Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning
Special Session on KR and Machine Learning

Learning Typed Rules over Knowledge Graphs

Hong Wu[1] , **Zhe Wang**[1] , **Kewen Wang**[1] , **Yi-Dong Shen**[2]
[1]School of Information and Communication Technology, Griffith University, Australia
[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
hong.wu2@griffithuni.edu.au, {zhe.wang, k.wang}@griffith.edu.au, ydshen@ios.ac.cn

Abstract

Rule learning from large datasets has regained extensive interest as rules are useful for developing explainable approaches to many applications in knowledge graphs. However, existing methods for rule learning are still limited in terms of rule expressivity and rule quality. This paper presents a new method for learning typed rules by employing type information. Our experimental evaluation shows the superiority of our system compared to state-of-the-art rule learners. In particular, we demonstrate the usefulness of typed rules in reasoning for link prediction.

1 Introduction

Knowledge graphs (KGs) are a special kind of knowledge bases that highlight interconnection among entities and represent such knowledge as graphs. They have found wide applications in information retrieval, recommender systems, question answering, and semantic data integration (Ji et al. 2020). A number of large-scale and general-purpose knowledge graphs have been developed, such as WordNet (Miller 1995), YAGO (Suchanek, Kasneci, and Weikum 2007), and Freebase (Bollacker et al. 2008). Although significant progress has been made, some important research challenges remain open. One challenge is to extract high-level schema information from large KGs, and another is KG completion or link prediction, as most KGs are still highly incomplete.

Hence, the problem of learning first-order rules over KGs has recently regained extensive interest, as rules provide a promising tool for KG reasoning in various KG applications. Compared to methods purely based on deep neural networks, logic rules are human-comprehensible knowledge and naturally offer explainability of reasoning. Several scalable rule learners have been proposed, including (Galárraga et al. 2015; Chen, Wang, and Goldberg 2016b; Yang, Yang, and Cohen 2017; Omran, Wang, and Wang 2018; Meilicke et al. 2019; Ahmadi et al. 2020; Pirrò 2020), and the learned rules have been successfully applied to link prediction and demonstrated competitive performance (Meilicke et al. 2019; Pirrò 2020).

Several recent rule learners can only extract so-called *closed path rules* (or *CP-rules*) (Yang et al. 2014; Chen, Wang, and Goldberg 2016b; Omran, Wang, and Wang 2018; Pirrò 2020). A major shortcoming of such rules is their limited expressivity. Consider an example, $B(x, y) \wedge C(y, z) \rightarrow N(x, z)$ is a CP-rule for the head predicate $N(x, z)$, which means that if person x was born in city y of country z, then x has the nationality z. So this CP-rule is essentially intended to express the typed rule $B(x : person, y : city) \wedge C(y : city, z : country) \rightarrow N(x : person, z : country)$. In fact, this typed rule can be expressed as a first order Horn rule $B(x, y) \wedge C(y, z) \wedge person(x) \wedge city(y) \wedge country(z) \rightarrow N(x, z)$. But such Horn rules cannot be expressed by the syntax of CP-rules. Thus, typed rules provide a natural and useful extension of CP-rules.

More importantly, a KG usually contains rich type information on entities (Hao et al. 2019; Zhang et al. 2020), which is useful for optimising the search of rules. However, this has not received sufficient attention in the literature on rule learning over KGs, except for ScaLeKB (Chen, Wang, and Goldberg 2016b) and RARL (Pirrò 2020), which are based on ontological pathfinding (OP) (Chen et al. 2016a). While these approaches utilise type information in both rule search and rule evaluation, they do not learn typed rules. Also, ontological pathfinding approaches typically use type information as hard constraints on the candidate rules, i.e., each of them must represent a path going through entities belonging to some types. Yet in practice, type information in KGs is often highly incomplete, which would make such hard constraints too restrictive. Besides path-based rule learning, embedding-based rule learning has recently been proposed and shows promising performance (Omran, Wang, and Wang 2018), which employs latent representations from Statistical Relational Learning.

In this paper, we propose a new rule learning approach that is able to learn typed rules and better utilises type information to guide the rule search. We adopt a combined approach of both path-based and embedding-based strategies for our rule search. In the path-based strategy, candidate rules are obtained from both ABox paths and TBox paths. Moreover, type information is employed to confine the path exploration. In the embedding-based strategy, a novel scoring function is defined and used for rule evaluation, in which type information is encoded as latent representations to refine the embedding model.

We have implemented a prototype system TyRuLe and evaluated its performance on both standard benchmarks and a real-life KG AirGraph. The experiments show that our system can learn high-quality and informative rules, and

the advantages of using type information are demonstrated through both quantitative and qualitative analysis. As a commonly used rule quality indicator, link prediction results show that TyRuLe outperforms existing rule learners and is competitive compared to embedding-based link prediction systems.

The rest of this paper is organised as follows. Section 2 provides a brief review on models for rule learning and link prediction for knowledge graphs. Section 3 introduces some basics of knowledge graphs and rule technologies. Section 4 describes technical details of our proposed TyRuLe, including KG sampling, a path-based strategy and an embedding-based strategy for rule extraction. Our experimental results are presented in Section 5. Finally, we conclude the paper in Section 6.

2 Related Work

In what follows, we discuss existing works on rule learning and link prediction that are closely related to this paper.

2.1 Rule Learning

First-order rule learning has been extensively studied in Inductive Logic Programming (ILP) literature (Muggleton 1990; Muggleton 1991; Cropper et al. 2022) by exploring the rule space through refinement operators. Classical ILP systems like QuickFOIL (Zeng, Patel, and Page 2014) cannot be used directly to handle KGs due to the lack of negative examples and the large data sizes. Recently, rule learners such as AMIE and its extensions (Galárraga et al. 2015) have been developed with the aim to handle large datasets like KGs, which use plausibility metrics adapted from association rule mining to address the lack of negative examples. Some work has been proposed to propose new completeness-aware confidence metrics to better assess the quality of rules (Pellissier Tanon et al. 2017). Also, learning non-monotonic rules have been studied (Gad-Elrab et al. 2016; Lisi and Weikum 2017).

Another group of approaches generates candidate rules by exploring paths in KGs. AnyBURL (Meilicke et al. 2019) is a bottom-up approach that substitutes entities from sampled paths with variables, to generalise path instances to patterns for rule generation. This allows AnyBURL to learn a large number of rules, which together provide more accurate link prediction than many of the embedding-based approaches. Ontological pathfinding approaches ScaLeKB (Chen, Wang, and Goldberg 2016b) and RARL (Pirrò 2020) adopt a top-down approach by exploring the schema-level knowledge in KGs, i.e., entity types and their relations. They construct path patterns directly from schema-level graphs.

Recently, there is an emerging interest of applying deep neural networks (DNNs) in rule learning. Differentiable rule learners NeuralLP (Yang, Yang, and Cohen 2017) and DRUM (Sadeghian et al. 2019) have been proposed to learn rules directly using DNNs, by optimising objective functions that roughly correspond to plausible path patterns. In (Yang et al. 2014), the authors first suggest to use KG embeddings extracted from neural networks for rule learning, and RLvLR (Omran, Wang, and Wang 2018) is, and as far as know, the only rule learner that can handle large-scale KGs like DBpedia or Wikidata, by using a sampling strategy and matrix computation techniques.

Path-based and DNN-based rule learners can only learn rules that resemble path patterns, with only binary predicates. While ontological pathfinding approaches use type information, the rules learned by them do not contain unary predicates either; that is, type information is only used to confine path search and the learned rules do not contain types. While ILP-based methods do not necessarily have such restrictions, scalable systems like AMIE$^+$ adopt a language bias that only slightly extends path patterns and do not contain unary predicates either.

2.2 Link Prediction

Link prediction is a major inference task for KG completion. Intensive research has been conducted to embed entities and relations in KGs into low-dimensional latent representations, known as *embeddings*. Approaches to link prediction train various scoring functions based on embeddings to estimate the plausibility of triples. Existing approaches along this line can be roughly divided into three categories: tensor decomposition models (Yang et al. 2014; Trouillon et al. 2016), geometric translational models (Bordes et al. 2013; Sun et al. 2019) and deep neural models (Dettmers et al. 2018; Nguyen et al. 2018).

Some recent approaches also explore the type information in KGs and propose embeddings for types. TKRL (Xie et al. 2016) is based on translation model and embeds hierarchical types as projection matrices for entities. JOIE (Hao et al. 2019) adopts a more flexible approach that allows various embedding models as base embeddings and uses nonlinear transformations to capture class membership and hierarchies. HAKE (Zhang et al. 2020) uses a more complex embedding model based on polar coordinate systems to capture hierarchical types.

Compared to black-box neural models, rules are explicit knowledge and are also used for KG reasoning with human-understandable explanations for the results. And link prediction is also considered an important indicator for the quality of rules learned over KGs. With the recent advance of rule learning, rule learners are increasingly used for link prediction and demonstrate competitive accuracy and sometimes significantly better scalability over embedding-based approaches (Omran, Wang, and Wang 2018; Meilicke et al. 2019; Pirrò 2020).

3 Preliminaries

In this section, we briefly introduce some basics of knowledge graphs and rules as well as fixing some notations to be used later.

3.1 Knowledge Graphs

A *knowledge graph* (*KG*) is a directed multi-relational graph, often expressed as a set of *triples* of the form (s, p, o), i.e., $(\mathit{subject}, \mathit{relation}, \mathit{object})$, where subjects and objects are the vertices of the graph and they are connected via relations. Triples in a KG $\mathcal{K}$ can be separated into two

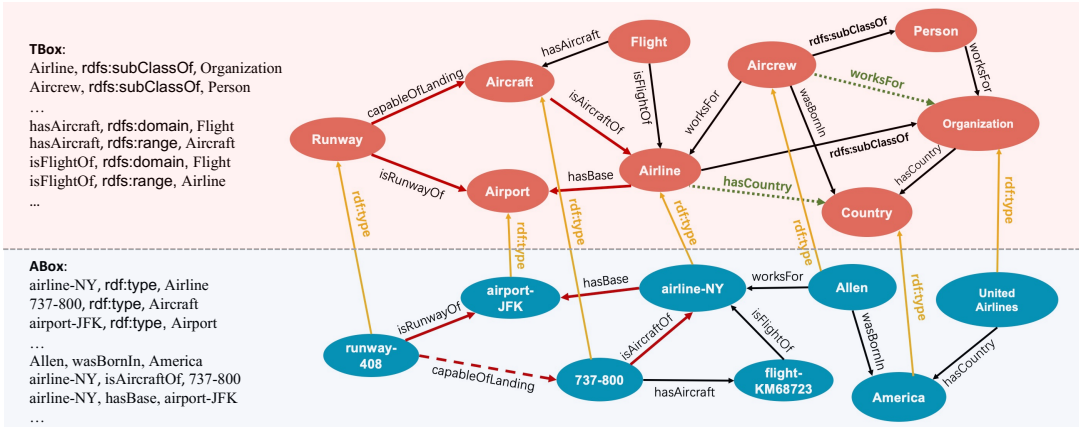

Figure 1: An example knowledge graph.

categories, i.e., $\mathcal{K} = \mathcal{T} \cup \mathcal{A}$, where $\mathcal{T}$ is the *TBox* and $\mathcal{A}$ is the *ABox*. ABox $\mathcal{A}$ describes instance level (or assertional) knowledge about *entities* and their *properties*; for instance, a triple $(\text{airline-NY}, \text{hasBase}, \text{airport-JFK})$ describes that the two entities airline-NY and airport-JFK are associated by the property hasBase. A property is a binary predicate. $\mathcal{A}$ also describes the *classes* of entities, such as a triple $(\text{airline-NY}, \text{rdf:type}, \text{Airline})$ expressing that airline-NY is a member of the class Airline. A class is a unary predicate. TBox $\mathcal{T}$, on the other hand, describes schema level (or terminological) knowledge about the relations between classes and between classes and properties. For example, triple $(\text{Airline}, \text{rdfs:subClassOf}, \text{Organization})$ says that class Airline is a subclass of Organization, and the two triples $(\text{hasBase}, \text{rdfs:domain}, \text{Airline})$ and $(\text{hasBase}, \text{rdfs:range}, \text{Airport})$ state that property hasBase has a domain Airline and a range Airport.

Formally, let sets of entities, classes, and properties in the KG $\mathcal{K}$ be respectively $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{P}$. Triples in $\mathcal{A}$ are of the forms

- (e_1, p, e_2) with $e_1, e_2 \in \mathcal{E}$ and $p \in \mathcal{P}$, and
- $(e, \text{rdf:type}, c)$ with $e \in \mathcal{E}$, $p \in \mathcal{P}$, and $c \in \mathcal{C}$.

Following the tradition in knowledge bases, we denote the two types of ABox triples as *facts* of the forms $p(e_1, e_2)$ and $c(e)$, respectively. We write $c(e) \in \mathcal{A}$ or $p(e_1, e_2) \in \mathcal{A}$ meaning the triple occurs in the ABox. Triples in $\mathcal{T}$ are of forms

- $(c_1, \text{rdfs:subClassOf}, c_2)$ with $c_1, c_2 \in \mathcal{C}$,
- $(p, \text{rdfs:domain}, c)$ with $p \in \mathcal{P}$ and $c \in \mathcal{C}$, and
- $(p, \text{rdfs:range}, c)$ with $p \in \mathcal{P}$ and $c \in \mathcal{C}$.

For simplicity, the first type of triple is denoted $c_1 \sqsubseteq c_2$. We use $\sqsubseteq^*$ to denote the transitive closure of the relation $\sqsubseteq$. Following (Hao et al. 2019; Pirrò 2020), we denote each pair of triples $(p, \text{rdfs:domain}, c_1)$ and $(p, \text{rdfs:range}, c_2)$ as a single triple (c_1, p, c_2), and they should not be confused with ABox triples.

Triples of the form $c_1 \sqsubseteq c_2$ are sometimes referred to as class (or type) hierarchy, (c_1, p, c_2) as domain and range constraints, and $c(e)$ as class (or type) membership. They together are called *type information* in the KG. In this paper, we will use the terms "class" and "type" interchangeably. For a relation p, p^- denotes its *inverse*, i.e., triple (o, p^-, s) is equivalent to (s, p, o). Let $\mathcal{P}^* = \mathcal{P} \cup \{p^- \mid p \in \mathcal{P}\}$.

3.2 Rules

Existing rule learners mostly focus on *closed-path* (*CP*) rules (Yang et al. 2014; Chen, Wang, and Goldberg 2016b; Omran, Wang, and Wang 2018; Pirrò 2020) of the form

$$p_1(x_0, x_1) \wedge p_2(x_1, x_2) \wedge \cdots \wedge p_n(x_{n-1}, x_n) \rightarrow p(x_0, x_n),$$

where $p \in \mathcal{P}$, $p_i \in \mathcal{P}^*$ $(1 \le i \le n)$ and x_j's $(0 \le j \le n)$ are variables.

For instance, a Horn rule

$$\text{hasAircraft}(x, y) \wedge \text{isFlightOf}(x, z) \rightarrow \text{isAircraftOf}(y, z)$$

says if a flight x has an aircraft y and x is a flight of an airline z then y is an aircraft of z. It corresponds to a CP rule

$$\text{hasAircraft}^-(x_0, x_1) \wedge \text{isFlightOf}(x_1, x_2) \rightarrow \text{isAircraftOf}(x_0, x_2).$$

As discussed in the introduction, it is useful to include types of variables in the rules.

$$\text{hasAircraft}^-(x_0 : \text{Aircraft}, x_1 : \text{Flight}) \wedge \text{isFlightOf}(x_1 : \text{Flight}, x_2 : \text{Airline}) \rightarrow \text{isAircraftOf}(x_0 : \text{Aircraft}, x_2 : \text{Airline}).$$

Such types can be naturally expressed as unary predicates, and hence we want to learn first-order Horn rules with unary predicates in the rule body.

A *typed rule* (or simply a *rule*) r is of the form

$$c_0(x_0) \wedge p_1(x_0, x_1) \wedge c_1(x_1) \wedge p_2(x_1, x_2) \wedge \cdots \wedge p_n(x_{n-1}, x_n) \wedge c_n(x_n) \rightarrow p(x_0, x_n). \quad (1)$$

Each c_i $(1 \le i \le n-1)$ is either a class from $\mathcal{C}$ or $\top$, where $\top$ is the class of all entities and $\top(e)$ is always valid for any entity e. A variable in the rule body does not have to be attached a type, and multiple occurrences of the same variable may have multiple types. If no specific

type is attached, it has the $\top$ type. Clearly, CP-rules are a special case of typed rules where all the c_i's are $\top$. The atom $p(x_0, x_n)$ is the *head* of r and the set of atoms $\{c_0(x_0), p_1(x_0, x_1), c_1(x_1), p_2(x_1, x_2), \cdots, p_n(x_{n-1}, x_n), c_n(x_n)\}$ is the body of r. The *length* of the rule body is n. Note that the rule heads only contain binary predicates as we learn such rules mainly for link prediction.

A key step in rule learning is to assess the plausibility of candidate rules, and most rule learners for KGs use *support*, *standard confidence*, and *head coverage* (Chen, Wang, and Goldberg 2016b; Omran, Wang, and Wang 2018), or their variants (Galárraga et al. 2015), as metrics for rule plausibility. For a rule r of the form (1), $ins_H(r)$ consists of all the pairs of entities $e_0, e_n \in \mathcal{E}$ such that when x_0 and x_n are replaced with e_0 and e_n, the fact obtained from the head of r occurs in $\mathcal{A}$. Similarly, $ins_B(r)$ consists of all the pairs of entities $e_0, e_n \in \mathcal{E}$ such that there exist $e_1, \ldots, e_{n-1} \in \mathcal{E}$, when each x_i is replaced with e_i $(0 \leq i \leq n)$, all the facts obtained from the body of r occur in $\mathcal{A}$. Then, the *support* of r is defined as $|ins_H(r) \cap ins_B(r)|$. That is, the support of r is defined as the number of entity pairs that satisfy both the head and the body of r. The *standard confidence* (*SC*) and *head coverage* (*HC*) of r are defined as follows

$$sc(r) = \frac{|ins_H(r) \cap ins_B(r)|}{|ins_B(r)|} \text{ and}$$

$$hc(r) = \frac{|ins_H(r) \cap ins_B(r)|}{|ins_H(r)|}$$

Hence, SC is the normalisation of support through the number of entity pairs that satisfy the body, while HC is the normalisation of support through the number of entity pairs that satisfy the head. The higher the values are for these metrics, the more plausible the rule is.

3.3 Rule-based Link Prediction

Link prediction is the task that given an entity $e \in \mathcal{E}$ and a property $p \in \mathcal{P}^*$, to predict entities e' such that $p(e, e')$ is plausible. Unlike embedding-based approaches that rank the possible entities e' via scoring functions, a rule-based approach tries to derive plausible facts $p(e, e')$ by applying the learned rules to the existing facts in the KG. For a rule r of the form (1), the derived facts are $p(e_0, e_n)$ for $e_0, e_n \in \mathcal{E}$ such that there exist $e_1, \ldots, e_{n-1} \in \mathcal{E}$, when each x_i is replaced with e_i $(0 \leq i \leq n)$, all the facts obtained from the body of r occur in $\mathcal{A}$.

The ranking of the derived fact is obtained from the confidences of the rules deriving it. Note that each fact α can be derived by multiple rules $r_1, \ldots, r_m$. We adopt the ranking method called Max-Aggregation (Meilicke et al. 2019), which ranks a fact based on the maximum SC of all rules deriving it. More specifically, let $\mathbf{d}_\alpha$ be a vector of the SC values of all the rules deriving α in descending order, i.e., $\mathbf{d}_\alpha = (sc(r_{k_1}), \ldots, sc(r_{k_m}))$ with $1 \leq k_1, \ldots, k_m \leq m$ and $sc(r_{k_i}) \geq sc(r_{k_{i+1}})$ for $1 \leq i < m$. Then, $\mathbf{d} < \mathbf{d}'$ if there exists some $i \geq 0$ such that $d_i < d'_i$ and $d_j = d'_j$ for all $0 \leq j < i$, where d_i (resp., d'_i) is the i-th element of $\mathbf{d}$ (resp., $\mathbf{d}'$).

4 Rule Learning

In this section, we introduce our approach for rule learning. The proposed approach combines path-based and embedding-based methods for candidate rule generation. To obtain such an effective combination, major challenges include how to use type information to narrow down the path search, and how to embed type information and utilize the embedding.

To learn the following rule for a property isAircraftOf

$$\mathsf{Aircraft}(x_0) \wedge \mathsf{hasAircraft}^-(x_0, x_1) \wedge \mathsf{Flight}(x_1) \wedge \mathsf{isFlightOf}(x_1, x_2) \wedge \mathsf{Airline}(x_2) \rightarrow \mathsf{isAircraftOf}(x_0, x_2),$$

we essentially want to extract a sequence of types and properties like (Aircraft, hasAircraft$^-$, Flight, isFlightOf, Airline). For simplicity, we refer such a sequence as a *path pattern*. In a knowledge graph, there are at least two ways for obtaining a path pattern for the target property isAircraftOf. It can be achieved by exploring a path in the TBox in Figure 1 formed by the two triples (Aircraft, hasAircraft$^-$, Flight) and (Flight, isFlightOf, Airline). Alternatively, the rule can be obtained by exploring many paths in the ABox in Figure 1 such as (737-800, hasAircraft$^-$, flight-KM68732) and (flight-KM68732, isFlightOf, airline-NY).

Given a knowledge graph $\mathcal{K}$, a *path* is a sequence of triples $(s_1, p_1, o_1), (s_2, p_2, o_2), \ldots, (s_n, p_n, o_n)$ in $\mathcal{K}$ where $o_i = s_{i+1}$ $(1 \leq i < n)$, denoted $(s_1, p_1, s_2, p_2, \ldots, p_n, o_n)$, and its *length* is n. A *TBox path* (resp., *ABox path*) is a path where $s_i, o_i \in \mathcal{C}$ $(s_i, o_i \in \mathcal{E})$ and $p_i \in \mathcal{P}^*$ $(1 \leq i \leq n)$.

Thus, we formally define *path patterns* in KGs as follows.

Definition 1. *A* path pattern *of a KG $\mathcal{K}$ is of the form $\theta = (c_0, p_1, c_1, p_2, \cdots, p_n, c_n)$, where $c_i \in \mathcal{C}$ $(0 \leq i \leq n)$ and $p_j \in \mathcal{P}^*$ $(1 \leq j \leq n)$.*

A path $\beta = (s_0, p_1, s_1, p_2, \ldots, p_n, s_n)$ in $\mathcal{K}$ is an instance *of θ if for each $0 \leq i \leq n$,*

- *if $s_i \in \mathcal{C}$, $s_i \sqsubseteq^* c_i$, and*
- *if $s_i \in \mathcal{E}$, $c(s_i) \in \mathcal{A}$ for some $c \sqsubseteq^* c_i$.*

Conversely, θ is a pattern *of β.*

Clearly, each TBox path is a path pattern, but not necessarily vice versa, as a path pattern does not necessarily consist of triples in the KG. Learning rules of the form (1) is essentially learning path patterns with significant numbers of ABox paths as instances of the form $(e_0, p_1, e_1, p_2, \ldots, p_n, e_n)$ such that $p(e_0, e_n) \in \mathcal{A}$.

In our approach, we first extract a subgraph of $\mathcal{K}$ for each p, which contains the relevant paths and type information (Section 4.1). Then, we search path patterns on both the TBox graph and ABox graph levels, while the type information is used to reduce the search space (Section 4.2). Meanwhile, we embed the entities, relations, and classes in a latent space, and use the embeddings to complement the search (Section 4.3). Finally, the generated candidate rules, which are in a much more manageable number, are assessed using the SC and HC metrics.

4.1 KG Modules

Sampling of large-scale KGs has been shown to be effective in significantly reducing the data sizes (Omran, Wang, and Wang 2018). For each predicate p that can occur in the head of rules and a maximum rule body length n, we extract a subgraph of $\mathcal{K}$ that is sufficient to generate path patterns.

Definition 2. *For a property $p \in \mathcal{P}$ and an integer $n \geq 1$, a (p, n)-*module *of $\mathcal{K}$ is a subgraph of $\mathcal{K}$ that contains all the ABox paths $\beta = (e_0, p_1, e_1, p_2, \ldots, p_n, e_n)$ with $p(e_0, e_n) \in \mathcal{A}$, and all the triples containing some classes in the patterns of β.*

To extract a (p, n)-module, our method deploys a breath-first tree search. Let $l = \lfloor \frac{n+1}{2} \rfloor$.

- Take $\mathcal{A}_0$ as the set of triples in $\mathcal{A}$ that contain p, and $\mathcal{E}_0$ is the set of entities in $\mathcal{A}_0$;
- For each $1 \leq i \leq l$, let $\mathcal{A}_i$ be the set of triples in $\mathcal{A}$ that contain some entity in $\mathcal{E}_{i-1}$, $\mathcal{E}_i$ and $\mathcal{C}_i$ be the set of respectively entities and classes in $\mathcal{A}_i$, and $\mathcal{T}_i$ be the set of triples in $\mathcal{T}$ that contain some class in $\mathcal{C}_i$.

Observation 1. *$\mathcal{T}_l \cup \mathcal{A}_l$ is a (p, n)-module of $\mathcal{K}$.*

For example, consider an ABox path of length 3, $(e_0, p_1, e_1, p_2, e_2, p_3, e_3)$, such that $p(e_0, e_3) \in \mathcal{A}$, then $e_0, e_3 \in \mathcal{E}_0$, $p_1(e_0, e_1), p_3(e_2, e_3) \in \mathcal{A}_1$, $e_1, e_2 \in \mathcal{E}_1$, and $p_2(e_1, e_2) \in \mathcal{A}_2$.

Compared with the sampling in RLvLR (Omran, Wang, and Wang 2018), our method has superior time efficiency, taking only about half the time RLvLR uses. Also, our method preserves all relevant path patterns, whereas RLvLR is not necessary the case, as its method has to limit the entities or relations in each iteration due the large sizes of sampling. In (Pirrò 2020), the author extracts a reduced ABox for each candidate rule body that can be used to assess the plausibility. We extract a module for each head property, which can be used to generate all candidate bodies.

After we obtain a (p, n)-module, we use two strategies to search for path patterns on the module to form candidate rules. One strategy is to explore paths in both the TBox and ABox in the module, and the other strategy is through embeddings over the module. We present the two strategies in the following two subsections. When the context is clear, for annotation simplicity, we will consider the module as our KG and use $\mathcal{K}$ to represent the module.

4.2 Path-based Strategy

TBox Paths For a KG $\mathcal{K}$, we define its *TBox coverage* (TC) as the ratio t/s, where t (resp., s) is the number of TBox paths (resp., all path patterns) in $\mathcal{K}$ that have some ABox paths as their instances. A KG is *fully TBox covered* if its TBox coverage is 1. When the TBox coverage is relatively high, exploring TBox paths is an effective way to generate candidate rules, as shown in (Chen, Wang, and Goldberg 2016b; Pirrò 2020).

Yet generating candidate rules directly from TBox patterns may cause important information in the class hierarchies being neglected. For example, suppose there are two triples $(\mathsf{Aircrew}, \mathsf{worksFor}, \mathsf{Airline})$ and $(\mathsf{Organization}, \mathsf{hasCountry}, \mathsf{Country})$ in the TBox, they do not form a TBox path from Aircrew to Country. Yet we know airlines are organizations and have their associated countries. With the class hierarchy $\mathsf{Airline} \sqsubseteq \mathsf{Organization}$ in the TBox, the two path patterns are both reasonable, $(\mathsf{Aircrew}, \mathsf{worksFor}, \mathsf{Airline}, \mathsf{hasCountry}, \mathsf{Country})$ and $(\mathsf{Aircrew}, \mathsf{worksFor}, \mathsf{Organization}, \mathsf{hasCountry}, \mathsf{Country})$.

Based on this observation, we first relax the condition of a TBox path to consider class hierarchies.

Definition 3. *For a path formed by triples $(s_1, p_1, o_1), \ldots, (s_n, p_n, o_n)$, suppose p_i is $\sqsubseteq$ or its inverse with some $1 \leq i \leq n$, then a result of* bridging *is obtained by replacing s_i with o_i, or vice versa, and removing the i-th triple.*

An extended TBox path *is obtained by recursively bridging a path formed by triples from $\mathcal{T}$ till all the remaining relations are from $\mathcal{P}^*$.*

In (Chen, Wang, and Goldberg 2016b), the authors replace classes in each TBox path with their subclasses. This is different from our bridging operation. For instance, suppose there is a class hierarchy $\mathsf{EuropeAirline} \sqsubseteq \mathsf{Airline}$, their approach would add a triple $(\mathsf{Aircrew}, \mathsf{worksFor}, \mathsf{EuropeAirline})$, which may not be expected.

The notions TBox coverage and being fully TBox covered can be extended by considering extended TBox paths instead of TBox paths.

We explore the extended TBox paths for path patterns that can be used for candidate rules. In particular, to learn rules of the form (1), for each pair of classes $c, c' \in \mathcal{C}$ that is directly connected by p in the TBox, we check all the extended TBox paths from c to c'. Our exploration method is combined with the KG module extraction process, as all extended TBox paths of length k that have ABox path instances can be constructed from $\mathcal{T}_l$ with $l = \lfloor \frac{k+1}{2} \rfloor$.

ABox Paths Generating candidate rules directly from extended TBox paths may have the following issues. First, many extended TBox paths may have few or no instances in the ABox, which means the support of such rules will be low. Second, for KGs with low TBox coverage, their extended TBox paths may fail to capture many plausible path patterns. We address these two issues by exploring ABox paths.

To address the first issue, we use ABox paths to filter the extended TBox paths. This can be done on KG modules, by evaluating whether the extended TBox paths have significant numbers of ABox path instances. This can efficiently eliminate irrelevant path patterns before the plausibility assessment via the SC and HC metrics. For the second issue, we use lift ABox paths to complement the path patterns extracted from the TBox. Unlike the bottom-up approach in (Meilicke et al. 2019), where individual ABox paths are lifted by substituting entities with variables, we use vector computations to efficiently evaluate large numbers of ABox paths together.

Our goal is to explore path patterns, from extended TBox paths or lifted ABox paths, that have significant ABox path instances. The ABox instances of a path pattern can be retrieved using a SPARQL query, which is rather inefficient

as one query is needed for each path pattern. We use a lightweight check for rapid evaluation as below. For each $p \in \mathcal{P}^*$, let $dom(p) = \{e \in \mathcal{E} \mid p(e, e') \in \mathcal{A}\}$ and $ran(p) = \{e \in \mathcal{E} \mid p(e', e) \in \mathcal{A}\}$. Intuitively, for a path pattern that forms the body of rule (1), there should be a significant number of ABox path instances of the form $(e_0, p_1, e_1, p_2, \ldots, p_n, e_n)$ such that $p(e_0, e_n) \in \mathcal{A}$; that is, a significant number of entities exist in the following sets

- $dom(p) \cap dom(p_1)$ (i.e., as e_0),
- $ran(p) \cap ran(p_n)$ (i.e., as e_n), and
- $ran(p_i) \cap dom(p_{i+1})$ (i.e., as e_i for $1 \leq i < n$).

This is similar to the co-occurrence measure proposed in (Omran, Wang, and Wang 2018), where embeddings of $dom(p)$ and $ran(p)$ are defined as the averages of the embeddings of the entities occurring in the corresponding positions. Yet the meaning of such averages is less clear. Unlike (Omran, Wang, and Wang 2018), we use one-hot encodings for $dom(p)$ and $ran(p)$. Suppose all the entities in $\mathcal{E}$ are indexed from 1 to $|\mathcal{E}|$. Let $\mathbf{p}^{\mathsf{dom}}$ (resp., $\mathbf{p}^{\mathsf{ran}}$) be a vector of length $|\mathcal{E}|$, such that the scalar at position i is $\frac{1}{|dom(p)|}$ (resp., $\frac{1}{|ran(p)|}$) if $e_i \in dom(p)$ (resp., $e_i \in ran(p)$), and 0 otherwise, for $1 \leq i \leq |\mathcal{E}|$.

The *path scoring function* is defined as follows.

$$f_{path}(r) = sim(\mathbf{p}_1^{\mathsf{dom}}, \mathbf{p}^{\mathsf{dom}}) + sim(\mathbf{p}_n^{\mathsf{ran}}, \mathbf{p}^{\mathsf{ran}}) + sim(\mathbf{p}_1^{\mathsf{ran}}, \mathbf{p}_2^{\mathsf{dom}}) + ... + sim(\mathbf{p}_{n-1}^{\mathsf{ran}}, \mathbf{p}_n^{\mathsf{dom}}),$$

where $sim(\cdot, \cdot)$ is defined by the Frobenius norm, i.e. $sim(\mathbf{v}_1, \mathbf{v}_2) = \mathsf{exp}(-\|\mathbf{v}_1 - \mathbf{v}_2\|)_F$.

Our one-hot encoding is a computational method to obtain statistics on entity distributions over predicates, which is simple and effective in calculating ABox support. Hence, we use the path scoring function to filter extended TBox paths and to generate other path patterns from the ABox. While the path scoring function does not include classes, classes of entities on the ABox paths can be added and validated through the SC and HC metrics.

4.3 Embedding-based Strategy

Our path exploration method can effectively generate potentials rules that are well supported by ABox facts, yet some useful rules can be overlooked in such a search, especially when the TBox coverage is not high, i.e., when many ABox paths cannot be captured by extended TBox paths. In this subsection, we present our embedding-based method that further complements the path-based methods.

Embedding models capture the graph structure of KGs through latent representations, which provide better granularity in modelling and can be efficiently manipulated. We embed each entity $e \in \mathcal{E}$ and each property or its inverse $p \in \mathcal{P}^*$ as d_A-dimensional vectors respectively $\mathbf{e}, \mathbf{p} \in \mathbb{R}^{d_A}$, where d_A is an integer, as in TransE (Bordes et al. 2013). In TransE, the embeddings are generated such that for each triple (e_1, p, e_2) in the ABox, their embeddings satisfy $\mathbf{e}_1 + \mathbf{p} \approx \mathbf{e}_2$. Intuitively, the embedding $\mathbf{p}$ of a property p is a translation from $\mathbf{e}_1$ to $\mathbf{e}_2$ for each pair of entities e_1, e_2 such that $p(e_1, e_2) \in \mathcal{A}$. For ABox paths of the form $(e_0, p_1, e_1, p_2, \ldots, p_n, e_n)$, it should satisfy $\mathbf{e}_0 + \mathbf{p}_1 + \mathbf{p}_2 + \cdots + \mathbf{p}_n \approx \mathbf{e}_n$; that is $\mathbf{p}_1 + \mathbf{p}_2 + \cdots + \mathbf{p}_n$ is a translation from $\mathbf{e}_0$ to $\mathbf{e}_n$ for each pair of entities e_0, e_n connected via an ABox paths as above.

Hence, for a CP rule of the form $p_1(x_0, x_1) \wedge p_2(x_1, x_2) \wedge \cdots \wedge p_n(x_{n-1}, x_n) \rightarrow p(x_0, x_n)$, it is expected that $\mathbf{p}_1 + \mathbf{p}_2 + \cdots + \mathbf{p}_n \approx \mathbf{p}$. We can use such an intuition to search for potential rules as in (Omran, Wang, and Wang 2018). Unlike the matrix embeddings adopted in (Omran, Wang, and Wang 2018), TransE embeddings can significantly improve learning efficiency. Yet, this approach do not take classes into consideration. In what follows, we will extend it with class embeddings that can describe class membership and class hierarchy.

We embed each class $c \in \mathcal{C}$ as a d_T-dimensional vector $\mathbf{c} \in \mathbb{R}^{d_T}$, where d_T is an integer that can be different from d_A; that is, class embeddings can be in a different latent space from that of the entity embeddings. The class embeddings should capture the following three kinds of knowledge: (i) membership of entities $c(e) \in \mathcal{A}$; (ii) class hierarchy $c_1 \sqsubseteq c_2 \in \mathcal{T}$; and (iii) domain and range constraints of the form $(c_1, p, c_2) \in \mathcal{T}$.

Our embedding approach for (i) and (ii) are inspired by (Hao et al. 2019), and we map embeddings of entities (resp., subclasses) to those of their classes (resp., superclasses) through transformations. In particular, let $f_M(\mathbf{e}) = \sigma(\mathbf{W}_M \cdot \mathbf{e} + \mathbf{b}_M)$ be a non-linear affine transformation that maps an entity embedding $\mathbf{e}$ to that of its class, where $\mathbf{W}_M \in \mathbb{R}^{d_T \times d_A}$ is a weight matrix, $\mathbf{b}_M \in \mathbb{R}^{d_T}$ is a bias vector, and $\sigma(\cdot)$ is a non-linear activation function such as tanh. Similarly, $f_H(\mathbf{c}) = \sigma(\mathbf{W}_H \cdot \mathbf{c} + \mathbf{b}_H)$ is another non-linear transformation that maps a class embedding $\mathbf{c}$ to that of its superclass, with $\mathbf{W}_H \in \mathbb{R}^{d_T \times d_T}$ and $\mathbf{b}_H \in \mathbb{R}^{d_T}$. Then, our class embeddings should satisfy

- for each $c(e) \in \mathcal{A}$, $f_M(\mathbf{e}) \approx \mathbf{c}$; and
- for each $c_1 \sqsubseteq c_2 \in \mathcal{T}$, $f_H(\mathbf{c}_1) \approx \mathbf{c}_2$.

For (iii), the embedding method in (Hao et al. 2019) treats $(c_1, p, c_2) \in \mathcal{T}$ in the same way as a triple $(e_1, p, e_2) \in \mathcal{A}$, which is not intuitive for domain and range constraints. Also, the approach in (Hao et al. 2019) assumes the properties in the TBox do not overlap with those in the ABox, which would result in the TBox coverage of the KG being 0. Indeed, what (c_1, p, c_2) conveys is the membership of entities in $dom(p)$ and $ran(p)$. Hence, our class embeddings should satisfy for each $(c_1, p, c_2) \in \mathcal{T}$:

- for each $e \in dom(p)$, $f_M(\mathbf{e}) \approx \mathbf{c_1}$; and
- for each $e \in ran(p)$, $f_M(\mathbf{e}) \approx \mathbf{c_2}$.

With the class embeddings trained together with the entity and property embeddings, the property embeddings would satisfy the domain and range constraints, and each entity occurring on an ABox path is always confined by its class membership. The *embedding scoring function* is defined as follows.

$$f_{embd}(r) = sim(\mathbf{p}_1 + \cdots + \mathbf{p}_n, \mathbf{p}),$$

where $sim(\cdot, \cdot)$ is the L2 norm of vector distance as in TransE, i.e. $sim(\mathbf{v}_1, \mathbf{v}_2) = -\|\mathbf{v}_1 - \mathbf{v}_2\|_2$.

We use the embedding scoring function to generate path patterns to complement those generated from the path-based strategy. Again, although the embedding scoring function does not include classes, type information is utilized through embeddings to search for path patterns and classes can be added to the candidate rules and validated through the SC and HC metrics.

5 Experiments

We have implemented a prototype system TyRuLe (Typed Rule Learning) and conducted experiments on rule learning and link prediction over both standard benchmarks and a real-life dataset. Our experiments are designed to validate the following claims:

1. TyRuLe can learn quality and informative typed rules, and the benefit of such typed rules over CP rules is also shown in enhancing the accuracy of link prediction.
2. TyRuLe outperforms major rule learners and several embedding models on link prediction.
3. Both our path-based and embedding-based components contribute to the typed rule learning.

For Claim 1, we first show the intuitiveness of typed rules TyRuLe learns on a real-life dataset in the aviation domain, which is extracted from Web sources and contains very specific domain knowledge. We also compare the rules learned by TyRuLe with and without type information, and demonstrate the benefit of typed rules in enhancing the accuracy of link prediction. For Claim 2, we compare TyRuLe with existing rule learners and embedding models on link prediction over two commonly used benchmarks FB15K-237 and WN18RR, and show the advantages of TyRuLe. Finally, for Claim 3, we conduct an ablation study to analyse the respective benefits of our path-based and embedding-based measures.

All the experiments are conducted on a PC with Intel Xeon Gold 5215 CPU at 2.50GHz and with 16G of RAM, running on Ubuntu 18.04.3. Our system and benchmarks can be found at https://github.com/Rainbow0625/TyRuLe.

5.1 Datasets

Our experiments use three commonly used benchmarks and one real-life KG. Statistics of the four datasets are shown in Table 1.

KG	#Entity	#Relation	#Class	#Triple
FB15K237	14541	237	79	310116
WN18RR	40943	11	-	89969
YAGO26K906	26078	34	106	390738
AirGraph	64782	40	24	360650

Table 1: Statistics of datasets.

FB15K237 (Toutanova and Chen 2015) and WN18RR (Dettmers et al. 2018) are widely used benchmarking datasets obtained from respectively Freebase and WordNet. They are considered challenging benchmarks for link prediction, and were developed to address the test leakage issues in FB15K and WN18 (Toutanova and Chen 2015). We included explicit type information in FB15K237, which is extracted from FB15K following (Xie et al. 2016). Note that the type information is extracted from the original FB15K dataset and should not be considered as external knowledge. As most of the classes contain a very small number of entities, we only kept the classes that contain at least 1000 entities and those asserted as domains and ranges of relations, which leads to a total number of 79 classes. The version of WN18RR is the same as in (Dettmers et al. 2018) and we did not add explicit type information.

Another dataset is YAGO26K906 from (Hao et al. 2019), which is obtained from YAGO with explicit type information. The TBox and ABox in YAGO26K906 have disjoint sets of properties, which means ontological pathfinding approaches cannot be applied, and we adopted it mainly for comparison with embedding model JOIE (Hao et al. 2019) for which the dataset was initially used.

Finally, since existing benchmarks are obtained from established general-purpose KGs, we add a dataset we developed about a specific domain called AirGraph. AirGraph is a KG about airlines, airports, and aircraft, obtained by extracting and integrating data from four major sources: Federal Aviation Administration (FAA)[1], ourairports.com[2], openflights.org[3], and DBpedia. The AirGraph is unavoidably more noisy and incomplete compared to the established KGs, which poses challenges to rule learners and link prediction systems. On the other hand, it has very focused domain knowledge so it is easier to compare rules learned from different systems.

5.2 Rule Learning

In the first set of experiments, we evaluate the performance of TyRuLe in rule learning. We evaluate the benefits of learning typed rules from both quantitative and qualitative aspects. In particular, we compare the number of rules learned by TyRuLe with the rule learners AMIE+ (Galárraga et al. 2015) and AnyBURL (Meilicke et al. 2019) on AirGraph. RARL (Pirrò 2020) was not compared since we could not access the system at the time of the experiment. The maximum rule body length is 3 and the minimum SC is 0.001 for all the rule learners. A time limit of 3 hours per head property is set for each rule learner, as AnyBURL is an anytime rule learner and did not stop in 3 hours.

As AnyBURL learns partial-grounded rules, i.e., CP rules with some variables substituted by entities, several such rules can be obtained from a single CP rule. For a fair comparison on rule numbers, we also record the CP rules learned by AnyBURL. On AirGraph, TyRuLe learned 941 typed rules, including 815 CP rules. In the given time, AnyBURL could learn 173396 rules (mostly partial-grounded) including 357 CP rules, and AMIE+ could learn 76 rules.

In Table 2, we list the top 3 rules learned by each system on the head property capableOfLanding with their standard

[1] https://www.faa.gov/data_research/

[2] https://ourairports.com/

[3] https://openflights.org/data.html

AMIE+	r_1:0.114	$\mathsf{capableOfLanding}(x, z_1) \wedge \mathsf{hasPhysicalClass}(z_1, z_2) \wedge \mathsf{hasGearConfig}(z_2, y) \rightarrow \mathsf{capableOfLanding}(x, y)$,
AnyBURL	r_2:0.044	$\mathsf{capableOfLanding}(x, z_1) \wedge \mathsf{isAircraftOf}(z_1, z_2) \wedge \mathsf{isAircraftOf}(y, z_2) \rightarrow \mathsf{capableOfLanding}(x, y)$,
	r_3:0.042	$\mathsf{hasSurface}(x, z_1) \wedge \mathsf{hasSurface}(z_2, z_1) \wedge \mathsf{capableOfLanding}(z_2, y) \rightarrow \mathsf{capableOfLanding}(x, y)$,
	r_4:0.038	$\mathsf{capableOfLanding}(x, z_1) \wedge \mathsf{hasWakeCategory}(z_1, z_2) \wedge \mathsf{hasGearConfig}(y, z_2) \rightarrow \mathsf{capableOfLanding}(x, y)$,
TyRuLe	r_5:0.851	$\mathsf{Runway}(x) \wedge \mathsf{capableOfLanding}(x, z_1) \wedge \mathsf{Aircraft}(z_1) \wedge \mathsf{isAircraftEventOf}(z_2, z_1) \wedge \mathsf{Event}(z_2)$ $\wedge \mathsf{isAircraftEventOf}(z_2, y) \wedge \mathsf{Aircraft}(y) \rightarrow \mathsf{capableOfLanding}(x, y)$,
	r_6:0.507	$\mathsf{capableOfLanding}(x, z_1) \wedge \mathsf{hasGearConfig}(z_1, z_2) \wedge \mathsf{hasWingtip}(y, z_2) \rightarrow \mathsf{capableOfLanding}(x, y)$,
	r_7:0.098	$\mathsf{Runway}(x) \wedge \mathsf{isRunwayOf}(x, z_1) \wedge \mathsf{Airport}(z_1) \wedge \mathsf{hasBase}(z_2, z_1) \wedge \mathsf{Airline}(z_2)$ $\wedge \mathsf{isAircraftOf}(y, z_2) \wedge \mathsf{Aircraft}(y) \rightarrow \mathsf{capableOfLanding}(x, y)$

Table 2: Rules learned by different systems.

confidence degrees. This property is interesting as it associates data from various original sources, such as aircraft information from FAA, airline data from openflights.org, and airport and runway information from ourairports.com. For this property, AMIE+ could only learn one rule with an SC of 0.114, AnyBURL learned 17 CP rules with the maximum SC of 0.044, and TyRuLe learned 25 typed rules with some SC as high as 0.851. For the convenience of reading, inverse properties are flipped around.

Both rules r_5 and r_7 are typed rules, and we observe that specifying types of variables can (while not always) lead to higher SC. Intuitively, with variable types, it can constrain the grounding of the rules to those potentially well supported by the ABox (i.e., with significant numbers of instances). The typed rules are intuitive, for example, rule r_7 says if an aircraft y belongs to an airline z_2 that has a base airport z_1 with a runway x, then y can land on the runway x. Rule r_5 says if an aircraft is involved in an event on a specific runway then the aircraft can land on the runway. Rules r_2 and r_3 explore some commonalities between aircrafts (i.e., same airline) and between runways (i.e., same surface). Finally, rules r_1, r_4 and r_6 discover some physical features of the aircraft and runway that are related to landing capability.

5.3 Link Prediction

A major quantitative indicator on the quality of learned rules is their performance in link prediction. Hence, in the second set of experiments, we evaluate the performance of TyRuLe in link prediction on various datasets. The way to apply the learned rules for link prediction is described in Section 3.3.

First, we evaluated TyRuLe on FB15K-237 and WN18RR, in comparison with rule learners AMIE+ (Galárraga et al. 2015), NeuralLP (Yang, Yang, and Cohen 2017), RLvLR (Omran, Wang, and Wang 2018), AnyBURL (Meilicke et al. 2019) and RARL (Pirrò 2020), as well as with major embedding-based models RESCAL (Nickel, Tresp, and Kriegel 2011), TransE (Bordes et al. 2013), DistMult (Yang et al. 2014), CompIEx (Trouillon et al. 2016), ConvE (Dettmers et al. 2018), RotatE (Sun et al. 2019) and HAKE (Zhang et al. 2020). We adopt the standard metrics for link prediction, namely Mean Reciprocal Rank (MRR) and the proportion of top n hits (Hits@1 and Hits@10), to represent the accuracy of prediction. And we follow the literature to filter the predictions before ranking (Bordes et al. 2013).

Table 3 shows the results. Besides the results of HAKE is from (Zhang et al. 2020), all the other embedding-based models for FB15K237 and WN18RR are obtained from (Broscheit et al. 2020). The results of AMIE+, AnyBURL and RARL are from (Pirrò 2020). And NeuralLP and RLvLR are from (Omran, Wang, and Wang 2018). The best results are highlighted in bold, and the second best ones are underlined.

From Table 3, we can see that TyRuLe outperforms all the rule learners in link prediction, especially on Hits@1, where an improvement of up to 6.8% is achieved. This shows the quality of the rules learned by TyRuLe and the benefit of type information in link prediction. The performance of TyRuLe is even comparable to the embedding methods, and outperforms all the models on two metrics for FB15K237. For WN18RR, as type information is implicit, HAKE, which employs a more complex embedding for implicit hierarchical structures, shows better performance. And our performance is still close to HAKE.

Following the comparison in Section 5.2 on AirGraph, we compared the quality of learned rule through link prediction. Table 4 shows the results. The rules learned by TyRuLe again show better accuracy for link prediction, which confirms the quality of the learned rules.

Finally, as our embedding method is inspired from JOIE (Hao et al. 2019), we compare with JOIE on YAGO26K-906. The results are shown in Table 5, where JOIE-X is the version of JOIE that uses a base embedding method X with a configuration closest to ours (regarding the embedding of class membership and hierarchies), and the results on JOIE are from (Hao et al. 2019).

From Table 5, we can see that with the combination of path-based and embedding-based strategies, our approach can effectively utilize the type information and outperforms JOIE, sometimes by a significant margin.

5.4 Ablation Study

In this set of experiments, we analyse the contributing factors of TyRuLe's outstanding performance. First, we evaluate the usefulness of type information in rule learning. More specifically, we compare the quantities and quality of the learned rules with and without types. In particular, we separate the rules learned by TyRuLe into two groups, the CP rules (with no types) and the other rules (with types, called T-rules). We also group the rules by their body lengths $n = 1, 2, 3$. Table 6 shows the numbers of learned rules per head property (#Rule) and the overall link prediction results (LP) of different groups of rules learned by TyRuLe on FB15K237.

	Model	FB15K237			WN18RR		
		MRR	Hits@1	Hits@10	MRR	Hits@1	Hits@10
Link Predictor	RESCAL	35.6	26.3	54.1	46.7	43.9	51.7
	TransE	31.3	22.1	49.7	22.8	5.3	52.0
	DistMult	34.3	25.0	53.1	45.2	41.3	53.0
	ComplEx	34.8	25.3	53.6	47.5	43.8	54.7
	ConvE	33.9	24.8	52.1	44.2	41.1	50.4
	RotatE	33.3	24.0	52.2	47.8	43.9	55.3
	HAKE	34.6	25.0	**54.2**	**49.7**	**45.2**	**58.1**
Rule Learner	AMIE+		17.4	40.9		35.8	38.8
	NeuralLP	24		36.1			
	RLvLR	24		39.3			
	AnyBURL	31	23.3	48.6	47	44.1	55.2
	RARL	32	25.1	49.1	36.13	35.1	40.9
	TyRuLe	**39.5**	**33.5**	52.0	48.3	44.8	54.1

Table 3: Link prediction on FB15K237 and WN18RR: TyRuLe outperforms all rule learner and competitive to other models for link prediction.

Model	MRR	Hits@1	Hits@3	Hits@10
AMIE+	12.3	11.9	12.7	12.9
AnyBURL	31.3	26.2	34.3	43.1
TyRuLe	**35.3**	**30.6**	**37.5**	**45.1**

Table 4: Link prediction on AirGraph.

Model	MRR	Hits@1	Hits@10
JOIE-TransE	30.6	18.6	51.7
JOIE-DistMult	29.6	19.4	45.5
JOIE-HolE	32.7	22.4	**52.4**
TyRuLe	**42.8**	**37.7**	**52.4**

Table 5: Link prediction on YAGO26K-906.

#Rule	$n=1$	$n=2$	$n=3$	Total
CP rules	2.9	29.0	127.3	159.2
T-rules	1.7	23.5	116.5	141.7
Together	4.6	52.5	243.8	300.9
LP	MRR	Hits@1	Hits@3	Hits@10
CP rules	33.7	26.7	35.5	47.9
T-rules	38.7	32.8	40.1	50.8
Together	39.5	33.5	40.9	52.0

Table 6: Link prediction with and without types.

	Path		Emb.		Overlap	
Property	CP	TR	CP	TR	CP	TR
modeTransportation	201	101	168	73	143	65
disciplineOrSubject	21	6	18	2	16	2
parentGenre	45	36	80	43	27	21
eventLocation	122	113	141	130	115	109
directorFilm	157	153	237	208	144	130

Table 7: Path-based vs embedding-based strategies.

We can see that while the numbers of T-rules are smaller than those of CP rules, which may be due to the limited type information, the T-rules achieve higher accuracy in link prediction. And the link prediction accuracy is further enhanced when the two groups rules are combined. This shows the benefits of type information in both enriching and enhancing the quality of learned rules.

Then, we analyse the contributions of our path-based and embedding-based strategies in rule learning. We separate the rules respectively learned through the two strategies on FB15K237. Table 7 shows the numbers of rules learned for some head properties by the two different strategies, where we separately record the numbers of CP rules (CP) and T-rules (TR).

6 Conclusion

In this paper, we have developed a novel approach to learn typed rules over KGs, by exploiting the rich type information in KGs. Instead of using such type information as hard constraints for rule search, we adopted a combined approach using both path-based and embedding-based strategies for the search of candidate rules. Type information from the KG is used both in the path exploration and embedding generation, and the selected paths and generated embeddings in turn are used to generate typed rules. We have evaluated our prototype system in both rule learning and link prediction tasks, through quantitative and qualitative analysis. TyRuLe showed superior performance over existing rule learners in link prediction, and is competitive compared to embedding models, which shows the quality of our learned rules.

In our typed rules, type atoms can only appear in the body part of the rule. Next, we plan to extend our approach to learn more expressive rules with type in the rule heads. Furthermore, we will also explore the application of learned rules in reasoning tasks over KGs, such as question answering, to provide explainability of predictions.

Acknowledgments

The authors would like to thank three anonymous referees for their helpful comments. This work was partially supported by National Natural Science Foundation of China (NSFC) (61976153).

References

Ahmadi, N.; Huynh, V.-P.; Meduri, V.; Ortona, S.; and Papotti, P. 2020. Mining expressive rules in knowledge graphs. *Journal of Data and Information Quality* 12(2):1–27.

Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; and Taylor, J. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proc. of SIGMOD-08*, 1247–1250.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. *Proc. of NIPS-13* 26:2787–2795.

Broscheit, S.; Ruffinelli, D.; Kochsiek, A.; Betz, P.; and Gemulla, R. 2020. LibKGE - A knowledge graph embedding library for reproducible research. In *Proc. of EMNLP-20 (Demos)*, 165–174.

Chen, Y.; Goldberg, S.; Wang, D. Z.; and Johri, S. S. 2016a. Ontological pathfinding. In *Proc. of COMAD-16*, 835–846.

Chen, Y.; Wang, D. Z.; and Goldberg, S. 2016b. Scalekb: scalable learning and inference over large knowledge bases. *The VLDB Journal* 25(6):893–918.

Cropper, A.; Dumančić, S.; Evans, R.; and Muggleton, S. H. 2022. Inductive logic programming at 30. *Mach. Learn.* 111(1):147–172.

Dettmers, T.; Minervini, P.; Stenetorp, P.; and Riedel, S. 2018. Convolutional 2d knowledge graph embeddings. In *Proc. of AAAI-18*, 1811–1818.

Gad-Elrab, M. H.; Stepanova, D.; Urbani, J.; and Weikum, G. 2016. Exception-enriched rule learning from knowledge graphs. In *Proc. of ISWC-16*, 234–251.

Galárraga, L.; Teflioudi, C.; Hose, K.; and Suchanek, F. M. 2015. Fast rule mining in ontological knowledge bases with amie+. *The VLDB Journal* 24(6):707–730.

Hao, J.; Chen, M.; Yu, W.; Sun, Y.; and Wang, W. 2019. Universal representation learning of knowledge bases by jointly embedding instances and ontological concepts. In *Proc. of KDD-19*, 1709–1719.

Ji, S.; Pan, S.; Cambria, E.; Marttinen, P.; and Yu, P. S. 2020. A survey on knowledge graphs: Representation, acquisition and applications. *CoRR* abs/2002.00388.

Lisi, F. A., and Weikum, G. 2017. Towards nonmonotonic relational learning from knowledge graphs. In *Proc. of ILP-17*, 94.

Meilicke, C.; Chekol, M. W.; Ruffinelli, D.; and Stuckenschmidt, H. 2019. Anytime bottom-up rule learning for knowledge graph completion. In *Proc. of IJCAI-19*, 3137–3143.

Miller, G. A. 1995. Wordnet: a lexical database for english. *Communications of the ACM* 38(11):39–41.

Muggleton, S. 1990. Inductive logic programming. In Arikawa, S.; Goto, S.; Ohsuga, S.; and Yokomori, T., eds., *Proc.of ALT-90*, 42–62.

Muggleton, S. 1991. Inductive logic programming. *New Gener. Comput.* 8(4):295–318.

Nguyen, T. D.; Nguyen, D. Q.; Phung, D. Q.; et al. 2018. A novel embedding model for knowledge base completion based on convolutional neural network. In *Proc. of NAACL-HLT-18*, 327–333.

Nickel, M.; Tresp, V.; and Kriegel, H.-P. 2011. A three-way model for collective learning on multi-relational data. In *Proc. of ICML-28*, 809–816.

Omran, P. G.; Wang, K.; and Wang, Z. 2018. Scalable rule learning via learning representation. In *Proc. of IJCAI-18*, 2149–2155.

Pellissier Tanon, T.; Stepanova, D.; Razniewski, S.; Mirza, P.; and Weikum, G. 2017. Completeness-aware rule learning from knowledge graphs. In *Proc. of ISWC-17*, 507–525.

Pirrò, G. 2020. Relatedness and tbox-driven rule learning in large knowledge bases. In *Proc. of AAAI-20*, 2975–2982.

Sadeghian, A.; Armandpour, M.; Ding, P.; and Wang, D. Z. 2019. Drum: End-to-end differentiable rule mining on knowledge graphs. *Proc. of NIPS-19* 32:15321–15331.

Suchanek, F. M.; Kasneci, G.; and Weikum, G. 2007. Yago: a core of semantic knowledge. In *Proc. of WWW-07*, 697–706.

Sun, Z.; Deng, Z.; Nie, J.; and Tang, J. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. In *Proc. of ICLR-19(Poster)*.

Toutanova, K., and Chen, D. 2015. Observed versus latent features for knowledge base and text inference. In *Proc. of CVSC-15*, 57–66.

Trouillon, T.; Welbl, J.; Riedel, S.; Gaussier, É.; and Bouchard, G. 2016. Complex embeddings for simple link prediction. In *Proc. of ICML-16*, 2071–2080.

Xie, R.; Liu, Z.; Sun, M.; et al. 2016. Representation learning of knowledge graphs with hierarchical types. In *Proc. of IJCAI-16*, 2965–2971.

Yang, B.; Yih, W.-t.; He, X.; Gao, J.; and Deng, L. 2014. Embedding entities and relations for learning and inference in knowledge bases. In *Proc. of ICLR-14*.

Yang, F.; Yang, Z.; and Cohen, W. W. 2017. Differentiable learning of logical rules for knowledge base reasoning. *Proc. of NIPS-2017* 30:2319–2328.

Zeng, Q.; Patel, J. M.; and Page, D. 2014. Quickfoil: Scalable inductive logic programming. *Proc. of the VLDB Endowment-14* 8(3):197–208.

Zhang, Z.; Cai, J.; Zhang, Y.; and Wang, J. 2020. Learning hierarchy-aware knowledge graph embeddings for link prediction. In *Proc. of AAAI-20*, 3065–3072.