&= \Big(\prod_{p \in \mathbf{v}_c} w_b(p) \Big) \int_{\mathcal{X}_c} w_x(\mathbf{x}) \; d\mathbf{x}.
\end{aligned}$$

Note that the separation of the sum from the integral in the last step is possible because the body $\mathcal{X}_c$ is constant across all Boolean assignments, since c is a conjunction of atoms enforcing a unique set of real constraints. Hence, the WMI of c given a factorised, concave w amounts to a product computation of Boolean literal probabilities plus a weighted integral computation over $\mathcal{X}_c$ in the real domain $\mathbb{R}^n$. In CLAUSEWEIGHT, the product computation corresponds to line 2, whereas weighted integral computation corresponds to lines 4 and 5. Weighted integral computation is

performed using a transformation of $\mathcal{X}_c$ to $\mathcal{X}'_c$, as described in Section 7.3.2, and it is trivial to prove that the volume of $\mathcal{X}'_c$ is equivalent to the weighted integral of $\mathcal{X}_c$. Therefore, ClauseWeight indeed returns an approximation for $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$.

We now show that ClauseWeight meets the error and confidence criteria of Corollary 7.3.1. Multiplication of Boolean weights is error-free, hence it is necessary and sufficient that the call to Volume have error and confidence upper-bounded by ϵ_V and δ_V respectively, as is the case in Algorithm 1, for ClauseWeight to meet the requirements of Corollary 7.3.1.

Sample. We first show that sampling from the transformation result $\mathcal{X}'_c$ is equivalent to sampling from $\mathcal{X}_c$ according to w. Let $\mathbf{x}'$ be a real assignment sampled uniformly from $\mathcal{X}'_c \in \mathbb{R}^{n+1}$, and let $\mathbf{x}$ be $\mathbf{x}'$ with the last dimension omitted i.e., $x'_{1:n}$. The weight of $\mathbf{x}$ is therefore given by:

$$\int_0^{w_x(\mathbf{x})} 1 \; d\mathbf{x}_{n+1} \sim w_x(\mathbf{x}),$$

as required. This implies that, when $\mathcal{X}'_c$ is successfully sampled with multiplicative error ϵ_S, $\mathbf{x}$ is sampled over $\mathcal{X}_c$ given weight function w with the same error. We now show that Sample meets the requirements of Corollary 7.3.1. Boolean variable sampling, corresponding to line 9, occurs with zero error. Hence, it is necessary and sufficient to run ConvexBodySampler with parameters ϵ_S and δ_S satisfying Condition 2 of Corollary 7.3.1, to ensure Sample meets the requirements. This is indeed the case with Sample, since ConvexBodySampler is called with parameters ϵ_S and δ_S, as specified in Algorithm 1, and these parameters meet Condition 2. In particular, $\delta_P = 0$, since Evaluate is deterministic, hence $\delta_S = \frac{\delta}{2276 \ln(\frac{8}{\delta}) \frac{k}{\epsilon^2}}$ is the largest δ_S satisfying Condition 2.

Evaluate. This function is deterministic, so trivially meets error and confidence requirements. The function is also clearly correct.

Hence, ApproxWMI is an FPRAS for WMI given a concave and factorised w and given FPRAS functions ClauseWeight and Sample, and the deterministic Evaluate. □

As with ApproxUnion, ApproxWMI does not rely on specific configurations of its oracles for correctness, but only on their approximation properties. Indeed, ApproxWMI can be run with exact oracles, or with approximate weak oracles where error and confidence can be made arbitrarily small, and will provide probabilistic guarantees in both scenarios. The approximation ability of ApproxUnion is only

affected once weak oracles cannot be made arbitrarily accurate (e.g., heuristics), which would bound the algorithm to minimum error values, or eliminate guarantees entirely.

In terms of running time, AppроxWMI strongly depends on the running time of its chosen oracles, as well as the required error and confidence for its guarantees. More specifically, for ClauseWeight runtime r_C, Sample runtime r_S, and Evaluate runtime r_E, AppроxWMI runs in $O(k \cdot r_C + T(r_S + r_E))$. Though desired error ϵ and confidence δ directly affect this running time via the number of sampling iterations T, they can also indirectly affect approximate oracle runtimes, leading to even higher computational requirements. Nonetheless, when every oracle is an FPRAS, AppроxWMI runs in polynomial time, yields the required guarantees, and thus is itself an FPRAS.

We now concretely illustrate the running time of AppроxWMI using the most general setting of its oracles, allowing for arbitrary convex bodies, and for concave and factorised w. In this setting, we can use the algorithm of Lovàsz and Vempala [109] as the Volume oracle in ClauseWeight, and hit-and-run for ConvexBodySampler [38]. These choices make ClauseWeight run in time $O^*(m + n^4(\frac{1}{\epsilon_V})^2)$, Sample run in time $O^*(m + n^3(\frac{1}{\epsilon_S})^2)$, where m is the number of Boolean variables, n is the number of real variables, and * denotes suppressed logarithmic factors (including, in particular, confidence terms). Since Evaluate runs in $O(m + Wn)$, where W is the width of a conjunction c, AppроxWMI therefore runs in time:

$$O^*\Big(km + kn^4(\frac{1}{\epsilon_V})^2 + Tm + Tn^3(\frac{1}{\epsilon_S})^2 + T(m + Wn)\Big).$$

Using the bounds $T = O(\frac{k}{\epsilon^2}\ln(\frac{1}{\delta}))$, $\epsilon_V = O(\frac{\epsilon^2}{k})$, and $\epsilon_S = O(\frac{\epsilon^2}{k})$ from Theorem 7.3.2, we obtain the complexity:

$$O^*\Big(km + \frac{k^3}{\epsilon^4}n^4 + \frac{k}{\epsilon^2}m + \frac{k^3}{\epsilon^6}n^3 + \frac{k(Wn+m)}{\epsilon^2}\Big) = O^*\Big(\frac{k^3}{\epsilon^4}n^4 + \frac{k^3}{\epsilon^6}n^3 + \frac{k}{\epsilon^2}(m + Wn)\Big).$$

Note that the time complexity of Sample can be reduced by restricting the class of bodies and weight functions used. Indeed, if w is linear, e.g., $3x_1 + 2x_2 - x_4$, then all bodies can be sampled approximately using optimised polytope sampling methods, e.g., geodesic walks [99], which run significantly faster for bodies defined with a small number of LRA constraints. Further to this, box-shaped bodies, defined with LRA constraints having only one variable, and a constant, uniform w, are an instance of unweighted model integration, and can be trivially sampled. Nonetheless, we assume the general convex case in this work, and build our algorithm accordingly.

In addition to naturally running over standard WMI formulations with LRA constraints, AppроxWMI can also apply to WMI instances defined with other theories,

so long as these theories define convex bodies in individual DNF clauses. For instance, ApproxWMI also applies to WMI with *concave* polynomial constraints over real variables, a theory strictly subsumed by the theory of non-linear real arithmetic (NRA), but which strictly contains LRA. Hence, ApproxWMI can capture a richer class of WMI formulations, at no additional computational cost.

Nonetheless, ApproxWMI is limited to concave, factorised weight functions by construction: Concavity is needed to ensure the convexity of the intermediate bodies, and the factorisation assumption is essential to computations in ClauseWeight and Sample. Unfortunately, relaxing the concavity requirement does not currently look promising, as we run into intractable computational tasks, such as arbitrary body volume computation, which typically do not have computationally feasible approximation schemes. Therefore, in our subsequent study of WMI, we focus on generalising ApproxWMI in a tractable fashion, for instance through relaxing the common factorisation assumption, so as to tackle more general weight functions allowing dependencies between Boolean and real variables.

7.3.4 Extending ApproxWMI: ApproxWMI$_{\mathbf{D}}$

In Section 7.3.2, we presented ApproxWMI, an FPRAS for WMI over DNF formulas given a factorised and concave weight function w. The factorisation of w simplifies WMI, in that (i) it makes Boolean variable weights independent from real variables, and (ii) simplifies the joint distribution over Booleans to a product. However, this factorisation prevents any interaction between Boolean and real variables, which limits the representation power of the weight function. Even simple weight functions cannot be captured using the factorisation assumption, as we illustrate next.

Example 7.3.1. *Consider* $\mathbf{X} = \{x_1\}$ *and* $\mathbf{V} = \{p_1\}$*. Then, the weight function*

$$w(\mathbf{x}, \mathbf{v}) = \begin{cases} 2x_1 & \text{if } p_1 \\ 1 & \text{otherwise,} \end{cases}$$

cannot be captured with the factorisation assumption, as the real component weight changes based on $\mathbf{v}$*.*

To alleviate the limitations of this factorisation, it has been proposed to define separate weight functions over *mutually exclusive* Boolean partitions of the problem domain, and subsequently solve the WMI problem separately over each of these partitions [117]. For instance, in our example, one can define two weight functions

$w_0(\mathbf{x}) = 2x_1$ and $w_1(\mathbf{x}) = 1$, corresponding to the different truth values of each possible Boolean assignment, solve the corresponding problems and subsequently aggregate their results. Through this partitioning scheme, the hybrid domain is separated into subspaces where dependencies do not exist, and therefore where the factorisation assumption applies. However, this partitioning can produce up to $2^{|\mathbf{V}|}$ sub-problems, and that makes this separation approach intractable in practice.

In this section, we extend the applicability of ApproxWMI to more general weight functions allowing for such dependencies, while maintaining tractability. More concretely, we study weight functions:

$$w(\mathbf{x}, \mathbf{v}) = w_x(\mathbf{x}, \mathbf{v}) \prod_{p \in \mathbf{v}} w_b(p), \tag{7.3}$$

where w_x also depends on Boolean variables (unlike Equation 7.2), allowing the aforementioned dependencies.

First, we note that this factorisation is very general, as the w_x term alone is sufficient to capture any arbitrary weight function, and therefore it can clearly capture the function $w(\mathbf{x}, \mathbf{v})$ presented in Example 7.3.1. In fact, it is only a factorisation in that it maintains independence between Boolean variables, and thus reduces the joint distribution over these variables to a product distribution, which simplifies their sampling.

In this more general setting, we build on ApproxWMI, and propose an FPRAS ApproxWMI$_{\mathrm{D}}$ that naturally incorporates the additional dependencies supported by Equation 7.3. ApproxWMI$_{\mathrm{D}}$ modifies clause weight computation to consider multiple different randomly sampled Boolean assignments per clause, and aggregates results for these assignments to yield an approximate overall clause weight. It also extends point sampling to respect the different distributions which distinct Boolean variable assignments may impose.

Prior to introducing ApproxWMI$_{\mathrm{D}}$, we first introduce some key notation and concepts. As earlier, Let $\mathcal{X}_c \subseteq \mathbb{R}^n$ be the convex polytope induced by the LRA constraints of a clause c, $\mathbf{v} \in \mathbb{B}^m$ be a Boolean assignment, and w_x be the hybrid component of weight function w, defined following Equation 7.3. We now define two functions a and b that map all subsets of $\mathbb{R}^n$ to positive real values. Intuitively, these functions will act as lower and upper bounds for the integral of w_x such that, for any subset of $\mathbb{R}^n$, the integral of w_x over this subset will be bounded by a and b, for all possible Boolean assignments.

In defining a and b, we aim to capture the *variability* of w_x relative to changes in the Boolean domain. This variability, introduced by our generalisation of w, directly

impacts the sample complexity needed to approximate the weight of a given clause, and thus we define a and b to threshold the range of w_x, and maintain a tractable number of random samples needed for clause weight computation in $\textsc{ApproxWMI}_{\mathrm{D}}$.

Formally, we define $a, b : P(\mathbb{R}^n) \mapsto \mathbb{R}^+$, where P denotes the power set operation, such that

$$a(\mathcal{X}_c) = \min_{\mathbf{v}} \int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) \; d\mathbf{x}, \text{ and}$$
$$b(\mathcal{X}_c) = \max_{\mathbf{v}} \int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) \; d\mathbf{x}.$$

We first note that the functions a and b are guaranteed to exist, as the Boolean domain $\mathbb{B}^m$ only defines a finite set of 2^m distinct possible values for the integral of $w_x(\mathbf{x}, \mathbf{v})$, and any finite set admits maximum and minimum values. We also note that a and b are finite-valued since, by construction, all integral computations in our setting of WMI are finite-valued, as continuous variables are bounded. As a result, there exist lower and upper bound functions a and b for the integral of any function w_x (with finite density over $\mathbb{R}^n$), over any subset of $\mathbb{R}^n$, in particular a convex polytope $\mathcal{X}_c$. This translates to the following statement:

$$\forall \mathcal{X}_c \in P(\mathbb{R}^n), \exists a, b, \forall \mathbf{v} \in \mathbb{B}^m : P(\mathbb{R}^n) \mapsto \mathbb{R}^+, a(\mathcal{X}_c) \leq \int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) \; d\mathbf{x} \; \leq \; b(\mathcal{X}_c). \tag{7.4}$$

Using a and b, we now define the maximum ratio ρ, to better study the variability of w_x:

$$\rho = \max_{c} \frac{b(\mathcal{X}_c)}{a(\mathcal{X}_c)}.$$

Intuitively, ρ implicitly produces a range of integral values for w_x given a convex polytope $\mathcal{X}_c$. More concretely, if w_x yields an integral value I for $\mathcal{X}_c$ for a given Boolean assignment $\mathbf{v}$, then w_x cannot produce integrals over $\mathcal{X}_c$ with a different assignment $\mathbf{v}'$ that are larger than ρI, or smaller than $\frac{I}{\rho}$. ρ correlates directly with Boolean-real dependency. In fact, the special case $\rho = 1$, its minimum possible value, corresponds to independence between Boolean and real variables, and thus to w_x defined using the factorisation assumption.

As with a and b, we define ρ to estimate the sample complexity needed for approximating clause weights under the factorisation of Equation 7.3. In fact, we show that this sample complexity depends quadratically on ρ, and therefore that, for any w, ρ

must be upper-bounded by a polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, n, m, and k, to maintain polynomial sample complexity for clause weight computation within $\textsc{ApproxWMI}_{\text{D}}$. In what follows, we refer to weight functions w satisfying this criterion as *ρ-restricted*.

In spirit, our definition and restriction of ρ resembles the restriction of *tilt* θ in approximation methods for weighted model counting [31]. For those approximate WMC approaches, tilt imposes a maximum ratio between the minimum and maximum *values* of the weight function over the Boolean domain, with the analogous objective of limiting variability. However, such restrictions on θ are *tighter* than restrictions on ρ. Indeed, ρ is based on the ratio between the maximum integral and minimum integral of a function over a *fixed* real body, and so does not compare between integral values over different bodies. By contrast, tilt is based on a ratio over function values at potentially different points. As a result, when θ is set to a value H, it naturally follows that ρ is also upper-bounded by H, whereas the opposite direction does not hold, yielding the inequality $\rho \leq \theta$. We now illustrate the difference between ρ and θ with two examples.

Example 7.3.2. *Let* $\mathbf{V} = \{p_1\}, \mathbf{X} = \{x_1\}, 0 \leq x_1 \leq 10$*, and let*

$$w_x(\mathbf{x}, \mathbf{v}) = \begin{cases} 3 & \text{if } p_1 \\ 2 & \text{otherwise} \end{cases}$$

Then, $\rho = 1.5$*, and* $\theta = 1.5$*, and thus the bound* $\rho \leq \theta$ *is tight. More generally,* $\rho = \theta$ *holds for all* w_x *defined as piece-wise constants over* $\mathbf{V}$*.*

In Example 7.3.2, we consider a piece-wise constant function over $\mathbf{V}$, and observe that, in this case, both ratio ρ and tilt θ are equal to 1.5: For tilt, the maximum and minimum values of w_x are 3 and 2 respectively, yielding the ratio 1.5, whereas $\rho = 1.5$ since the maximum integral ratio over any body necessarily involves alternating the value of p_1. Hence, the inequality $\rho \leq \theta$ is tight. However, piece-wise constant functions are very limited in their representation capacity. By contrast, for functions with variations for a fixed Boolean assignment, the scenario changes completely, as Example 7.3.3 shows.

Example 7.3.3. *Let* $\mathbf{V} = \{p_1\}, \mathbf{X} = \{x_1\}, 0 \leq \mathbf{x} \leq 10$*, and let*

$$w_x(\mathbf{x}, \mathbf{v}) = \begin{cases} \frac{e^{10}}{1+e^{-x_1}} - 0.5e^{10} + 1 & \text{if } \mathbf{v} \\ \frac{2e^{10}}{1+e^{-x_1}} - e^{10} + 2 & \text{otherwise} \end{cases}$$

Then, $\rho = 2$*, but* $\theta \approx 2 + e^{10}$*. Though* w_x *for* $p_1 = 1$ *is simply half the other case, the variability within every case is significant, and alone produces a tilt of* $1 + 0.5e^{10}$*. By contrast,* ρ *is not affected by this variability.*

In this example, ρ is upper-bounded by 2, as the case where p_1 is false doubles w_x relative to when p_1 is true, and hence doubles the integral for any body over which integration is performed. However, the tilt θ of w_x is almost equal to $2+e^{10}$, since the maximum of w_x over its range of definition is $w_x(10, \mathit{false}) = \frac{2e^{10}}{1+e^{-10}} - e^{10} + 2 \approx e^{10} + 2$, and its minimum is $w_x(0, \mathit{true}) = \frac{e^{10}}{1+e^{-0}} - 0.5e^{10} + 1 = 1$. In contrast with piece-wise constant functions, w_x in Example 7.3.3 has large variability in its cases, and this raises tilt, but not does affect the ratio ρ. Hence, ρ is less vulnerable to case-specific function variability than θ, and is therefore a less restrictive parameter.

Theoretically, ρ helps assess the viability of clause weight approximation using sampling techniques. However, its definition as a maximum over all possible subsets of $\mathbb{R}^n$ may not be amenable to practical computation, in which case, a simpler quantity ρ', evaluated over points in $\mathbb{R}^n$, rather than subsets, can be used instead. Fundamentally, ρ' is defined as the ratio between the maximum $w(\mathbf{x}, \mathbf{v})$ and the minimum $w(\mathbf{x}, \mathbf{v}')$ over all Boolean assignments. Note that these extrema are defined relative to an *identical* real assignment $\mathbf{x}$, but potentially distinct $\mathbf{v}$ and $\mathbf{v}'$. More formally,

$$\rho' = \max_{\mathbf{x}} \frac{w_{\max}(\mathbf{x})}{w_{\min}(\mathbf{x})}, \tag{7.5}$$

where $w_{\min}(\mathbf{x}) = \min_{\mathbf{v}} w_x(\mathbf{x}, \mathbf{v})$ and $w_{\max}(\mathbf{x}) = \max_{\mathbf{v}} w_x(\mathbf{x}, \mathbf{v})$.

Observe that tilt θ is also more restrictive than ρ', as its objective, though similar to that of ρ', uses *distinct* assignments $\mathbf{x}$ and $\mathbf{x}'$ in the numerator and denominator of Equation 7.5 respectively. Hence, $\rho' \leq \theta$, and this inequality is tight by Example 7.3.2, as $\rho' = 1.5$ as well. Furthermore, since ρ' also restricts its ratio to identical real assignments, similarly to ρ, it is also less vulnerable to variability within fixed Boolean cases, and can be significantly smaller than θ. This can be seen in Example 7.3.3, where $\rho' = 2$. Relative to ρ, we note that $\rho \leq \rho'$, as no integral over any subset of $\mathbb{R}^n$ can exceed the ratio ρ', given that weight functions return strictly positive values. This yields the overall inequality $\rho \leq \rho' \leq \theta$.

We now introduce ApproxWMI$_{\mathrm{D}}$, an extension to the ApproxWMI FPRAS presented in Section 7.3.2, that supports weight functions w factorised according to Equation 7.3. At a high level, ApproxWMI$_{\mathrm{D}}$ operates identically to ApproxWMI, but replaces the oracles ClauseWeight and Sample used in Algorithm 1, with ClauseWeight$_{\mathrm{D}}$ and Sample$_{\mathrm{D}}$, respectively, such that these oracles now incorporate Boolean-real dependencies. Otherwise, all algorithmic details of ApproxWMI, namely the sampling iteration structure based on LTC, and clause membership checking via Evaluate, remain the same.

Algorithm 3 Subroutines of ApproxWMI$_{\mathrm{D}}$ as functions ClauseWeight$_{\mathrm{D}}$ and Sample$_{\mathrm{D}}$.

function ClauseWeight$_{\mathrm{D}}(c, w, \mathbf{X}, \mathbf{V})[\epsilon, \delta]$
  $\epsilon_{\mathrm{Samp}}, \epsilon_{\mathrm{Comp}} \leftarrow \frac{\epsilon}{1+\sqrt{2}}$, $\delta_{\mathrm{Samp}} \leftarrow \frac{\delta}{2}$
  $s \leftarrow \ln(\frac{2}{\delta_{\mathrm{Samp}}})\frac{1}{2\epsilon^2_{\mathrm{Samp}}}\rho^2$ ▷ # of sampling trials
  $\delta_{\mathrm{Comp}} \leftarrow \frac{\delta}{2s}$
  for $i \leftarrow 1$ to s **do** ▷ Perform Monte-Carlo sampling over clause weights
    $\mathbf{v}_i \leftarrow$ sample from $\mathbf{V}$ respecting $\mathbf{v}_c$ and w_b
    Introduce a new variable d in $\mathbf{X}$
    $\mathcal{X}'_c \leftarrow \mathcal{X}_c \wedge (\mathbf{x} \in \mathbb{R}^n, 0 \leq d \leq w_x(\mathbf{x}, \mathbf{v}_i))$ ▷ Create convex body using w_x
    $X_i \leftarrow$ Volume$(\mathcal{X}'_c)[\epsilon_{\mathrm{Comp}}, \delta_{\mathrm{Comp}}]$ ▷ Volume is the Monte-Carlo sample
  BoolWeight $\leftarrow \prod_{p \in \mathbf{v}_c} w_b(p)$
  return BoolWeight $\cdot \frac{1}{s}\sum_{i=1}^{s} X_i$ ▷ Return sample mean as weight estimate
function Sample$_{\mathrm{D}}(c, w, \mathbf{X}, \mathbf{V})[\epsilon, \delta]$
  $\mathbf{v} \leftarrow$ sample from $\mathbf{V}$ respecting $\mathbf{v}_c$ and w_b ▷ Boolean sampling
  Introduce a new variable d in $\mathbf{X}$
  $\mathcal{X}'_c \leftarrow \mathcal{X}_c \wedge (\mathbf{x} \in \mathbb{R}^n, 0 \leq d \leq w_x(\mathbf{x}, \mathbf{v}))$
  $\mathbf{x} \leftarrow$ ConvexBodySampler$(\mathcal{X}'_c)[\epsilon, \delta]_{1:n}$ ▷ Perform sampling on $\mathcal{X}'_c$
  return $(\mathbf{x}, \mathbf{v})$

Intuitively, ClauseWeight$_{\mathrm{D}}$ considers Boolean-real dependencies via *Monte-Carlo sampling* over the Boolean sub-domain, and subsequently averaging volume computations across different samples to yield an overall estimate of the clause weight. At this stage, ρ-restriction plays a key role, as this allows for an (ϵ_V, δ_V)-approximation for the oracle to be computable using a polynomial number of Boolean samples and calls to the volume computation FPRAS. On the other hand, Sample$_{\mathrm{D}}$ only marginally changes relative to Sample, in that Boolean assignments are initially sampled, but real assignment sampling is now conditioned on these Boolean assignments, rather than done independently and in parallel. The pseudo-code for these two oracles is provided in Algorithm 3. We now discuss these oracles in detail.

ClauseWeight$_{\mathrm{D}}$. To exactly compute the weight of a clause c under the factorisation setting of Equation 7.3, one must iterate over all satisfying Boolean assignments of c, compute scores for each assignment using a volume computation tool, similarly to ClauseWeight, and finally return a weighted average of these scores as the WMI of c. This procedure involves an exponential number of calls to a volume oracle, which is computationally prohibitive. Therefore, ClauseWeight$_{\mathrm{D}}$ performs Monte-Carlo sampling over the set of Boolean assignments to estimate the WMI of c.

Note that, in this setup, ClauseWeight$_{\mathrm{D}}$ has two sources of error and failure, namely, (i) the Monte-Carlo sampling procedure itself, which has error ϵ_{Samp} and confidence δ_{Samp}, and (ii) the error and confidence of the volume computation tool, which we will denote as ϵ_{Comp} and δ_{Comp} respectively. Hence, ClauseWeight sets these parameters such that their combination yields error and confidence bounds within the overall (ϵ_V, δ_V) requirement.

We now present the outer sampling loop. In ClauseWeight$_{\mathrm{D}}$, sampling is conducted for s sampling trials, where s is a function of ϵ_{Samp} and δ_{Samp}, as well as ρ (cf. Algorithm 3 for the closed-form equation of s). Within every trial, Volume is called with error ϵ_{Comp} and confidence δ_{Comp}. More specifically, each sampling iteration consists of randomly sampling a Boolean assignment $\mathbf{v}_i$ satisfying c according to w_b (line 6), computing the induced weight function $w_x(\mathbf{x}, \mathbf{v}_i)$ by fixing the Boolean variables, and using this induced function to obtain the transformed convex body $\mathcal{X}'_c$ (line 7), analogously to Sample and ClauseWeight. At this point, ClauseWeight$_{\mathrm{D}}$ behaves identically to ClauseWeight: It computes the weighted integral over c using a volume computation subroutine Volume (line 8).

Following all s sampling steps, the average of all samples is computed, multiplied by the product of all $\mathbf{v}_c$ literal weights, and returned (lines 9 and 11). The left-hand side of the returned value is identical to that of ClauseWeight, and reflects the continued independence of individual Boolean variables in this new factorisation, whereas the right-hand side completes the approximation of $\mathrm{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ and incorporates the w_x term via the sampling average weight.

Sample$_{\mathrm{D}}$. This function is defined almost identically to Sample in ApproxWMI, with the only difference being that $w_x(\mathbf{x}, \mathbf{v})$ is now induced from $\mathbf{v}$ (line 16) *prior* to applying the transformation. This is because w_x also depends on Boolean variables in this setting. Unlike Sample, where Boolean and real variable sampling can be done in any order, it is necessary for Boolean sampling to run first in Sample$_{\mathrm{D}}$, so as to condition w_x on the Boolean sample output and subsequently sample from $\mathcal{X}_c$ according to the induced weight function $w_x(\mathbf{x}, \mathbf{v})$.

7.3.5 ApproxWMI$_{\mathrm{D}}$ is an FPRAS

We now show that ApproxWMI$_{\mathrm{D}}$ is an FPRAS for WMI(DNF), by lifting the result given in Theorem 7.3.2. First, we show the correctness of ClauseWeight$_{\mathrm{D}}$, and prove that it is an FPRAS for $\mathrm{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ over concave, ρ-restricted w factorised

according to Equation 7.3, given sampling and volume computation operations whose error and confidence can be made arbitrarily small.

The correctness proof for $\textsc{ClauseWeight}_{\text{D}}$ relies on the correctness of Monte-Carlo sampling for producing approximations with probabilistic guarantees, and its polynomial-time tractability is shown with a probabilistic argument, proving that the $\textsc{ClauseWeight}_{\text{D}}$ sampling procedure only requires a polynomial number of samples given the ρ-restriction of w_x. In particular, given a clause c and its convex polytope $\mathcal{X}_c$, this proof establishes that all random samples are necessarily bound by the functions $a(\mathcal{X}_c)$ and $b(\mathcal{X}_c)$, as per Equation 7.4. Then, since these samples are independent and identically distributed, their average approximates the expected value of the sampled distribution with probabilistic guarantees from the Hoeffding bound. These guarantees, complemented by the ρ-restriction of w_x, yield a polynomial lower bound for the number of samples s. Hence, the sampling procedure can have arbitrarily small error, and introduces a worst-case polynomial overhead to $\textsc{ClauseWeight}_{\text{D}}$.

Finally, the proof shows that $\textsc{ClauseWeight}_{\text{D}}$ produces an (ϵ, δ)-approximation of $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ given the error and confidence values for sampling and volume computation shown in Algorithm 3, as these values, once aggregated across the operation of $\textsc{ClauseWeight}_{\text{D}}$, remain below ϵ and δ respectively, and therefore, $\textsc{ClauseWeight}_{\text{D}}$ is an FPRAS as required. We now formally state this result, along with its necessary conditions, as a Lemma, and provide its complete proof.

Lemma 7.3.1. $\textsc{ClauseWeight}_{\text{D}}$ *relative to Monte-Carlo sampling error* ϵ_{Samp} *and confidence* δ_{Samp}*, and an FPRAS* Volume *with error* ϵ_{Comp} *and confidence* δ_{Comp}*, is an FPRAS for* $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$*, where* c *is a conjunctive clause and* w *is concave,* ρ*-restricted, and factorised according to Equation 7.3, with error* ϵ*, confidence* δ*, and using* $s = \ln(\frac{2}{\delta_{Samp}})\frac{1}{2\epsilon^2_{Samp}}\rho^2$ *iterations, for*

1. $\epsilon_{Samp}, \epsilon_{Comp} \leq \frac{\epsilon}{1+\sqrt{2}}$,
2. $\delta_{Samp} \leq \frac{\delta}{2}$, *and*
3. $\delta_{Comp} \leq \frac{\delta}{2s}$.

Proof. We first show the correctness of $\textsc{ClauseWeight}_{\text{D}}$ for computing $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$. Under the factorisation of Equation 7.3, w_x now depends on both real and Boolean variables. Hence, the decomposition of WMI computation into two separate Boolean

and real parts used in the proof of Theorem 7.3.2 no longer holds. Instead, WMI computation over a clause c given w can only be written as:

$$\begin{aligned}\mathrm{WMI}(c, w \mid \mathbf{X}, \mathbf{V}) &= \sum_{\mathbf{v} \models c} \prod_{p \in \mathbf{v}} w_b(p) \int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) \; d\mathbf{x} \\ &= \Big(\prod_{p \in \mathbf{v}_c} w_b(p) \Big) \sum_{\mathbf{v} \models c} \Big(\prod_{p \in \mathbf{v} \setminus \mathbf{v}_c} w_b(p) \int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) \; d\mathbf{x} \Big),\end{aligned} \tag{7.6}$$

where $\mathbf{v}_c$ denotes the set of all Boolean literals appearing in c.

Note that the outer summation no longer simplifies as in Theorem 7.3.2, as the inner integral now depends on the Boolean assignment $\mathbf{v}$. Therefore, WMI computation over c in this setting requires $2^{m-|\mathbf{v}_c|}$ calls to a weighted volume computation tool, which is simulated by calling the VOLUME FPRAS on $\mathcal{X}'_c$, the convex body transformation of $\mathcal{X}_c$ having an additional weight-constrained dimension.

Given the intractability of exactly performing the computation in Equation 7.6, CLAUSEWEIGHT$_{\mathrm{D}}$ uses *Monte-Carlo sampling* to approximate the WMI of c. We show in this proof that this sampling is (i) correct, (ii) returns probabilistic guarantees, and (iii) can be performed in polynomial time, since w is ρ-restricted.

We start by proving the correctness of Monte-Carlo sampling for approximating $\mathrm{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$. In Equation 7.6, observe that the final form for $\mathrm{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ can also be written as

$$\prod_{p \in \mathbf{v}_c} w_b(p) \; \mathbb{E}_{\mathbf{v} \models c} \Big[\int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) \; d\mathbf{x} \Big], \tag{7.7}$$

since the summation, weighted by the probability product for literals not appearing in c, corresponds to the expected value of the inner integral given a Boolean assignment randomly sampled according to the distribution w_b. Hence, Equation 7.6 can be approximated with probabilistic guarantees by taking Monte-Carlo samples from the aforementioned distribution and computing the mean of all integral samples. This sampling is done as follows:

1. Sample a random Boolean assignment $\mathbf{v}$ using w_b (CLAUSEWEIGHT$_{\mathrm{D}}$ line 6).
2. Compute the resulting weighted model integral over $\mathcal{X}_c$ given $w_x(\mathbf{x}, \mathbf{v})$ using VOLUME$[\epsilon_{\mathrm{Comp}}, \delta_{\mathrm{Comp}}]$. (CLAUSEWEIGHT$_{\mathrm{D}}$ lines 7-8).

As the mean of this sampling procedure approximates the summation in Equation 7.6, it only remains to multiply this mean by the product of Boolean weights, which

corresponds to lines 9 and 11 of CLAUSEWEIGHT$_{\mathrm{D}}$, to correctly yield an approximation of WMI$(c, w \mid \mathbf{X}, \mathbf{V})$, as required.

We now study probabilistic guarantees for CLAUSEWEIGHT$_{\mathrm{D}}$. The sampling procedure produces s independent and identically distributed (i.i.d) random variables from the w_b-weighted distribution of possible integral values. Let $\bar{X}$ denote the mean of these s random variables, and let $\mu = \mathbb{E}_{v \models c}\Big[\int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v})\ d\mathbf{x}\Big]$ for ease of notation. From Equation 7.4, we infer that $a(\mathcal{X}_c) \leq \mu \leq b(\mathcal{X}_c)$. Therefore, we can use $a(\mathcal{X}_c)$ and $b(\mathcal{X}_c)$ within the Hoeffding bound for sampling s i.i.d variables. This yields, for a target additive difference t,

$$\Pr\left(|\bar{X} - \mu| \geq t\right) \leq 2\exp\left(\frac{-2st^2}{(b(\mathcal{X}_c) - a(\mathcal{X}_c))^2}\right)$$

We now replace t with $\epsilon_{\mathrm{Samp}}\mu$ to obtain a multiplicative error bound, and upper-bound the failure probability by δ_{Samp} to compute a lower bound for sample complexity:

$$s \geq \ln(\frac{2}{\delta_{\mathrm{Samp}}})\frac{1}{2\epsilon_{\mathrm{Samp}}^2}\frac{(b(\mathcal{X}_c) - a(\mathcal{X}_c))^2}{\mu^2} \tag{7.8}$$

$$\geq \ln(\frac{2}{\delta_{\mathrm{Samp}}})\frac{1}{2\epsilon_{\mathrm{Samp}}^2}\frac{(b(\mathcal{X}_c) - a(\mathcal{X}_c))^2}{a(\mathcal{X}_c)^2} \tag{7.9}$$

$$\geq \ln(\frac{2}{\delta_{\mathrm{Samp}}})\frac{1}{2\epsilon_{\mathrm{Samp}}^2}\left(\frac{b(\mathcal{X}_c)}{a(\mathcal{X}_c)}\right)^2 \tag{7.10}$$

$$\geq \ln(\frac{2}{\delta_{\mathrm{Samp}}})\frac{1}{2\epsilon_{\mathrm{Samp}}^2}\rho^2 \tag{7.11}$$

To obtain Equation 7.9 from Equation 7.8, we replace μ by $a(\mathcal{X}_c)$ to yield the most pessimistic lower bound possible for s: Since μ is unknown, but satisfies $a(\mathcal{X}_c) \leq \mu \leq b(\mathcal{X}_c)$, then replacing it by its minimum possible value maximises the fraction, and whatever the correct value of μ, the number of samples corresponding to the worst-case $\mu = a(\mathcal{X}_c)$ is sufficient to produce the required error and confidence guarantees for any value of μ. To obtain Equation 7.10, we eliminate the $a(\mathcal{X}_c)$ term in the numerator, which is possible, since $a(\mathcal{X}_c) \geq 0$, and thus this step yields a tighter sample complexity lower bound. Though this step raises the number of samples, it remains a correct, albeit more pessimistic, bound, but most importantly, it produces a form where an explicit ratio between a and b can be seen. From here, the final bound in Equation 7.11 can be reached by replacing this ratio with ρ as ρ, by definition, is greater or equal to the maximum value of this ratio for any convex polytope[1].

[1]Note that, in practice, one can use ρ' instead of ρ for a higher, but tractably computable sampling bound, as $\rho \leq \rho'$.

Given s Monte-Carlo samples, ClauseWeight$_{\text{D}}$ ensures that sampling introduces a multiplicative error of at most ϵ_{Samp} with probability $1 - \delta_{\text{Samp}}$. This sample complexity s is polynomial in $\frac{1}{\epsilon_{\text{Samp}}}$, $\frac{1}{\delta_{\text{Samp}}}$, n, m, and k as w is ρ-restricted. Furthermore, a sampling step runs in polynomial time, as both Boolean sampling and the call to the Volume FPRAS run in polynomial time. Hence, ClauseWeight$_{\text{D}}$ runs in polynomial time overall, more precisely in $O^*(sn^4\epsilon_{\text{Comp}}^{-2}) = O^*(n^4\epsilon_{\text{Samp}}^{-2}\epsilon_{\text{Comp}}^{-2}) = O^*(n^4\epsilon_V^{-4}) = O^*(n^4\epsilon^{-8})$, where ϵ is the target error for ApproxWMI$_{\text{D}}$.

Finally, we show that, under the conditions of Lemma 7.3.1, ClauseWeight$_{\text{D}}$ produces an (ϵ, δ)-approximation of $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$. In what follows, we assume that ClauseWeight$_{\text{D}}$ fails if Monte-Carlo sampling fails or if *any* of the Volume calls fail. Therefore, we first prove that, when no failure occurs, the error bounds in Lemma 7.3.1 produce an estimate within the ϵ error requirement. Then, we prove that the asserted confidence bounds upper-bound the probability of any ClauseWeight$_{\text{D}}$ failure, denoted f_v, by δ.

Error bounds. We first assume a successful run where both Monte-Carlo sampling and Volume produce values within their respective multiplicative error bounds. Setting ϵ_{Samp} and ϵ_{Comp} within the bounds of Lemma 7.3.1 yields the following bound on $\bar{X}$:

$$\Big(1 - \frac{\epsilon}{1+\sqrt{2}}\Big)^2 \mu \leq \bar{X} \leq \Big(1 + \frac{\epsilon}{1+\sqrt{2}}\Big)^2 \mu.$$

For $\forall 0 < \epsilon \leq 1$, we have that:

$$(1+\epsilon) - \Big(1 + \frac{\epsilon}{1+\sqrt{2}}\Big)^2 = \epsilon\Big(\frac{(1+\sqrt{2})^2 - 2(1+\sqrt{2}) - \epsilon}{(1+\sqrt{2})^2}\Big) = \frac{\epsilon(1-\epsilon)}{(1+\sqrt{2})^2} \geq 0.$$

Analogously,

$$\Big(1 - \frac{\epsilon}{1+\sqrt{2}}\Big)^2 - (1-\epsilon) = \epsilon - \frac{2\epsilon}{1+\sqrt{2}} + \frac{\epsilon^2}{(1+\sqrt{2})^2} = \frac{\epsilon(1+\epsilon)}{(1+\sqrt{2})^2} \geq 0.$$

Therefore, we can replace the lower and upper bounds in the original inequality by $(1+\epsilon)$ and $(1-\epsilon)$ respectively, as follows:

$$\Big(1-\epsilon\Big)\mu \leq \bar{X} \leq \Big(1+\epsilon\Big)\mu,$$

and, following multiplication by $\Big(\prod_{p \in \mathbf{v}_c} w_b(p)\Big)$ on all sides, we obtain:

$$\Big(1-\epsilon\Big)Q \leq \bar{X}\Big(\prod_{p \in \mathbf{v}_c} w_b(p)\Big) \leq \Big(1+\epsilon\Big)Q,$$

where $Q = \text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$, as required.

Confidence bounds. VOLUME is called s times within CLAUSEWEIGHT$_\text{D}$. Hence, we apply the union bound, to upper-bound f_V, and obtain:

$$\Pr(f_V) \leq s\delta_{\text{Comp}} + \delta_{\text{Samp}} = s\frac{\delta}{2s} + \frac{\delta}{2} = \frac{\delta}{2} + \frac{\delta}{2} = \delta.$$

□

We now combine Theorem 7.3.2 and Lemma 7.3.1 to show that APPROXWMI$_\text{D}$ is an FPRAS for WMI (DNF) given a concave and ρ-restricted w under the factorisation of Equation 7.3.

Theorem 7.3.3. APPROXWMI$_\text{D}$ *relative to FPRAS oracles* CLAUSEWEIGHT$_\text{D}$, SAMPLE$_\text{D}$, *and* EVALUATE *having error* ϵ_V, ϵ_S, ϵ_P *and confidence* δ_V, δ_S, δ_P, *respectively, is an FPRAS for* WMI(DNF) *over a* w *that is concave,* ρ*-restricted, and factorised according to Equation 7.3, with error* ϵ *and confidence* δ *and using* $T = \frac{8\ln(\frac{8}{\delta})(1+\tilde{\epsilon})k}{\tilde{\epsilon}^2 - 8(\tilde{C}-1)k}$ *iterations, for*

1. $\epsilon_V, \epsilon_S \leq \frac{\epsilon^2}{47k}$, $\epsilon_P \leq \frac{\epsilon^2}{47k^2}$, *and*
2. $\delta_V \leq \frac{\delta}{4k}$, $\delta_S + \delta_P \leq \frac{\delta}{2276\ln(\frac{8}{\delta})\frac{k}{\epsilon^2}}$.

Finally, the running time of APPROXWMI$_\text{D}$, as a function of CLAUSEWEIGHT$_\text{D}$ runtime r'_C, SAMPLE$_\text{D}$ runtime r'_S, and EVALUATE runtime r_E, is $O(k \cdot r'_C + T(r'_S + r_E))$, analogously to APPROXWMI. Furthermore, for the same choice of CONVEXBODYSAMPLER, it is easy to see that $r_S = r'_S$. However, the main difference in running time between APPROXWMI and APPROXWMI$_\text{D}$ comes from the difference between r'_C and r_C. Indeed, given the same choice of VOLUME oracle, with running time r_V, $r_C = O(m + r_V)$, whereas $r'_C = O(s(m + r_V)) = O(\frac{1}{\epsilon^4}\ln(\frac{1}{\delta})(m + r_V))$. Hence, the wider applicability of APPROXWMI$_\text{D}$ comes at the expense of a larger runtime complexity, owing to the larger power of $\frac{1}{\epsilon}$ required for CLAUSEWEIGHT$_\text{D}$.

7.4 Experimental Evaluation

In this section, we empirically evaluate the performance of APPROXWMI on a set of randomly generated DNF formulas. In particular, we measure the running time APPROXWMI required to solve different DNF instances, and analyse the impact of different DNF parameters, such as clause width, number of variables, and number of clauses, on this running time.

First, we present our DNF generation procedure. Then, we present and justify our experimental setup, namely our choices for weight functions and oracles for sampling and volume computation. Finally, we report our experimental results, and show the runtime efficiency, efficacy and scalability of ApproxWMI.

7.4.1 Generating Evaluation Data

To evaluate ApproxWMI, we randomly generate an evaluation set of DNF formulas. This set is generated with different configurations relative to the number of Boolean and real variables, the width of clauses, and the number of these clauses, such that all possible configurations are evenly represented during evaluation. More specifically, we set our configuration parameters as follows:

- **Number of Boolean and real variables (m, n):** We generate DNF formulas such that they have an equal number of real and Boolean variables, i.e., $m = n$, $|\mathbf{X}| = |\mathbf{V}|$. We opt for setting $m = n$ to produce DNF instances that fairly represent both real and Boolean domains in the hybrid domain, and avoid edge case instances that too closely resemble either volume computation (for $m \approx 0$) or standard weighted model counting (for $n \approx 0$).

- **Clause width (W):** For simplicity, we opt for a uniform clause width W in our generation, such that all the clauses of a DNF formula have width W.

- **Number of clauses (k):** The number of clauses is set as $k = \lfloor \frac{m+n+20}{W} \rfloor$, as this choice for k ensures a balanced computational requirement for different similarly-sized DNF instances. More precisely, this value of k ensures that the total number of atoms present across all DNF clauses (including redundancies) in a given formula remains highly similar to that of distinct formulas with identical values of m and n, but different widths W.

For every configuration (m, n, k, W), a DNF formula is randomly generated. Initially, the configuration imposes a high-level structure of the DNF, namely that it has a fixed number of clauses k, and each of these clauses has W atoms. What remains to obtain a fully specified DNF is therefore to *assign* values to the atoms in every clause, either to Boolean literals or LRA constraints defined over $\mathbf{X}$. We perform this allocation of atoms analogously to the common "balls in boxes" combinatorial problem.

More specifically, we define the kW atom locations in the DNF as "balls", and define "boxes" corresponding to all Boolean variables and to a number of LRA constraints. Then, we randomly allocate the balls to these boxes, such that no box remains empty. Thus, this allocation ensures that all Boolean variables, and a predetermined number of LRA clauses, appear in at least one clause of the DNF. Concretely, we define m Boolean boxes and n LRA boxes, which are placeholders that each *independently* yield LRA constraints in the final DNF. This ensures that an almost even split between LRA and Boolean atoms arises in our generated formulas. Furthermore, since n LRA boxes are defined over $\mathbf{X}$, and $|\mathbf{X}| = n$, and since LRA atoms involve at least one real variable, then every real variable is expected to appear in at least one LRA atom. We illustrate this construction with a running example in Example 7.4.1.

Example 7.4.1. *Consider a configuration (m, n, k, W), where $m = n = 3$, $k = 3$, and $W = 3$. Then, we obtain $kW = 9$ "balls" corresponding to all 9 atom positions in the* DNF*, and 6 "boxes", namely 3 Boolean boxes corresponding to Boolean variables p_0, p_1, p_2 respectively, and 3 LRA placeholder boxes l_0, l_1, l_2. These placeholders can be represented with a single symbol l^*, as each of the appearances of l^* will later be independently instantiated with LRA constraints over $\mathbf{X}$.*

Remark. Though we use a single placeholder l^* to generate LRA constraints, we nonetheless represent this placeholder using n boxes (rather than 1) in our generation procedure, as this produces more uniform allocations distributions for l^* with respect to Boolean variables. More concretely, by using n boxes, we have equality of Boolean literal and LRA constraint appearances in expectation.

To allocate atom balls to boxes, we use analogous allocation mechanisms to the data generation procedure in Chapter 6, namely:

- **Uniform allocation:** All balls are allocated uniformly from the set of all $\binom{kW-1}{m+n-1}$ configurations[2] yielding non-empty boxes as in the standard balls in boxes problem. Moreover, an additional check is imposed to prevent a box from receiving more balls than the number of clauses k[3].

- **Privileged allocation:** A small subset of "privileged" boxes is uniformly randomly selected, and randomly allocated another random "surplus" subset of balls, while keeping at least as many balls as overall remaining boxes.

[2]In the balls and boxes problem, putting a balls in b boxes such that no box is empty can be done in $\binom{a-1}{b-1}$ ways. Our number of configurations is therefore a direct application of this formula.

[3]Note that the intra-clause redundancy requirement when allocating positions does not apply to l^*, as this placeholder will yield independent constraints at the end of generation.

Location selection for all variable appearances also follows the same procedure as Chapter 6: Following box allocation, a heuristic is used to identify specific DNF atom locations for every variable and LRA constraint, such that no variable or constraint appears twice in the same clause, and this heuristic sets the allocations of boxes in decreasing order of their current ball allocations, while keeping the number of available positions in all clauses as evenly spread as possible. Finally, once all clause positions are filled, signs for Boolean literals are also analogously set. That is:

1. All positions stemming from uniform allocation have their signs independently sampled are uniformly at random.

2. All privileged Boolean atoms allocated are *jointly* set a literal sign. That is, *all* atom allocations share one same literal sign, which is negated with probability 0.5.

Thus, this generation procedure only differs from the standard Boolean procedure in Chapter 6 in that an LRA placeholder l^* is used as a "virtual" Boolean variable, which is eventually replaced with LRA constraints.

This allocation scheme and heuristic are analogous to the procedure described in Chapter 6, which generates formulas for neural weighted DNF counting. An allocation of balls to boxes, mapped to DNF clause positions, is shown for the running example in Example 7.4.2.

Example 7.4.2. *Consider the earlier configuration where* $m = n = 3, k = 3$*, and* $W = 3$*. Then, an example atom allocation, generated uniformly, is:*

$$(p_0 \wedge p_2 \wedge \neg p_1) \vee (l^* \wedge p_2 \wedge l^*) \vee (p_1 \wedge l^* \wedge l^*),$$

where, as before, l^* *is a placeholder such that, at every location it appears, an LRA constraint is subsequently generated* independently*. An example privileged allocation, with privileged Boolean variable* p_0*, is:*

$$(p_0 \wedge p_2 \wedge \neg p_1) \vee (l^* \wedge p_0 \wedge l^*) \vee (p_1 \wedge l^* \wedge p_0).$$

Observe that p_0 *is a positive literal in all its appearances, by construction.*

It now remains to replace the LRA placeholder l^* with concrete constraints over $\mathbf{X}$. This is done independently for every appearance of l^*, such that constraints yield a non-empty convex body at the *clause* level. For a given DNF clause c, this is done as follows:

1. First, a random point z in $\mathbb{R}^n$ is uniformly sampled, such that all generated constraints in c must be satisfied by z. This ensures that the convex polytope defined by c is non-empty.

2. For every placeholder: (i) randomly select a subset S of $Geom(0.5)$ real variables, where $Geom$ denotes the geometric distribution, (ii) sample $w_S \in \mathbb{R}^{|S|}$ weights for these variables from Gaussian distributions and (iii) sample a random Gaussian value v and set the linear constraint $w_S.S \leq v$.

3. If z satisfies $w_S.S \leq v$, then $w.S \leq v$ is added to c, irrespective of the original placeholder sign. Otherwise, $w_S.S \geq v$, which z must then satisfy, is added.

Example 7.4.3. *We again consider the earlier example:*

$$(p_0 \wedge p_2 \wedge \neg p_1) \vee (l^* \wedge p_2 \wedge l^*) \vee (p_1 \wedge l^* \wedge l^*).$$

Then, an example LRA constraint generation yields points $(0,0,0)$ *and* $(1,1,1)$ *for the second and third clauses respectively, and the following constraints are randomly generated:*

$$(p_0 \wedge p_2 \wedge \neg p_1) \vee (x_2 - x_0 \geq -1 \ \wedge \ p_2 \wedge x_1 + x_2 \leq 1) \vee (p_1 \wedge x_0 + 3x_2 \leq 6 \ \wedge \ x_0 - 3.5x_1 \leq 1),$$

where $x_2 - x_0 \geq -1$ *is negated to be satisfied by* $(0,0,0)$.

7.4.2 Experimental Setup

In our experiments, we bound all real variables in $\mathbf{X}$ such that $\forall x \in \mathbf{X}, 0 \leq x \leq 10$. This ensures a bounded domain of integration, and that integrals for a finite-valued weight function w are finite-valued. In terms of w, we opt for *polynomial* functions w_x, given their simplicity and popularity for evaluation in the WMI literature, and given the ease of generating concave polynomial functions, in keeping with our theoretical requirement.

Concretely, we generate a polynomial function w_x consisting of a sum of up to 4 terms. Within this polynomial, every non-constant term consists of a coefficient, which is a uniformly randomly sampled integer in $\{1, ..., 10\}$, as well as a degree randomly chosen from the distribution $Geom(0.6)$ and upper-bounded by 5. This degree is then uniformly randomly allocated to one or more distinct real variables. For instance, a degree of 5 can be allocated as $x_1^3 x_2^2$. Next, the coefficients of terms with degree of 2 or higher are made negative to maintain concavity, and finally a

constant is added to the polynomial to ensure the positivity of the weight function over the defined real domain.

For the volume oracle VOLUME within CLAUSEWEIGHT, we use LattE [11], an exact volume computation tool. Though this theoretically is not scalable given the complexity of the algorithm, it is viable in practice for multiple reasons. First of all, LattE, despite being exact, natively supports polynomial weight functions, and performs reliably in practice for smaller-scale formulas compared with approximate techniques [54]. Second, we use LattE in a more optimised fashion, which eliminates all variables in $\mathbf{X}$ not appearing in the LRA constraints of a given clause, or in w_x, which minimises computation time. More specifically, we optimise our use of LattE by separately (trivially) integrating over variables not appearing in a clause or a term of w, and only running LattE over the appearing variables.

This optimisation is effective in practice for two reasons. First, the number of real variables appearing in a clause is small in expectation, at around $2W$ in the worst case of an all-LRA clause, and W for an even distribution of Boolean and LRA atoms, and thus the dimensionality of the polytope space is small for computation purposes. Second, the modularity of the DNF representation breaks down an n-dimensional problem into k smaller sub-problems, which can be solved more efficiently. Observe that this is not the case with CNF representations, as the outer conjunctions prevent an efficient computation of smaller sub-problems. By contrast, the clauses in a DNF formula are not entangled, as each clause defines an individual lower-dimensional polytope, and these are combined with the outer disjunction. Hence, DNF representations allow for significantly more efficient division of the input problem relative to CNF.

For the oracle SAMPLE used within CONVEXBODYSAMPLER, we use hit-and-run [38], with a constant factor 10^3 used to compute the number of walk iterations, as is standard in practice. It is *unknown* whether this constant preserves theoretical guarantees, but it is widely used given that the best-known theoretical constant factors for hit-and-run are loose and prohibitively large (i.e., 10^{30}) [108], and because this value allows for a reasonable trade-off between accuracy and efficiency. We investigate the accuracy of this setup in detail with a dedicated experiment in Section 7.4.4, where we compare the outputs of our APPROXWMI setup against those produced by an exact solver on small DNF instances. Finally, we run all experiments with a timeout of 5000 seconds, on a server with a Haswell 5-2640v3, 2.60GHz CPU and 12 GB of RAM.

7.4.3 Runtime Experiments

To evaluate the runtime of ApproxUnion, we produce a dataset consisting of 160 DNF formulas, resulting from 40 distinct configurations, each represented by 4 DNF formulas. More specifically, we jointly set the value for m and n to values from 50 to 500 inclusive, in increments of 50, and thus the total number of variables is between 100 and 1,000, and set width W to values from the set $\{3, 5, 8, 13\}$, to ensure a comprehensive coverage of all width ranges. Then, we run ApproxWMI on each formula using 4 distinct (ϵ, δ) settings: (0.15, 0.05), (0.20, 0.10), (0.25, 0.15), and (0.35, 0.25), and compute the mean runtime across 5 runs per setting. These error and confidence values are chosen in keeping with realistic practical targets, with 0.35 representing a reasonable loose error requirement, and 0.15 a common tighter requirement, particularly for larger instances.

All ApproxWMI running times with respect to the total number of variables $m+n$, clause width W, and the target error ϵ and confidence δ are presented in Figure 7.1. We see that ApproxWMI solves DNF instances with up to $1,000$ variables within 5000 seconds over all W for $\epsilon = 0.35$ and $\delta = 0.25$. In fact, we also observe that, for instances with $1,000$ variables having $W = 5, 8, 13$, all run within 2000 seconds, for $(\epsilon, \delta) = (0.25, 0.15)$, and that the increase in running time relative to the $(0.35, 0.25)$ setting for all widths is minor, at around 40%, which is far smaller than the theoretically expected increase of $\frac{0.35}{0.25}^6 \approx 7.5$. This is due to the efficiency of the practical setup of Sample relative to tighter ϵ_S and δ_S, which alleviates the impact of theoretical higher negative powers of ϵ (implicitly ϵ_S) derived in Section 7.3.3. More concretely, increasing the sampling iterations (due to tighter bounds) within the current hit-and-run implementation incurs a minimal additional overhead given the commonly applied constant factor and scale of our experimental instances, as samples can easily be parallelised.

For tighter ϵ and δ, the system maintains high performance, despite the increase in T and the theoretical complexity of ApproxWMI, which involves high negative powers of ϵ. Indeed, with $\epsilon = 0.15$ and $\delta = 0.05$, all instances with widths 8 and 13 finish within ∼2000 seconds, whereas large instances of width 3 and $m + n \geq 500$, and width 5, $m + n \geq 800$, time out. Furthermore, all experiments at widths 5, 8, and 13 terminate within the time limit for $\epsilon = 0.2$, $\delta = 0.1$, with only width 3 experiments timing out for $m + n \geq 700$. These results highlight the scalability of ApproxWMI, deployed with standard sampling and volume computation tools, for computing approximations to WMI, even relative to tighter error and confidence targets, and on large instances involving $1,000$ variables.

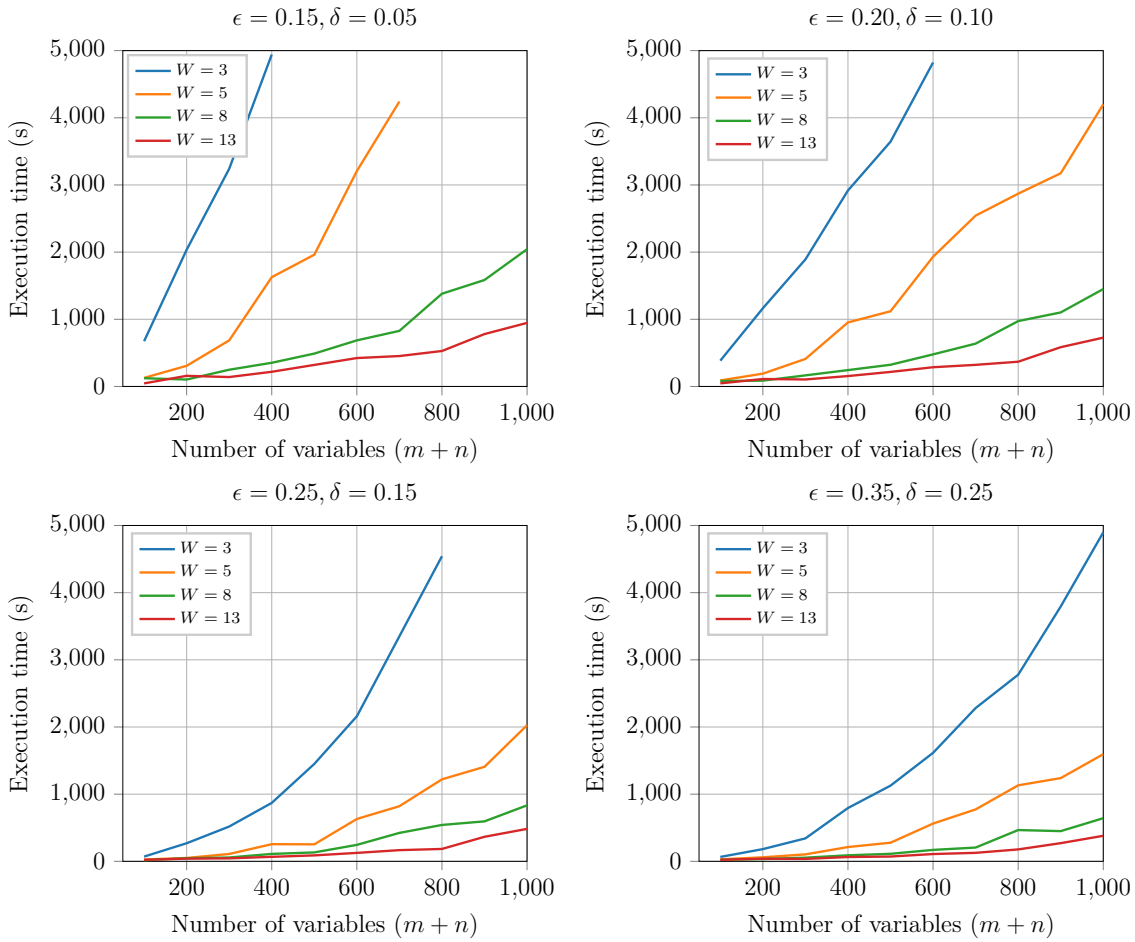


Figure 7.1: Runtime results for ApproxWMI relative to different ϵ, δ. For $\epsilon = 0.35, \delta = 0.25$, all instances, including those with 1,000 variables, terminate within 5000 seconds, and higher-width instances ($W = 8, 13$) all terminate within $\sim$ 2000 seconds for all ϵ, δ.

Somewhat unintuitively, our experiments show that system performance worsens as W *decreases*, irrespective of error and confidence targets. In fact, we can see that, for DNF instances with $W = 3$, ApproxWMI requires almost triple the time compared to instances with higher W across all (ϵ, δ) configurations. Though this behaviour is surprising, it can be attributed to an increased number of sampling replacements and calls within ApproxWMI. Indeed, for smaller widths, it is likelier that a call to Evaluate will yield "True" since there are less constraints and Boolean literals to satisfy. As a result, further calls to Sample, which runs in time $O^*(m+n^3)$, will be required. Though the implementation for Sample is efficient relative to ϵ_S, as mentioned earlier, it still remains an expensive component in the ApproxWMI sampling iteration, and has an overhead when called. For comparison, a membership

check runs in $O(m+Wn)$, which is very small in our setting as W is small. Thus, the increased number of calls to sample due to likelier replacements at small W configurations, combined with the typically large number of sampling iterations T, impose a significant computational overhead on ApproxWMI. This behaviour also justifies the improved performance with increased width observed in Figure 7.1, as higher-width clauses are less likely to cause replacements, which decreases the expected number of calls to Sample and reduces running time.

Overall, our results confirm our intuitions about ApproxWMI: ApproxWMI indeed scales to large instances having up to 1,000 total variables, and can provide approximations in a reasonable time using common oracle choices, even for tight ϵ and δ. Our experiments also highlight that calls to ClauseWeight, though invoking an exact volume computation tool, are not a bottleneck to system performance. In fact, for any instance in our evaluation, ClauseWeight calls require at most 450 seconds *in total* to run. This also aligns with our experimental setup choices and intuitions, as the bounded width W used in our experiments and the modularity of the DNF structure allow for lower-dimensional volume computations that LattE can perform very efficiently. Finally, our results show that the main performance bottleneck for ApproxWMI is the number of calls to Sample. This is particularly evident from the high dependence of running time on width W, and shows that, even with a reduced constant factor, convex body sampling is a highly costly operation when called repeatedly.

We conclude our discussion with an added theoretical note that further supports our choice of an exact Volume oracle: In addition to competitive practical performance at smaller scale, exact Volume oracles set $\epsilon_V, \epsilon_P, \delta_V, \delta_P = 0$, which enables the derivation of looser error and confidence bounds for Sample. More concretely, replacing these values in the derivation of Lemma 3 of ApproxUnion (cf. Corollary 7.3.1) [25] yields the improved bound $\epsilon_S < \frac{\epsilon^2}{8k}$, plugging in $\delta_p = 0$ into the proof of Corollary 7.3.1 improves the bound by a factor of 1.5, yielding $\delta_S \leq \frac{\delta}{1518 \ln(\frac{8}{\delta})\frac{k}{\epsilon^2}}$. Though these new bounds are only constant factor improvements over the case with approximate Volume, these factors are rather large, at 5.875 and 1.5 respectively, and amplify when exponentiated, particularly for ϵ. Thus, they allow for significant practical gains, which were very beneficial in our experiments, all while preserving the theoretical guarantees of standard ApproxWMI. Moreover, the use of an exact volume computation tool also preserves the polynomial running time of ApproxWMI given bounded width and a bounded number of variables in all LRA constraints. Though these assumptions are somewhat restrictive, they are typically encountered

in practice: Clause widths typically do not exceed 100, and LRA constraints do not include large sets of variables. Hence, the use of an exact VOLUME oracle is justified theoretically, both in terms of reducing the overhead of sampling, and for its effectiveness in practice given the boundedness of width, LRA constraints, and the modularity of DNF structures.

7.4.4 Accuracy Experiments

APPROXWMI is an FPRAS for WMI(DNF), and provably provides approximation guarantees. However, in practice, hidden constant factors in the subroutines of APPROXWMI, such as the prohibitively large constant factor in hit-and-run (10^{30}), make a theoretically faithful implementation intractable. Given this limitation, and in line with standard practice, we therefore used a lower constant factor of 10^3 for hit-and-run in our setup. However, this constant is not known to preserve the theoretical guarantees of hit-and-run. Therefore, a natural question is whether this scaling down of the hit-and-run constant nonetheless achieves performance that still matches the FPRAS guarantees, and thus whether this setup is reliable in practice.

To address this question, we generate an additional dataset consisting of smaller formulas, which can tractably be solved exactly. More specifically, this dataset consists of 20 formulas, each having $m = n = 20$, and with widths evenly distributed among the set $\{2, 3, 5, 8\}$. Furthermore, this dataset only uses weight functions where the real component w_x is restricted to be linear, and to involve at most three variables, so as to best match the performance profile of exact solvers.

To solve dataset instances exactly, we use a compilation tool based on extended algebraic decision diagrams (XADD) [93], offered as part of the PyWMI library [94]. Then, we set up APPROXWMI with $\epsilon = 0.15$ and $\delta = 0.05$, and run it 1000 times with 3 different constant factors for hit-and-run: 10, 100, and 1000 (as in the earlier experiment). Finally, we check whether, for every hit-and-run constant factor, the WMI returned by APPROXWMI is within ϵ relative to the exact solution with a frequency exceeding $1 - \delta = 0.95$.

The frequency results for this experiment are shown in Table 7.1, with values exceeding 0.95 shown in bold. First, we observe that, at the default constant value of 1000, APPROXWMI significantly oversatisifes the theoretical confidence guarantees, and near-perfectly makes predictions within the approximation error bounds. Surprisingly, APPROXWMI continues to oversatisfy the confidence bound even with a hit-and-run constant of 100. In fact, APPROXWMI only underperforms when the constant is set to 10, a very low and unrealistic setting. This suggests that hit-and-run

Table 7.1: Frequency of ApproxWMI runs ($\epsilon = 0.15, \delta = 0.05$) satisfying error bounds relative to different hit-and-run constant factors. Even with a constant value of 100, ApproxWMI comfortably satisfies the approximation confidence bound (0.95).

Hit-and-run constant	10	100	1000
Within ϵ frequency	0.635	**0.986**	**0.998**

can indeed make good predictions with a reasonable choice of constant, and that the current setup of ApproxWMI, with hit-and-run constant factor 1000, comfortably achieves high-quality performance that aligns well with theoretical guarantees.

All in all, these are encouraging findings, which further motivate the practical use of ApproxWMI, and corroborate the empirical evidence for the looseness of the theoretical hit-and-run constant. In particular, these results, combined with the earlier runtime performance of ApproxWMI, highlight that this ApproxWMI setup performs reliably, efficiently, and can scale to large instances involving up to 1,000 variables.

7.5 Related Work

WMC is a unifying framework for probabilistic inference across prominent probabilistic models: Inference in probabilistic graphical models [95] reduces to WMC(CNF) [146, 37], and similar reductions of inference tasks to WMC exist for Markov Logic Networks (MLNs) [141], probabilistic logic programming [59], and more generally, for relational models [65]. However, WMC cannot capture hybrid probabilistic models involving both continuous and discrete variables, which occur in various contexts, including hybrid MLNs [174], and hybrid Bayesian networks [64, 147]. WMI is proposed as a unifying inference framework for these hybrid models [17], and has been addressed in recent years both with exact and approximate solving techniques [126]

In terms of computational complexity, both WMI and WMC are #P-hard [168], so are highly intractable for exact solving, as the class $\mathrm{P}^{\#\mathrm{P}}$ contains the entire polynomial hierarchy [163]. Unfortunately, this intractability even extends to restricted settings of both these problems. For example, DNF counting remains #P-hard, even on positive, partitioned DNF formulas with clause width at most 2 [138]. Furthermore, WMI can be tractably solved if and only if formulas have a primal graph that is a tree with diameter logarithmic in the number of nodes [190]. In a primal graph, formula variables are represented as nodes, and edges between these nodes appear when

the variables co-appear in a same clause. Thus, this result shows that only branching tree-shaped dependencies between variables can be tractably solved exactly for WMI, and that even simple linear chain dependencies make formulas hard to solve.

Despite the intractability of exact solving, many general-purpose exact solvers, based on several optimisations, have been developed. These solvers exploit formula structure and perform substantial pre-processing to reduce the computation load of WMI. For instance, a tool based on SMT *predicate abstraction* techniques [125] has been proposed to reduce the number of models. Predicate abstraction introduces a set of predicates ψ and then abstracts away solutions of the original formula φ by iterating over truth assignments to ψ and considering solutions that are shared with φ. Hence, ψ, typically designed to be relevant to φ, that is, having predicates whose truth or falsehood affects φ, allows to significantly reduce the search space for the SMT solver. Indeed, predicate abstraction reduces the original exponential search over the atoms of φ is to a more manageable (but still exponential) search over ψ, where $|\psi| << \mathsf{atoms}(\mathbf{X}, \mathbf{V})$. This approach also supports a rich class of weight functions, allowing conditional dependencies and case splits.

In addition to predicate abstraction, Symbo [117] uses knowledge compilation to more efficiently perform WMI. More specifically, Symbo compiles φ into a d-DNNF representation, which can subsequently be run in polynomial time. Though the compilation is itself computationally expensive, it leads to substantial computational speedup when integrals over φ need to be computed multiple times. Symbo is inspired by knowledge compilation approaches for WMC, and performs compilation by casting WMI as an algebraic model counting (AMC) task. With the AMC formulation, Symbo then defines the corresponding literal labels and symbolic weights, which a symbolic integration tool then uses, along with the compiled d-DNNF φ, to compute WMI. Symbo uses the common factorisation assumption of Equation 7.2, but additionally supports continuous variables using the theory of non-linear real arithmetic (NRA), which supports sums of *rational powers* of real variables, as opposed to simple linear combinations of variables. Furthermore, Kolb et al. [93] use compilation based on extended algebraic decision diagrams (XADDs) to tackle WMI with LRA atoms, and where w is a piece-wise polynomial function. XADD enables parametrised (i.e., partial) solving of WMI, and is combined with symbolic dynamic programming (SDP) to integrate over the Boolean-real domain and perform partial computation caching, enabling more efficient exact WMI solving.

Finally, exact lower and upper solution bounds, as opposed to the exact solution, are computed for WMI, based on hyper-rectangular decomposition and orthogonal

transformations [121]. Intuitively, hyper-rectangular decomposition incompletely covers the integration domain using sets of hyper-rectangles, such that the total integral over these hyper-rectangles yields a lower bound for WMI. Analogously, upper bounds are computed by covering the negation of the domain of integration and subtracting the integral from the total domain integral. However, the domain of integration can be hard to cover efficiently using hyper-rectangles alone, if, for instance, this domain is not axis-aligned. Therefore, the approach proposes orthogonal transformations, which transform the input formula using rotations and Pythagorean triples to another representation that is more efficiently decomposable. This transformation is shown to preserve weights and the correctness of the problem setting for variables following a Gaussian density function. This method is therefore more efficient than standard WMI solving, as integration over hyper-rectangles is simple, yields exact bounds, and can be made arbitrarily accurate by increasing the number of used hyper-rectangles.

Parallel to exact solvers, *approximate* methods have been developed for WMI to circumvent its intractability in the exact setting. A common such approximation method is *hashing*. At a high level, hashing methods partition the solution space into disjoint, smaller, partitions where counting (resp., integration) is tractable, solve the problem over these easier partitions, then leverage their results to compute an approximation, with guarantees, for the overall model count (resp., integral). Hashing-based methods are popular for WMC [34], and were lifted to WMI [16] following its formulation [17]. More concretely, WMI is approached using *propositional abstraction*, such that WMI statements are directly cast into WMC. With this representation, it is then shown that universal hashing techniques used for WMC also apply to WMI, and therefore, that an NP-oracle can also effectively partition the solution space for hybrid domains. Hence, this abstraction-based hashing tool for WMI can compute probabilistic approximations, with guarantees, using a polynomial number of calls to an NP oracle.

In addition to hashing, knowledge compilation is also used to support WMI approximation. In particular, Sampo [117] extends Symbo to overcome the limitations of symbolic integration. More concretely, it uses Monte Carlo sampling to compute approximate integrals, and thus can apply to general WMI theories such as real arithmetic (RA), where certain integrals cannot be symbolically computed. Sampo performs a number of sampling iterations, in which random variables are sampled and fed into the compiled representation of φ, analogously to Symbo, to yield a weight. Then, the average of these weights yields an approximate density. Sampo therefore approximates WMI with probabilistic guarantees.

Beyond Monte Carlo sampling, Markov Chain Monte Carlo (MCMC) methods have been applied to WMI, but such approaches do not provide any guarantees. The only exception for this is a tool for #SMT[40], the unweighted case of WMI. This tool approximates the solution for #SMT by calling a SAT solver following a randomised algorithm, analogously to MCMC approaches for WMC (CNF), and produces probabilistic guarantees with polynomially many SAT calls [81].

Finally, WMI has recently been approximated with the help of *relaxation* schemes. More specifically, ReCoIn [190], which is short for "Relax, compensate and then integrate", *relaxes* a given WMI instance by removing some of its constraints to yield a "simpler" formula which can be tractably solved exactly. Then, it *compensates* for this relaxation by introducing new auxiliary constraints to the relaxed formula. These new constraints are added to maintain semantic similarity of the relaxed formula relative to the original formula, but are also formulated to keep the relaxed formula tractable. Following relaxation and compensation, the resulting formula is then solved exactly (the *integration* operation), and its solution returned as the approximate WMI. However, the modification of input instances can lead to arbitrarily dissimilar instances, and thus ReCoIn does not provide guarantees with its approximations.

To the best of our knowledge, there is no dedicated study for WMI(DNF), despite WMC (DNF) being extensively studied, partly motivated by the rich literature and research on probabilistic data management [158]. Indeed, query answering in probabilistic databases reduces to WMC (DNF) as every conjunctive query is equivalent to a DNF via its lineage representation and this correspondence extends to different probabilistic data models [30] and to special classes of ontology-mediated probabilistic query answering [23, 24]. This has led to a number of tools and algorithms from KLM [84] to hashing-based techniques [119], and recently to neural model counting approaches (cf. Chapter 6). However, these tools, by construction, cannot handle extended data models which also include continuous distributions. Such data models include Monte Carlo Databases (MCDBs) [80], and the system PIP [87], which extends MCDBs to efficiently query probabilistic data defined over hybrid distributions. Abstracting away subtle technical differences, MCDBs and PIP both introduce approximate inference algorithms. However, unlike our approach, these approaches do not provide guarantees. Therefore, our theoretical study of WMI(DNF), and our proposed FPRAS algorithms for this setting, additionally serve as a unifying framework and perspective for a large class of data models, spanning both standard probabilistic databases and hybrid data models.

7.6 Summary and Outlook

In this work, we studied weighted model integration on DNF structures. First, we presented ApproxWMI, an FPRAS for WMI(DNF) given a concave and factorised weight function w, which is based on FPRAS oracles for volume computation and point sampling over convex bodies. Then, we relaxed the factorisation assumption over w, and extended ApproxWMI to a new FPRAS ApproxWMI$_{\mathrm{D}}$, which uses Monte-Carlo sampling to capture dependencies between the Boolean and real subdomains. Our FPRAS results for WMI(DNF) complement the result of WMC (DNF), and help draw a more complete picture of approximability for these problems. Furthermore, our experimental analysis further shows the potential of ApproxWMI as a scalable and reliable WMI solver over DNF formulas.

Beyond this work, a key goal for research on WMI is to develop alternative approximate approaches with probabilistic guarantees, such as hashing-based approaches, to reduce the need for expensive sampling operations and improve the scalability of approximate WMI(DNF). Furthermore, it is important to investigate further generalisations of this work, allowing it to apply to new families of weight functions, as well as other SMT theories. Overall, we hope this work leads to a better understanding of the complexity of WMI, and stimulates further investigation and development of WMI approaches, leading to more robust WMI systems.

Chapter 8

Conclusions

Relational data provides a structured knowledge representation that arises in a wide range of application domains, ranging from knowledge graphs and citation networks to molecular graphs. This structure is very valuable for artificial intelligence, as it enables a more principled study of relational inductive biases, as well as a more comprehensive modelling and analysis of connectivity structures and dependencies in data. Therefore, a key objective for AI research is to develop models that are efficient, sufficiently expressive, reliable, and which can effectively incorporate relational inductive biases so as to unlock the potential of relational data. Some recent models, such as graph neural networks (GNNs) and shallow node embedding models, look promising, and already offer more principled means to process relational data. However, several challenges remain in this area in order to improve the representation power of inductive capacity of existing models, and further our understanding of the limits and strengths of models in this domain.

In this thesis, we addressed several problems pertaining to learning and inference over relational data. We summarise our contributions below:

1. In Chapter 3, we proposed BoxE, a shallow embedding model based on box embeddings for knowledge graph and knowledge base completion. We also showed that BoxE is fully expressive and can provably capture and inject relational inference patterns from a rich rule language. Empirically, we demonstrated the strength of BoxE on a variety of benchmarks across multiple tasks, including knowledge graph completion and higher-arity knowledge completion, and studied its performance and learned representations in depth. Finally, we validated the ability of BoxE to enforce background ontologies on a subset of the NELL knowledge graph, and evaluated the model in a setting with unary facts.

2. In Chapter 4, we jointly considered node classification and link prediction using a novel unifying perspective, and studied the performance of standard graph neural networks and shallow embedding models (with feature processing) on both tasks in a unified fashion using our carefully produced dataset, WikiAlumni. We showed that the closed-world assumption, ubiquitous in standard MPNNs, is detrimental to their performance on incomplete data, and that joint modelling node classification along with link prediction, offers a more robust means to enrich predictions in this setting and alleviate information loss.

3. In Chapter 5, we studied the expressive power of message passing neural networks (MPNNs) when their input node representations are supplemented with random features, and showed that this simple modification surprisingly makes MPNNs universal approximators of invariant functions over the bounded graph domain. Building on this result, we then designed synthetic datasets EXP and CEXP, and validated the strength of random node initialisation (RNI) empirically. In the process, we also compared MPNNs with RNI against several baselines, including higher-order GNNs.

4. In Chapter 6, we applied MPNNs to a special case of a challenging reasoning problem, weighted model counting, a key problem in probabilistic inference. More specifically, we represented the weighted model counting problem instances as graphs, and instantiated a custom MPNN to perform message passing in a problem-specific and structure-aware fashion. We trained this MPNN using an existing approximate solver, the KLM algorithm [84], and showed that it very effectively and efficiently approximates the input problem, even on unseen weighted model counting instances of a larger size than its training set.

5. In Chapter 7, we theoretically studied the weighted model integration (WMI) problem, a generalisation of weighted model counting, and proposed a fully polynomial randomised approximation scheme (FPRAS) to efficiently compute WMI estimates on larger-scale problem instances. This FPRAS, called ApproxWMI, vastly improves the viability of complex reasoning on large-scale probabilistic databases, and potentially allows for more efficient supervision of neural models in this challenging setting.

Building on this work, an important direction for future research is to develop alternatives and extensions to the standard message passing paradigm so as to overcome its limitations at higher depths, namely oversmoothing and oversquashing. To

this end, an interesting objective is to design models based on other graph kernels. Proposals along these lines include k-hop graph neural networks [132] and random walk graph neural networks [133], which are inspired by the shortest path [22] and random walk kernels [172] respectively. In fact, we have explored this question in our recent work proposing Shortest Path Networks (SPNs) [7]. By moving away from standard direct neighbourhood message passing, one can potentially overcome the oversmoothing and oversquashing bottlenecks and pass information more effectively between distant nodes. Furthemore, alternative message passing protocols could help encode more structure into representation learning. However, a main challenge in this direction is to improve on MPNNs while maintaining their tractability, particularly in light of the high computational complexity of alternate kernels.

Another interesting direction for future work is to propose more refined relational representations to improve the inductive capacity of MPNNs and align it with that of shallow node embedding models. More concretely, embedding models like BoxE can represent rich rule languages, but are limited to transductive applications, whereas MPNNs are inductive, but cannot currently represent or infer rich relational structures. Our work in Chapter 4 shows that the relational representations in MLP-X models, i.e., shallow embedding models, yield significant improvements, and avoid making assumptions on the completeness of edges in the input graph. Hence, it is important to further supplement messages in MPNNs with relational structure, potentially analogous to embedding models, so as to learn relational representations that capture inference patterns and thus perform more refined and rich message passing.

All in all, this thesis makes several contributions towards learning and inference over relational data, and establishes connections between the different models used across tasks in graph representation learning. However, many challenges remain to fully exploit the power of relational data. For instance, a key requirement is to make the studied neural models more reliable, and endow them with probabilistic guarantees. BoxE makes steps in this direction with its inductive capacity and rule injection ability, but similar advances remain needed generally, both within graph neural networks and node embedding models. Furthermore, more work must be done to effectively combine expressive power and generalisation within graph neural networks, all while overcoming the limitations of direct neighbourhood message passing. We hope that this work paves the way for further improvements in this growing research area, and ultimately leads to more powerful, reliable and interpretable techniques for processing relational data.

Bibliography

[1] Ralph Abboud and İsmail İlkan Ceylan. Node classification meets link prediction on knowledge graphs. *CoRR*, abs/2106.07297, 2021.

[2] Ralph Abboud, İsmail İlkan Ceylan, and Radoslav Dimitrov. On the approximability of weighted model integration on dnf structures. In *Proceedings of Seventeeth International Conference on Principles of Knowledge Representation and Reasoning, KR*, pages 828–837, 2020.

[3] Ralph Abboud, İsmail İlkan Ceylan, and Radoslav Dimitrov. Approximate weighted model integration on DNF structures. *Artificial Intelligence Journal, AIJ*, 2022.

[4] Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI*, pages 2112–2118, 2021.

[5] Ralph Abboud, İsmail İlkan Ceylan, and Thomas Lukasiewicz. Learning to Reason: Leveraging neural networks for approximate DNF counting. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI*, pages 3097–3104, 2020.

[6] Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. BoxE: A box embedding model for knowledge base completion. In *Proceedings of the Thirty-Third Annual Conference on Advances in Neural Information Processing Systems, NeurIPS*, pages 9649–9661, 2020.

[7] Ralph Abboud, Radoslav Dimitrov, and İsmail İlkan Ceylan. Shortest path networks for graph property prediction. *CoRR*, abs/2206.01003, 2022.

[8] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *Proceedings of the Ninth International Conference on Learning Representations, ICLR*, 2021.

[9] David L. Applegate and Ravi Kannan. Sampling and integration of near log-concave functions. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, STOC*, pages 156–163, 1991.

[10] Ivana Balazevic, Carl Allen, and Timothy M. Hospedales. TuckER: Tensor factorization for knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the Ninth International Joint Conference on Natural Language Processing, EMNLP-IJCNLP*, pages 5184–5193, 2019.

[11] Velleda Baldoni, Nicole Berline, Jesús A De Loera, Brandon Dutra, Matthias Köppe, Stanislav Moreinis, Gregory Pinto, Michele Vergne, and Jianqiu Wu. A user's guide for LattE integrale v1.7.2. *Optimization*, 22(2), 2014.

[12] Imre Bárány and Zoltán Füredi. Computing the volume is difficult. *Discrete and Computational Geometry*, 2:319–326, 1987.

[13] Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *Proceedings of the Eighth International Conference on Learning Representations, ICLR*, 2020.

[14] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. *Satisfiability modulo theories*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 20, pages 825–885. IOS Press, 2009.

[15] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[16] Vaishak Belle, Guy Van den Broeck, and Andrea Passerini. Hashing-based approximate probabilistic inference in hybrid domains. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI*, pages 141–150, 2015.

[17] Vaishak Belle, Andrea Passerini, and Guy Van Den Broeck. Probabilistic inference in hybrid domains by weighted model integration. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 2770–2776, 2015.

[18] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal on Operations Research, EJOR*, 290(2):405–421, 2021.

[19] Kurt D. Bollacker, Robert P. Cook, and Patrick Tufts. Freebase: A shared database of structured general human knowledge. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, AAAI*, pages 1962–1963, 2007.

[20] Antoine Bordes, Sumit Chopra, and Jason Weston. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 615–620, 2014.

[21] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Proceedings of the Twenty-Sixth Annual Conference on Advances in Neural Information Processing Systems, NIPS*, pages 2787–2795, 2013.

[22] Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining, ICDM*, pages 74–81, 2005.

[23] Stefan Borgwardt, İsmail İlkan Ceylan, and Thomas Lukasiewicz. Ontology-mediated queries for probabilistic databases. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI*, pages 1063–1069, 2017.

[24] Stefan Borgwardt, İsmail İlkan Ceylan, and Thomas Lukasiewicz. Ontology-mediated query answering over log-linear probabilistic data. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*, pages 2711–2718, 2019.

[25] Karl Bringmann and Tobias Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Computational Geometry*, 43(6-7):601–610, 2010.

[26] Gunnar Brinkmann and Brendan D. McKay. Fast generation of planar graphs. *MATCH Communications in Mathematical and Compututer Chemistry*, 58(2):323–357, 2007.

[27] Marc Brockschmidt. GNN-FiLM: Graph neural networks with feature-wise linear modulation. In *Proceedings of the Thirty-Seventh International Conference on Machine Learning, ICML*, pages 1144–1152, 2020.

[28] Marco Cadoli and Francesco Donini. A Survey on Knowledge Compilation. *AI Communications*, 10(3-4):137–150, 1997.

[29] Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992.

[30] İsmail İlkan Ceylan, Adnan Darwiche, and Guy Van den Broeck. Open-world probabilistic databases: Semantics, algorithms, complexity. *Artificial Intelligence Journal, AIJ*, 295, 2021.

[31] Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI*, pages 1722–1730, 2014.

[32] Supratik Chakraborty, Dror Fried, Kuldeep S. Meel, and Moshe Y. Vardi. From weighted to unweighted model counting. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 689–695, 2015.

[33] Supratik Chakraborty, Kuldeep Meel, and Moshe Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI*, pages 3569–3576, 2016.

[34] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable approximate model counter. In *Proceedings of the Nineteenth International Conference on the Principles and Practice of Constraint Programming, CP*, pages 200–216, 2013.

[35] L. Sunil Chandran, Mathew C. Francis, and Naveen Sivadasan. Geometric representation of graphs in low dimension using axis parallel boxes. *Algorithmica*, 56(2):129–140, 2008.

[36] Kai-Wei Chang, Wen-tau Yih, Bishan Yang, and Christopher Meek. Typed tensor decomposition of knowledge bases for relation extraction. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1568–1579, 2014.

[37] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence Journal, AIJ*, 172(6-7):772–799, 2008.

[38] Ming-Hui Chen and Bruce W. Schmeiser. General hit-and-run Monte Carlo sampling for evaluating multidimensional integrals. *Operations Research Letters*, 19(4):161–169, 1996.

[39] Sanxing Chen, Xiaodong Liu, Jianfeng Gao, Jian Jiao, Ruofei Zhang, and Yangfeng Ji. Hitter: Hierarchical transformers for knowledge graph embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 10395–10407, 2021.

[40] Dmitry Chistikov, Rayna Dimitrova, and Rupak Majumdar. Approximate counting in SMT and value estimation for probabilistic programs. *Acta Informatica*, 54(8):729–764, 2017.

[41] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In *Proceedings of the Fourth International Conference on Learning Representations, ICLR*, 2016.

[42] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM symposium on Theory of Computing, STOC*, pages 151–158. ACM, 1971.

[43] George Cybenko. Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:183–192, 1989.

[44] Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research, JAIR*, 17(1):229–264, 2002.

[45] George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. Coloring graph neural networks for node disambiguation. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI*, 2020.

[46] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic prolog and its application in link discovery. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence, IJCAI*, pages 2462–2467, 2007.

[47] Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Lifted rule injection for relation embeddings. In *Proceedings of the 2016 Annual Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1389–1399, 2016.

[48] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2D knowledge graph embeddings. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI*, pages 1811–1818, 2018.

[49] Boyang Ding, Quan Wang, Bin Wang, and Li Guo. Improving knowledge graph embedding using simple constraints. In *Proceedings of the Fifth-Sixth Annual Meeting of the Association for Computational Linguistics, ACL*, pages 110–121, 2018.

[50] Carmel Domshlak and Jörg Hoffmann. Probabilistic planning via heuristic forward search and weighted model counting. *Journal of Artificial Intelligence Research, JAIR*, 30(1):565–620, 2007.

[51] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the Twentieth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, pages 601–610, 2014.

[52] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the Twenty-Third ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, pages 135–144, 2017.

[53] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *CoRR*, abs/2003.00982, 2020.

[54] Ioannis Z. Emiris and Vissarion Fisikopoulos. Practical polytope volume approximation. *ACM Transactions on Mathematical Software, TOMS*, 44(4):38:1–38:21, 2018.

[55] Stefano Ermon, Carla Gomes, Ashish Sabharwal, and Bart Selman. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *Proceedings of the Thirtieth International Conference on Machine Learning, ICML*, pages 334–342, 2013.

[56] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *Proceedings of the Eighth Annual Conference on Learning Representations, ICLR*, 2020.

[57] Bahare Fatemi, Perouz Taslakian, David Vázquez, and David Poole. Knowledge hypergraphs: Extending knowledge graphs beyond binary relations. *CoRR*, abs/1906.00137, 2019.

[58] Jun Feng, Minlie Huang, Mingdong Wang, Mantong Zhou, Yu Hao, and Xiaoyan Zhu. Knowledge graph embedding by flexible translation. In *Proceedings of the Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning, KR*, pages 557–560, 2016.

[59] Daan Fierens, Guy Van Den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015.

[60] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben - Hur. Protein interface prediction using graph convolutional networks. In *Proceedings of the Thirtieth Annual Conference on Advances in Neural Information Processing Systems, NIPS*, pages 6530–6539, 2017.

[61] Tal Friedman and Guy Van den Broeck. Approximate knowledge compilation by online collapsed importance sampling. In *Proceedings of the Thirty-First Annual Conference on Advances in Neural Information Processing Systems, NeurIPS*, pages 8035–8045, 2018.

[62] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning, ICML*, pages 1263–1272, 2017.

[63] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS*, pages 249–256, 2010.

[64] Vibhav Gogate and Rina Dechter. Approximate inference algorithms for hybrid bayesian networks with discrete constraints. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence, UAI*, pages 209–216, 2005.

[65] Vibhav Gogate and Pedro Domingos. Probabilistic theorem proving. *Communications of the ACM*, 59(7):107–115, 2016.

[66] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In *Handbook of Satisfiability*, pages 633–654. IOS Press, 2009.

[67] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, IJCNN*, volume 2, pages 729–734, 2005.

[68] Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*, volume 47 of *Lecture Notes in Logic*. Cambridge University Press, 2017.

[69] Martin Grohe. The logic of graph neural networks. In *LICS*, 2021.

[70] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the Twenty-Second ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, pages 855–864, 2016.

[71] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In *Proceedings of the 2016 Annual Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 192–202, 2016.

[72] Víctor Gutiérrez-Basulto and Steven Schockaert. From knowledge graph embedding to ontology embedding? an analysis of the compatibility between vector space representations and rules. In *Proceedings of the Sixteenth International Conference on the Principles of Knowledge Representation and Reasoning, KR*, pages 379–388, 2018.

[73] William L. Hamilton. *Graph Representation Learning*. Morgan and Claypool Publishers, 2020.

[74] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 40(3):52–74, 2017.

[75] Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

[76] Jure Leskovec Hongyu Ren, Weihua Hu. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *Proceedings of the Eighth International Conference on Learning Representations, ICLR*, 2020.

[77] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[78] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Proceedings of the Thirty-Third Annual Conference on Advances in Neural Information Processing Systems, NeurIPS*, 2020.

[79] Harry B Hunt III, Madhav V Marathe, Venkatesh Radhakrishnan, and Richard E Stearns. The complexity of planar counting problems. *SIAM Journal on Computing*, 27(4):1142–1167, 1998.

[80] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J. Haas. MCDB: A Monte Carlo approach to managing uncertain

data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD*, pages 687–700, 2008.

[81] Mark R. Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science, TCS*, 43(2-3):169–188, 1986.

[82] Guoliang Ji, Kang Liu, Shizhu He, and Jun Zhao. Knowledge graph completion with adaptive sparse transfer matrix. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI*, pages 985–991, 2016.

[83] Ravi Kannan, László Lovász, and Miklós Simonovits. Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. *Random Structures and Algorithms*, 11(1):1–50, 1997.

[84] Richard M. Karp, Michael Luby, and Neal Madras. Monte-Carlo approximation algorithms for enumeration problems. *Algorithms*, 10(3):429–448, 1989.

[85] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference on Machine Learning, ICML*, pages 321–328, 2003.

[86] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *Proceedings of the Thirty-First Annual Conference onAdvances in Neural Information Processing Systems, NeurIPS*, pages 4289–4300, 2018.

[87] Oliver Kennedy and Christoph Koch. PIP: A database system for great and small expectations. In *Proceedings of the Twenty-Sixth International Conference on Data Engineering, ICDE*, pages 157–168, 2010.

[88] Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. In *Proceedings of the Thirty-Second Annual Conference on Advances in Neural Information Processing Systems, NeurIPS*, pages 7090–7099, 2019.

[89] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016.

[90] Sandra Kiefer, Ilia Ponomarenko, and Pascal Schweitzer. The Weisfeiler-Leman dimension of planar graphs is at most 3. *Journal of the ACM*, 66(6):44:1–44:31, 2019.

[91] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the Third International Conference on Learning Representations, ICLR*, 2015.

[92] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the Fifth International Conference on Learning Representations, ICLR*, 2017.

[93] Samuel Kolb, Martin Mladenov, Scott Sanner, Vaishak Belle, and Kristian Kersting. Efficient symbolic integration for probabilistic inference. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI*, pages 5031–5037, 2018.

[94] Samuel Kolb, Paolo Morettin, Pedro Zuidberg Dos Martires, Francesco Sommavilla, Andrea Passerini, Roberto Sebastiani, and Luc De Raedt. The PyWMI framework and toolbox for probabilistic inference using weighted model integration. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI*, pages 6530–6532, 2019.

[95] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[96] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Twenty-Fifth Annual Conference on Advances in Neural Information Processing Systems, NIPS*, pages 1097–1105, 2012.

[97] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In *Proceedings of the Thirty-Fifth International Conference on Machine Learning, ICML*, pages 2869–2878, 2018.

[98] Yann LeCun. Self-supervised learning. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI, Invited Talk*, 2020.

[99] Yin Tat Lee and Santosh S. Vempala. Geodesic walks in polytopes. In *Proceedings of the Forty-Ninth Annual ACM Symposium on Theory of Computing, STOC*, pages 927–940, 2017.

[100] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI*, pages 3538–3545, 2018.

[101] Xiang Li, Luke Vilnis, Dongxu Zhang, Michael Boratko, and Andrew McCallum. Smoothing the geometry of probabilistic box embeddings. In *Proceedings of the Seventh International Conference on Learning Representations, ICLR*, 2019.

[102] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *Proceedings of the Fourth International Conference on Learning Representations, ICLR*, 2016.

[103] Leonid Libkin. *Elements of finite model theory*, volume 41. Springer, 2004.

[104] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI*, pages 2181–2187, 2015.

[105] Ye Liu, Hui Li, Alberto García-Durán, Mathias Niepert, Daniel Oñoro-Rubio, and David S. Rosenblum. MMKG: multi-modal knowledge graphs. In *Proceedings of the Sixteenth European Conference on the Semantic Web, ESWC*, pages 459–474, 2019.

[106] Yu Liu, Quanming Yao, and Yong Li. Generalizing tensor decomposition for n-ary relational knowledge bases. In *Proceedings of the The Web Conference, WWW '20*, pages 1104–1114, 2020.

[107] Andreas Loukas. What graph neural networks cannot learn: Depth vs width. In *Proceedings of the Eighth International Conference on Learning Representations, ICLR*, 2020.

[108] László Lovász and Santosh Vempala. Where to start a geometric random walk. Technical report, Microsoft Research, 2003.

[109] László Lovász and Santosh S. Vempala. Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. *Journal of Computer and System Sciences, JCSS*, 72(2):392–417, 2006.

[110] Daniel Lowd and Pedro M. Domingos. Approximate inference by compilation to arithmetic circuits. In *Proceedings of the Twenty-Third Annual Conference on Advances In Neural Information Processing Systems, NIPS*, pages 1477–1485, 2010.

[111] Michael G. Luby. *Monte-Carlo Methods for Estimating System Reliability*. PhD thesis, UC Berkeley, 1983.

[112] Xin Lv, Lei Hou, Juanzi Li, and Zhiyuan Liu. Differentiating concepts and instances for knowledge graph embedding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1971–1979, 2018.

[113] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M. Suchanek. YAGO3: A knowledge base from multilingual wikipedias. In *Proceedings of the Seventh Biennial Conference on Innovative Data Systems Research, CIDR*, 2015.

[114] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Proceedings of the Thirty-Second Annual Conference on Advances in Neural Information Processing Systems, NeurIPS*, pages 2153–2164, 2019.

[115] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *Proceedings of the Seventh International Conference on Learning Representations, ICLR*, 2019.

[116] Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In *Proceedings of the Thirty-Sixth International Conference on Machine Learning, ICML*, pages 4363–4371, 2019.

[117] Pedro Martires, Anton Dries, and Luc De Raedt. Exact and approximate weighted model integration with probability density functions using knowledge compilation. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*, pages 7825–7833, 2019.

[118] K. Meel, A. Shrotri, and M. Vardi. Not all FPRASs are equal: Demystifying FPRASs for DNF-counting. *Constraints*, 24(3-4):211–233, 2018.

[119] Kuldeep Meel, Aditya Shrotri, and Moshe Vardi. On hashing-based approaches to approximate dnf-counting. In *Proceedings of the Thirty-Seventh Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, pages 41:1–41:14, 2017.

[120] Péter Mernyei, Konstantinos Meichanetzidis, and İsmail İlkan Ceylan. Equivariant quantum graph circuits. In *Proceedings of the Thirty-Ninth International Conference on Machine Learning, ICML*, 2022.

[121] David Merrell, Aws Albarghouthi, and Loris D'Antoni. Weighted model integration with orthogonal transformations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, pages 4610–4616, 2017.

[122] Johannes Messner, Ralph Abboud, and İsmail İlkan Ceylan. Temporal knowledge graph completion using box embeddings. In *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI*, 2022.

[123] Pasquale Minervini, Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Adversarial sets for regularising neural link predictors. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI*, 2017.

[124] Tom M. Mitchell, William W. Cohen, Estevam R. Hruschka Jr., Partha P. Talukdar, Bo Yang, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matt Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapandula Nakashole, Emmanouil A. Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. Never-ending learning. *Communications of the ACM*, 61(5):103–115, 2018.

[125] Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. Advanced SMT techniques for weighted model integration. *Artificial Intelligence Journal, AIJ*, 275:1 – 27, 2019.

[126] Paolo Morettin, Pedro Zuidberg Dos Martires, Samuel Kolb, and Andrea Passerini. Hybrid probabilistic inference with logical and algebraic constraints: a survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI*, 2021.

[127] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*, pages 4602–4609, 2019.

[128] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, UAI*, pages 467–475, 1999.

[129] Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak A. Rao, and Bruno Ribeiro. Relational pooling for graph representations. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the Thirty-Sixth International Conference on Machine Learning, ICML*, volume 97, pages 4663–4673, 2019.

[130] Deepak Nathani, Jatin Chauhan, Charu Sharma, and Manohar Kaul. Learning attention-based embeddings for relation prediction in knowledge graphs. In *Proceedings of the Fifty-Seventh Annual Conference of the Association for Computational Linguistics, ACL*, pages 4710–4723, 2019.

[131] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the Twenty-Eighth International Conference on Machine Learning, ICML*, pages 809–816, 2011.

[132] Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. k-hop graph neural networks. *Neural Networks*, 130:195–205, 2020.

[133] Giannis Nikolentzos and Michalis Vazirgiannis. Random walk graph neural networks. In *Proceedings of the Thirty-Third Annual Conference on Advances in Neural Information Processing Systems, NeurIPS*, pages 16211–16222, 2020.

[134] Judea Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the National Conference on Artificial Intelligence, AAAI*, pages 133–136, 1982.

[135] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[136] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1532–1543, 2014.

[137] Marcelo Prates, Pedro Avelar, Henrique Lemos, Luís Lamb, and Moshe Vardi. Learning to solve NP-complete problems-a graph neural network for the decision TSP. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*, pages 4731–4738, 2019.

[138] J. Scott Provan and Michael O. Ball. The Complexity of Counting Cuts And Of Computing The Probability That A Graph Is Connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.

[139] Omri Puny, Heli Ben-Hamu, and Yaron Lipman. From graph low-rank global attention to 2-FWL approximation. *CoRR*, abs/2006.07846v1, 2020.

[140] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

[141] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1):107–136, 2006.

[142] Fred S. Roberts. On the Boxicity and Cubicity of a graph. In *Recent Progress in Combinatorics, Academic Press*, pages 301–310, 1968.

[143] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL*, pages 1119–1129, 2015.

[144] Dan Roth. On the Hardness of Approximate Reasoning. *Artificial Intelligence Journal, AIJ*, 82(1-2):273–302, 1996.

[145] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You CAN teach an old dog new tricks! On training knowledge graph embeddings. In *Proceedings of the Eighth International Conference on Learning Representations, ICLR*, 2020.

[146] Tian Sang, Paul Bearne, and Henry Kautz. Performing bayesian inference by weighted model counting. In *Proceedings of the Twentieth AAAI Conference on Artificial Intelligence, AAAI*, pages 475–482, 2005.

[147] Scott Sanner and Ehsan Abbasnejad. Symbolic variable elimination for discrete and continuous graphical models. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, AAAI*, page 1954–1960, 2012.

[148] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining, SDM*, pages 333–341, 2021.

[149] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[150] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *Proceedings of the Fifteenth International Conference on the Semantic Web, ESWC*, pages 593–607, 2018.

[151] Bart Selman and Henry Kautz. Knowledge compilation and theory approximation. *Journal of the Association for Computing Machinery*, 43(2):193–224, 1996.

[152] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo Leonardo de Moura, and David Dill. Learning a sat solver from single-bit supervision. In *Proceedings of the Seventh International Conference on Learning Representations, ICLR*, 2019.

[153] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

[154] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018.

[155] Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. Reasoning with neural tensor networks for knowledge base completion. In *Proceedings of the Twenty-Sixth Annual Conference on Advances in Neural Information Processing Systems, NIPS*, pages 926–934, 2013.

[156] Larry Stockmeyer. The complexity of approximate counting. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing , STOC*, pages 118–126. ACM, 1983.

[157] Sandeep Subramanian and Soumen Chakrabarti. New embedded representations and evaluation protocols for inferring transitive relations. In *Proceedings of the Fourty-First International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR*, pages 1037–1040, 2018.

[158] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*, volume 3. Morgan & Claypool, 2011.

[159] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. RotatE: Knowledge graph embedding by relational rotation in complex space. In *Proceedings of the Seventh International Conference on Learning Representations, ICLR*, 2019.

[160] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the Twenty-Seventh Annual Conference on Advances in Neural Information Processing Systems, NIPS*, pages 3104–3112, 2014.

[161] Thomas Pellissier Tanon, Gerhard Weikum, and Fabian M. Suchanek. YAGO 4: A reason-able knowledge base. In *Proceedings of the Seventeenth European Semantic Web Conference, ESWC*, volume 12123, pages 583–596, 2020.

[162] Komal K. Teru, Etienne Denis, and Will Hamilton. Inductive relation prediction by subgraph reasoning. In *Proceedings of the Thirty-Seventh International Conference on Machine Learning, ICML*, pages 9448–9457, 2020.

[163] Seinosuke Toda. On the computational power of PP and +P. In *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science, FOCS*, pages 514–519, 1989.

[164] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the Third Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, 2015.

[165] Théo Trouillon, Éric Gaussier, Christopher R. Dance, and Guillaume Bouchard. On inductive abilities of latent factor models for relational learning. *Journal of Artificial Intelligence Research, JAIR*, 64:21–53, 2019.

[166] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the Thirty-Third International Conference on Machine Learning, ICML*, pages 2071–2080, 2016.

[167] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[168] Leslie G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science, TCS*, 8(2):189–201, 1979.

[169] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *Proceedings of the Sixth International Conference on Learning Representations, ICLR*, 2018.

[170] Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. Order-embeddings of images and language. In Yoshua Bengio and Yann LeCun, editors, *Proceedings of the Fourth International Conference on Learning Representations, ICLR*, 2016.

[171] Luke Vilnis, Xiang Li, Shikhar Murty, and Andrew McCallum. Probabilistic Embedding of Knowledge Graphs with Box Lattice Measures. In *Proceedings of the Fifty-Sixth Annual Meeting of the Association for Computational Linguistics, ACL*, pages 263–272, 2018.

[172] Vishy Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research, JMLR*, 11:1201–1242, 2010.

[173] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In *Proceedings of the Twenty-Seventh ACM International Conference on Information and Knowledge Management, CIKM*, 2018.

[174] Jue Wang and Pedro Domingos. Hybrid Markov logic networks. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI*, pages 1106–1111, 2008.

[175] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[176] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI*, pages 1112–1119, 2014.

[177] Boris Weisfeiler and Andrei A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

[178] Yair Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12(1):1–41, 2000.

[179] Jianfeng Wen, Jianxin Li, Yongyi Mao, Shini Chen, and Richong Zhang. On the Representation and Embedding of Knowledge Bases beyond Binary Relations. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1300–1307, 2016.

[180] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In *Proceedings of the Twenty-Third International World Wide Web Conference, WWW*, pages 515–526, 2014.

[181] Ruobing Xie, Zhiyuan Liu, and Maosong Sun. Representation learning of knowledge graphs with hierarchical types. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI*, pages 2965–2971, 2016.

[182] Chenyan Xiong, Russell Power, and Jamie Callan. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proceedings of the Twenty-Sixth International Conference on World Wide Web, WWW*, pages 1271–1279, 2017.

[183] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the Seventh International Conference on Learning Representations, ICLR*, 2019.

[184] Bishan Yang and Tom M. Mitchell. Leveraging knowledge bases in LSTMs for improving machine reading. In *Proceedings of the Fifty-Fifth Annual Meeting of the Association for Computational Linguistics, ACL, Volume 1: Long Papers*, pages 1436–1446, 2017.

[185] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the Third International Conference on Learning Representations, ICLR*, 2015.

[186] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the Twenty-Fourth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, pages 974–983, 2018.

[187] KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard S. Zemel, and Xaq Pitkow. Inference in probabilistic graphical models by graph neural networks. In *Workshop Track Proceedings of the Sixth International Conference on Learning Representations, ICLR*, 2018.

[188] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay S. Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *Proceedings of the Thirty-First Annual Conference on Advances in Neural Information Processing Systems, NeurIPS*, pages 6412–6422, 2018.

[189] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. Graph transformer networks. In *Proceedings of the Thirty-Second Annual Conference on Advances in Neural Information Processing Systems, NeurIPS*, pages 11960–11970, 2019.

[190] Zhe Zeng, Paolo Morettin, Fanqi Yan, Antonio Vergari, and Guy Van den Broeck. Probabilistic inference with algebraic constraints: Theoretical limits and practical approximations. In *Proceedings of the Thirty-Third Annual Conference on Advances in Neural Information Processing Systems, NeurIPS*, pages 11564–11575, 2020.

[191] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.

Appendix A

A.1 Experimental Details for BoxE (Chapter 3)

In this section, we give further details on the experiments that we have conducted. In particular, we report details of every dataset, the hyperparameter tuning setup used when training BoxE, as well as the final set of hyperparameters used in the configurations whose results we report.

All reported results for the KGC and KBC experiments are average results from 3 training runs, and empirically have very small variance. In particular, all MRR values fluctuate by no more than 0.002 between runs across all datasets. BoxE is trained using the Adam optimiser [91], to optimise negative sampling loss [159]. Training for every run was conducted on a Haswell CPU node with 12 cores, 64 GB RAM, and a V100 GPU. Hyper-parameter tuning was conducted over its learning rate λ, dimensionality d, loss margin γ, distance order x, and number of negative examples m. For all BoxE experiments, points and boxes were projected into the hypercube $[-1, 1]^d$, a bounded space, by simply applying the hyperbolic tangent function tanh element-wise on all final embedding representations.

Learning rate was varied between 10^{-6} and 10^{-2}, with root values of 1,2,5 and exponents from -6 to -2, i.e., $10^{-6}, 2\times10^{-6}, 5\times10^{-6}$, etc. . Margin was varied between 3 and 24 inclusive, in increments of 1.5, and in increments of 1 between 3 and 6. Adversarial temperature was varied between the integer values of 1 and 4 inclusive, and the number of negative samples was varied between 50, 100, and 150. Across all knowledge graph datasets, we additionally ran experiments with *data augmentation*, such that, for every relation r, a distinct inverse relation r' is defined, and every fact $r(e_1, e_2)$ is augmented with another fact $r'(e_2, e_1)$. This setting, however, was only marginally beneficial on YAGO3-10, yielding a slightly improved MR.

Finally, the distance order was set to either 1 (Manhattan distance) or 2 (Euclidian distance), and batch sizes (for number of positive examples) were varied between

Table A.1: Hyper-parameter settings of BoxE over different datasets.

Dataset	Embedding Dimension	Margin	Learning Rate	Adversarial Temperature	Negative Samples	Distance Order	Batch Size	Data Augmentation
FB15k-237(u)	500	12	1×10^{-4}	0.0	100	1	1024	No
FB15k-237(a)	1000	3	5×10^{-5}	4.0	100	2	1024	No
WN18RR(u)	500	5	1×10^{-3}	0.0	150	2	512	No
WN18RR(a)	500	3	1×10^{-3}	2.0	100	2	512	No
YAGO3-10(u)	200	10.5	1×10^{-3}	0.0	150	2	4096	Yes
YAGO3-10(a)	200	6	1×10^{-3}	2.0	150	2	4096	Yes
JF17K(u)	200	15	2×10^{-3}	0.0	100	2	1024	N/A
JF17K(a)	200	5	1×10^{-4}	2.0	100	2	1024	N/A
FB-AUTO(u)	200	18	2×10^{-3}	0.0	100	2	1024	N/A
FB-AUTO(a)	200	9	5×10^{-4}	2.0	100	2	1024	N/A
SportsNELL	200	6	1×10^{-3}	0.0	100	2	1024	No
SportsNELL+RI	200	6	1×10^{-3}	0.0	100	2	1024	No
YAGO-39K	100	6	1×10^{-3}	0.0	100	2	4096	No

all powers of two between 2^6 and 2^{12} inclusive. Hyper-parameters were initially selected randomly and tuned using grid search. The set of used hyperparameters in experiments is shown in Table A.1.

Aside from the reported hyperparameter settings, we have also attempted to fix box sizes, either in a hard fashion or softly by setting maximum total size. Hard sizes were based on statistical popularity of relations, whereas soft totals were tuned. However, neither of these settings yielded any improvements, and in fact both have been mostly detrimental to performance. This, in fact, further highlights the importance of box size variability to obtaining good predictive performance. Interestingly, it also confirms that statistical popularity alone is not sufficient to establish optimal box sizing. We also remain very confident that BoxE performance can further improve in the future, as more dedicated empirical studies and more comprehensive and bespoke tuning methods are applied.

A.2 Experimental Details for MLP-X (Chapter 4)

A.2.1 WikiAlumni Experiments

In our experiments on WikiAlumni, all MLP-X models are trained on a Haswell CPU node with 12 cores, 64 GB RAM, and a V100 GPU. We report (i) the mean peak validation accuracy across 5 runs for node classification, and (ii) the mean of MR/MRR/Hits@10 results on all dropped edges across 5 runs, using the best average MRR-Hits@10 as the validation metric for link prediction. In our experiments, we

Table A.2: Hyper-parameter settings for MLP-X models with *negative sampling loss*. Here, LR denotes the learning rate used by the Adam optimiser.

Model	WikiAlumni-80%			WikiAlumni-90%			WikiAlumni		
	LR	Margin	Batch Size	LR	Margin	Batch Size	LR	Margin	Batch Size
MLP-TransE	10^{-4}	15	1024	10^{-4}	15	1024	10^{-4}	15	1024
MLP-RotatE	10^{-4}	21	1024	10^{-4}	21	1024	10^{-4}	24	1024
MLP-BoxE	10^{-3}	9	512	10^{-3}	6	512	10^{-3}	6	512
TransE	10^{-3}	9	1024	10^{-3}	9	1024	10^{-3}	9	1024
RotatE	10^{-3}	9	1024	10^{-3}	9	1024	10^{-3}	9	1024
BoxE	10^{-3}	4	512	10^{-3}	5	512	10^{-3}	5	512

Table A.3: Hyper-parameter settings for MLP-X models with *cross-entropy loss*. Here, LR denotes the learning rate used by the Adam optimiser.

Model	WikiAlumni-80%		WikiAlumni-90%		WikiAlumni	
	LR	Batch Size	LR	Batch Size	LR	Batch Size
MLP-TransE	10^{-4}	128	10^{-4}	128	10^{-4}	128
MLP-RotatE	10^{-4}	1024	10^{-4}	128	10^{-4}	128
MLP-BoxE	10^{-3}	512	10^{-3}	512	10^{-3}	128
TransE	10^{-3}	1024	10^{-3}	1024	10^{-3}	1024
RotatE	10^{-3}	1024	10^{-3}	1024	10^{-3}	1024
BoxE	10^{-3}	512	10^{-3}	512	10^{-3}	512

observed that the error bars in terms of accuracy (0.2%) and MRR (0.1%) are small and consistent across models, and thus did not report them for better visibility.

When training the models, we set both all MLPs to have two hidden layers of size 1000, each using the ReLU activation layer. Moreover, we conduct training using the Adam optimiser [91], 100 negative samples per positive fact, and a learning rate chosen from the set $\{10^{-4}, 10^{-3}\}$. Negative sampling was conducted as described in the main paper, by negatively sampling over classes for class facts, to exploit the mutual exclusion between target classes. Furthermore, we experimented with λ, the embedding scale (cf. Equation (4.1)) from the set $\{0.5, 1]$ for λ, the embedding scale, and found that 0.5 achieved better results across all experiments and provided better regularisation for the models.

We tune batch size with values from the interval {128, 256, 512, 1024}. For negative sampling loss, we tune the margin from the range {1, 2, 3, 4, 5, 6, 9, 12, 15,

Table A.4: Hyper-parameters for MLP-BoxE on the node classification benchmarks.

Dataset	Margin	Dimensionality	Dropout
CiteSeer	2	20	0.8
Cora	2	20	0
PubMed	2	20	0.5
OGBN-arXiv	3	256	0

18, 21, 24, 27, 30}. The hyper-parameters corresponding to our results for negative sampling loss are reported in Table A.2, and those corresponding to our results for cross-entropy are reported in Table A.3.

Remark 1. We note that, in our experiments, the optimal hyper-parameters for node classification and link prediction with/without features coincide, despite being tuned independently. This further highlights the interdependence between both tasks, and confirm the point being made in Chapter 4. Furthermore, the optimal hyperparameters for link prediction for all configurations without classes are identical to those for the corresponding configurations with classes, i.e., the same hyperparameters achieve best performance for a given configuration with and without classes, all else equal.

Remark 2. When preparing our experimental setup, we also considered bilinear models, namely TuckER and ComplEx, but observed that these models performed substantially worse than their translational counterparts. For instance, ComplEx would only achieve a best node classification accuracy of 35.5%, whereas TuckER would achieve at most 28.5% on entity classification, despite extensive tuning efforts. We then found that this performance is primarily due to the high mean rank both models achieve, which negatively impacts node classification performance. More precisely, both models have substantially worse MR than the three used MLP-X models, and this implies that link prediction quality, though potentially better in specific cases, is worse overall, and thus predicted links are much noisier, leading to overall worse performance in our setting.

A.2.2 Citation Network Experiments

Citation networks experiments ran on the same computing infrastructure as WikiAlumni experiments. The full hyper-parameters used by MLP-BoxE on all four citation network datasets are shown in Table A.4.

Table A.5: Hyper-parameter configurations (Learning rate λ and probability distribution p) for all experiments.

Dataset	Exp		CExp	
	λ	p	λ	p
GCN	1×10^{-4}	N/A	1×10^{-4}	N/A
GCN-12.5%RNI	2×10^{-4}	N	2×10^{-4}	N
GCN-50%RNI	2×10^{-4}	N	2×10^{-4}	N
GCN-87.5%RNI	2×10^{-4}	N	5×10^{-4}	N
GCN-RNI	5×10^{-4}	N	5×10^{-4}	N
3-GCN	5×10^{-4}	N/A	2×10^{-4}	N/A

A.2.3 Bag-of-words Experiment on WikiAlumni

Interestingly, the optimal hyper-parameters for MLP-TransE and MLP-RotatE exactly coincide with those for negative sampling loss in the entity classification setting (cf. Table A.2), whereas the optimal margin for MLP-BoxE is 3, with all other hyper-parameters otherwise identical to the optimal BoxE entity classification configuration.

A.3 Experimental Details for MPNN-RNI (Chapter 5)

All GCN models with (partially or completely) deterministic initial node embeddings map a 2-dimensional one-hot encoding of node type (literal or disjunction) to a $d = 64$-dimensional embedding space. Furthermore, the final prediction for every graph is computed by aggregating all node embeddings using the max function, and then passing the result through a multi-layer perceptron (MLP) of 3 layers with dimensionality 64, 32 and 2 respectively. The activation function for the first two MLP layers is the ELU function [41], and the softmax function is used to make a final prediction at the final MLP layer.

All neural networks in this work are optimised using the Adam optimiser [91]. All training is conducted with a fixed learning rate λ, for fairer comparison between all models. Initially, decaying learning rates were used, but these were discarded, as they yielded sub-optimal convergence for all GCN-RNI models. Finally, all experiments were run on a V100 GPU. Detailed hyper-parameters, namely learning rate λ and RNI distribution p, per model on every evaluation dataset are shown in Table A.5.

A.4 Datasheets for Datasets: WikiAlumni

The "Datasheets for Datasets" framework form for WikiAlumni is appended to the end of this manuscript.

Datasheet for WikiAlumni

I. Motivation For Datasheet Creation

A. Why was the datasheet created? (e.g., was there a specific task in mind? was there a specific gap that needed to be filled?)

WikiAlumni was created to introduce and study the problems of *node classification given relational incompleteness*, and *link prediction given node features and classes*. WikiAlumni includes real-world node features, representative classes, and rich, heterogeneous relations. These classes and relations also address limitations in current node classification and link prediction benchmarks. In particular, the relations in WikiAlumni are refined, semantically distinct, and do not share redundancies, and its classes are not directly deducible from relations, and are challenging to predict.

B. Has the dataset been used already? If so, where are the results so others can compare (e.g., links to published papers)?

Our paper presenting this dataset is the first to use it, and conducts conducts an extensive benchmarking to highlight the performance of different models on this dataset.

C. What (other) tasks could the dataset be used for?

In addition to the aforementioned tasks, WikiAlumni can also be used as a benchmark for standard link prediction, node classification, and entity classification.

D. Who funded the creation of the dataset?

This dataset is a carefully selected and processed subset of the YAGO4 knowledge base [1], and thus did not require funding for its development.

E. Any other comment?

None.

II. Datasheet Composition

A. What are the instances?(that is, examples; e.g., documents, images, people, countries) Are there multiple types of instances? (e.g., movies, users, ratings; people, interactions between them; nodes, edges)

Instances are knowledge graph entities, which can represent people, locations, creative works, institutions, languages, and professions, in keeping with YAGO4. These entities appear in facts, which could be unary (class information) or binary (relational information).

B. How many instances are there in total (of each type, if appropriate)?

WikiAlumni includes 52,678 entities connected by 121,836 edges, and its edge subsets WikiAlumni-90% and WikiAlumni-80% contain 109,652 and 97.468 edges, respectively.

C. What data does each instance consist of ? "Raw" data (e.g., unprocessed text or images)? Features/attributes? Is there a label/target associated with instances? If the instances related to people, are subpopulations identified (e.g., by age, gender, etc.) and what is their distribution?

Each instance consists of a text-based unique identifier, which is associated with a 300-dimensional feature vector, corresponding to processed textual descriptions for their respective entities. Furthermore, relations are also represented with unique identifiers, such that facts in WikiAlumni are represented using a combination of relation and entity identifiers: a unary fact connects one entity identifier to a class identifier, and a binary fact connects one entity identifiers to a relation identifier.

D. Is there a label or target associated with each instance? If so, please provide a description.

For node classification under relational incompleteness, the target is to predict classes for entities whose class is not provided in the training set. For link prediction with node features and classes, the target is to predict missing relations among pairs of entities.

E. Is any information missing from individual instances? If so, please provide a description, explaining why this information is missing (e.g., because it was unavailable). This does not include intentionally removed information, but might include, e.g., redacted text.

To our knowledge, no information is missing from the dataset, and every effort has been made to provide every entity with descriptive features, and reliably preserve the information available in YAGO4.

F. Are relationships between individual instances made explicit (e.g., users' movie ratings, social network links)? If so, please describe how these relationships are made explicit.

Relationships between entities are explicitly encoded using knowledge base facts.

G. Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set? If the dataset is a sample, then what is the larger set? Is the sample representative of the larger set (e.g., geographic coverage)? If so, please describe how this representativeness was validated/verified. If it is not representative of the larger set, please describe why not (e.g., to cover a more diverse range of instances, because instances were withheld or unavailable).

The dataset is a sample from the larger YAGO4 knowledge graph, where entities were selected based on their connectivity and appearance in the relevant, selected classes for the node classification task, and where relations were selected to ensure connectivity between entities and provide diverse semantic structures.

H. Are there recommended data splits (e.g., training, development/validation, testing)? If so, please provide a description of these splits, explaining the rationale behind them.

We propose data splits for node classification, where we offer over half the entity labels for training, so as to support learning for this challenging task. This is also in keeping with recent trends in proposed datasets, e.g., OGBN-arXiv, [2], and further distances from initial paradigms, where minimal labels (20 instances per class) were provided and used to train on small datasets, e.g., CiteSeer [3], [4].

I. Are there any errors, sources of noise, or redundancies in the dataset? If so, please provide a description.

The facts in WikiAlumni are based on structured data, and extracted from YAGO4, whereas the features correspond to processed textual descriptions from the introduction of the English Wikipedia page of every entity. Therefore, WikiAlumni is unlikely to include substantial noise, as YAGO4 largely only includes correct facts, and as Wikipedia introductions typically are concise and descriptive of their subject.

J. Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)? If it links to or relies on external resources, a) are there guarantees that they will exist, and remain constant, over time; b) are there official archival versions of the complete dataset (i.e., including the external resources as they existed at the time the dataset was created); c) are there any restrictions (e.g., licenses, fees) associated with any of the external resources that might apply to a future user? Please provide descriptions of all external resources and any restrictions associated with them, as well as links or other access points, as appropriate.

The dataset is self-contained, and includes all necessary information for its target tasks. a) It will eventually be maintained on the University of Oxford servers. b) The YAGO4 version used in building this dataset is readily available online at https://yago-knowledge.org/data/yago4/en/2020-02-24/

Any other comments? None

III. Collection Process

A. What mechanisms or procedures were used to collect the data (e.g., hardware apparatus or sensor, manual human curation, software program, software API)? How were these mechanisms or procedures validated?

The data was collected by obtaining WikiData URIs for every entity from the earlier YAGO4 repository, retrieving its corresponding English Wikipedia page, and scraping its introduction to compute node features. This was done automatically, and was manually verified by inspecting retrieved features for a subset of entities.

B. How was the data associated with each instance acquired? Was the data directly observable (e.g., raw text, movie ratings), reported by subjects (e.g., survey responses), or indirectly inferred/derived from other data (e.g., part-of-speech tags, model-based guesses for age or language)? If data was reported by subjects or indirectly inferred/derived from other data, was the data validated/verified? If so, please describe how.

The data is directly observable, and corresponds to English Wikipedia pages parsed on April 25 2021.

C. If the dataset is a sample from a larger set, what was the sampling strategy (e.g., deterministic, probabilistic with specific sampling probabilities)?

The sampling strategy is based on class prominence. In particular, we selected entities appearing in the most prominent target classes (where we chose "prominent" to imply a class with 700 entities or more), such that, for every selected class, the 700 most connected (in terms of node degree) are chosen, along with their edges and neighbors. Furthermore, we only considered class considering to institutions in North America to minimize homophily in the dataset.

D. Who was involved in the data collection process (e.g., students, crowdworkers, contractors) and how were they compensated (e.g., how much were crowdworkers paid)?

Ralph Abboud, a PhD student at the University of Oxford, conducted this data collection as part of their doctoral research.

E. Over what timeframe was the data collected? Does this timeframe match the creation timeframe of the data associated with the instances (e.g., recent crawl of old news articles)? If not, please describe the timeframe in which the data associated with the instances was created.

The feature data was collected over a few hours of WikiData scraping on 25 April 2021. The data scraped,

however, appears in YAGO4, and thus could already have been present for several years prior, and even have earlier description versions.

IV. Data Preprocessing

A. Was any preprocessing/cleaning/labeling of the data done (e.g., discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)? If so, please provide a description. If not, you may skip the remainder of the questions in this section.

Words in entity descriptions were mapped to their respective GloVe embeddings [5] and averaged to yield entity feature vectors.

B. Was the "raw" data saved in addition to the preprocessed/cleaned/labeled data (e.g., to support unanticipated future uses)? If so, please provide a link or other access point to the "raw" data.

The raw data was saved as a cache file during data scraping, and is made available as a serialised dictionary file in the supplementary material.

C. Is the software used to preprocess/clean/label the instances available? If so, please provide a link or other access point.

The processing of YAGO4 and mapping to GloVe embeddings is done via a simple Python script. This script is also available in the supplementary files.

D. Does this dataset collection/processing procedure achieve the motivation for creating the dataset stated in the first section of this datasheet? If not, what are the limitations?

Yes, as it produces balanced classes and returns semantically diverse relations, while also providing informative, real-world features for all entities in the dataset.

E. Any other comments

None

V. Dataset Distribution

A. How will the dataset be distributed? (e.g., tarball on website, API, GitHub; does the data have a DOI and is it archived redundantly?)

Upon publication of its corresponding manuscript, the dataset will be distributed on the University of Oxford website.

B. When will the dataset be released/first distributed? What license (if any) is it distributed under?

The dataset will be released upon the publication of the paper, and will be distributed under a CC-BY-SA license.

C. Are there any copyrights on the data?

No

D. Are there any fees or access/export restrictions?

No

E. Any other comments?

No

VI. Dataset Maintenance

A. Who is supporting/hosting/maintaining the dataset?

The dataset will be supported and maintained by Dr İsmail İlkan Ceylan, and Ralph Abboud, at the University of Oxford, and will be hosted on the University of Oxford's Department of Computer Science website.

B. Will the dataset be updated? If so, how often and by whom?

The dataset is designed to be static, and thus we do not anticipate any regular updates.

C. How will updates be communicated? (e.g., mailing list, GitHub)

Not applicable.

D. If the dataset becomes obsolete how will this be communicated?

Not applicable.

E. Is there a repository to link to any/all papers/systems that use this dataset?

The dataset page, upon creation, will link to all related papers and repositories.

F. If others want to extend/augment/build on this dataset, is there a mechanism for them to do so? If so, is there a process for tracking/assessing the quality of those contributions. What is the process for communicating/distributing these contributions to users?

The dataset and its creation pipeline will be freely and publicly available, and others are welcome to modify the extraction process, or add new components. We are happy to share any constructive contributions on the dataset page when these arise.

VII. Legal and Ethical Considerations

A. Were any ethical review processes conducted (e.g., by an institutional review board)? If so, please provide a description of these review processes, including the outcomes, as well as a link or other access point to any supporting documentation.

Not applicable

B. Does the dataset contain data that might be considered confidential (e.g., data that is protected by legal privilege or by doctorpatient confidentiality, data that includes the content of individuals non-public communications)? If so, please provide a description.

Not applicable.

C. Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? If so, please describe why

No. The dataset only provides anonymized identifiers for entities and relations, as well as real-valued feature vectors.

D. Does the dataset relate to people? If not, you may skip the remaining questions in this section.

Yes, but in a very loose sense, as people with information in YAGO4 can be represented by entities in the dataset.

E. Does the dataset identify any subpopulations (e.g., by age, gender)? If so, please describe how these subpopulations are identified and provide a description of their respective distributions within the dataset.

The dataset does not identify any subpopulations. It only includes publicly available relational information about place of birth, age, etc. for individuals in the YAGO4 knowledge graph.

F. Is it possible to identify individuals (i.e., one or more natural persons), either directly or indirectly (i.e., in combination with other data) from the dataset? If so, please describe how.

Not applicable. All information presented in the dataset is publicly available on Wikipedia and Wikidata, and cannot be used to identify information beyond what is publicly known.

G. Does the dataset contain data that might be considered sensitive in any way (e.g., data that reveals racial or ethnic origins, sexual orientations, religious beliefs, political opinions or union memberships, or locations; financial or health data; biometric or genetic data; forms of government identification, such as social security numbers; criminal history)? If so, please provide a description.

Not applicable.

H. Did you collect the data from the individuals in question directly, or obtain it via third parties or other sources (e.g., websites)?

Not applicable.

I. Were the individuals in question notified about the data collection? If so, please describe (or show with screenshots or other information) how notice was provided, and provide a link or other access point to, or otherwise reproduce, the exact language of the notification itself.

Not applicable.

J. Did the individuals in question consent to the collection and use of their data? If so, please describe (or show with screenshots or other information) how consent was requested and provided, and provide a link or other access point to, or otherwise reproduce, the exact language to which the individuals consented.

Not applicable.

K. If consent was obtained, were the consenting individuals provided with a mechanism to revoke their consent in the future or for certain uses? If so, please provide a description, as well as a link or other access point to the mechanism (if appropriate).

Not applicable.

L. Has an analysis of the potential impact of the dataset and its use on data subjects (e.g., a data protection impact analysis)been conducted? If so, please provide a description of this analysis, including the outcomes, as well as a link or other access point to any supporting documentation.

Not applicable.

M. Any other comments?

None

References

[1] Thomas Pellissier Tanon, Gerhard Weikum, and Fabian M. Suchanek. YAGO 4: A reason-able knowledge base. In Andreas Harth, Sabrina Kirrane, Axel-Cyrille Ngonga Ngomo, Heiko Paulheim, Anisa Rula, Anna Lisa Gentile, Peter Haase, and Michael Cochez, editors, *ESWC*, volume 12123 of *Lecture Notes in Computer Science*, pages 583–596. Springer, 2020.

[2] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020.

[3] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Mag.*, 29(3):93–106, 2008.

[4] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018.

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.