DegreEmbed: incorporating entity embedding into logic rule learning for knowledge graph reasoning

Haotian Li [a], Hongri Liu[a], Yao Wang[a], Guodong Xin[a] and Yuliang Wei [a,*]

[a] *School of Computer Science and Technology, Harbin Institute of Technology at Weihai, China*
E-mails: lcyxlihaotian@126.com, wei.yl@hit.edu.cn

Abstract. Knowledge graphs (KGs), as structured representations of real world facts, are intelligent databases incorporating human knowledge that can help machine imitate the way of human problem solving. However, KGs are usually huge and there are inevitably missing facts in KGs, thus undermining applications such as question answering and recommender systems that are based on knowledge graph reasoning. Link prediction for knowledge graphs is the task aiming to complete missing facts by reasoning based on the existing knowledge. Two main streams of research are widely studied: one learns low-dimensional embeddings for entities and relations that can explore latent patterns, and the other gains good interpretability by mining logical rules. Unfortunately, previous studies rarely pay attention to heterogeneous KGs. In this paper, we propose DegreEmbed, a model that combines embedding-based learning and logic rule mining for inferring on KGs. Specifically, we study the problem of predicting missing links in heterogeneous KGs that involve entities and relations of various types from the perspective of the degree of nodes. Experimentally, we demonstrate that our DegreEmbed model outperforms the state-of-the-art methods on real world datasets and the rules mined by our model are of high quality and interpretability.

Keywords: Knowledge graph reasoning, Link prediction, Logic rule mining, Degree embedding, Interpretability of model

1. Introduction

Recent years have witnessed the growing attraction of knowledge graphs in a variety of applications, such as dialogue systems [1, 2], search engines [3] and domain-specific softwares [4, 5]. Capable of incorporating large-scale human knowledge, KGs provide graph-structured representation of data that can be comprehended and examined by humans. Knowledge in KGs is stored in triple form (e_s, r, e_o), with e_s and e_o denoting subject and object entities and r a binary relation (*a.k.a.* predicate). For example, the fact that Mike is the nephew of Pete can be formed as $(\texttt{Mike}, \texttt{nephewOf}, \texttt{Pete})$. However, information incompleteness can be seen in most modern KGs, that is, missing links in the graph, *e.g.*, the work of [6, 7] shows that there are more than 66% of the person entities missing a birthplace in two open KGs Freebase [8] and DBpedia [9].

Predicting missing triples based on the existing facts is usually called *link prediction* as a subtask of knowledge graph completion (KGC) [10], and numerous models have been developed for solving such problems. One prominent direction in this line of research is *representation learning* methods that learn distributed embedding vectors for entities and relations, such as TransE [11] and ComplEx [12]. In this work, they are referred to as embedding-based methods. This kind of models are capable of capturing latent knowledge through low-dimensional vectors, *e.g.* we can classify male and female entities in a family

[*] Corresponding author.

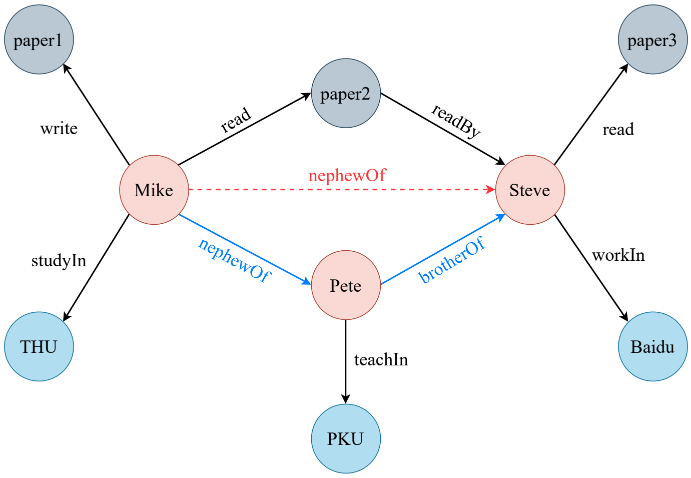


Fig. 1. An example of KG containing heterogeneous entities and relations: paper, person and institution. Entities in different colors mark their type. The existing links in the KG are presented as solid black lines, the missing one as dashed lines in red and the proper rule for inferring the link as blue lines.

KG by clustering their points at the semantic space. In spite of achieving high performance, these models suffer from non-transparency and can poorly be understood by humans, which is a common issue for most deep learning models. In addition, most embedding-based methods work in a transductive setting, where they require the entities in training and test data to overlap, hindering the way to generalize in some real-world situations.

Another popular approach is rule mining that discovers logical rules through mining co-occurrences of frequent patterns in KGs [13, 14]. This paper studies the problem of learning first-order logical Horn clauses for knowledge graph reasoning (KGR). As illustrated in Fig. 1, there is a missing link (*i.e.* `nephewOf`) between the subject `Mike` and the object `Steve`, but we can complete the fact through a logic rule `nephewOf(Mike, Pete)` ∧ `brotherOf(Pete, Steve)` ⇒ `nephewOf(Mike, Steve)`, meaning that if Mike is the nephew of Pete and Steve has a brother Pete, then we can infer that Mike is the nephew of Steve. Reasoning on KGs through Horn clauses has been previously studied in the area of *Inductive Logic Programming* [15]. One representative method, Neural LP [16], is the first fully differentiable neural system that successfully combines learning discrete rule structures as well as confidence scores in continuous space. Although learning logical rules equips a model with strong interpretability and the ability to generalize to similar tasks [17, 18], these methods often focus only on the relations of which the rules are made up, while the intrinsic properties of the involved entities are not considered. For example, in the KG shown in Fig. 1, it is definitely wrong to infer by a rule containing a female-type relation path like `sisterOf` starting from Mike, because Mike is the nephew of Pete, which indirectly tells us he is a male. This sort of deficiency is more severe in heterogeneous KGs where there are entities and relations of different types mixing up. In these KGs, there might be multiple rules of no use, which decreases the performance and interpretability of ILP models.

In this paper, in order to bridge the gap between the two lines of research mentioned above, we propose DegreEmbed, a model of logic rule learning that integrates the inner attributes of entities by embedding nodes in the graph from the perspective of their degrees. DegreEmbed is not only interpretable to humans, but also able to mine relational properties of entities. We also evaluate our model on several knowledge graph datasets, and show that we are able to learn more accurately, and meanwhile, gain strong interpretability of mined rules.

Our main contributions are summarized below:

- We propose an original model based on logic rule learning to predict missing links in heterogeneous KGs. Specifically, a new technique for encoding entities, called degree embedding, is designed to capture hidden features through the relation type of edges incident to a node.
- Comparative experiments on knowledge graph completion task with five benchmark datasets prove that our DegreEmbed model outperforms

baseline models. Besides, under the evaluation of a metric called Saturation, we show that our method is capable of mining meaningful logic rules from knowledge graphs.

- Visualizing learned entity embeddings, we demonstrate that clear features of entities can be obtained by our model, thus benefitting the prediction in heterogeneous settings. Moreover, we prove the necessity of each component of our model using ablation study.

This paper is structured as follows. We briefly introduce our related work and review preliminary definitions of knowledge graphs respectively in Section 2 and Section 3. Section 4 introduces our proposed DegreEmbed model based on logic rule learning for link prediction in heterogeneous KGs. We present the experimental results in Section 5 and conclude our work by pointing out the future direction.

2. Related work

Our work is first related to previous efforts on relational data mining, based on which, a large body of deep rule induction models have been developed for link prediction. Since our approach achieves a combination of logic rule learning and knowledge graph embedding, we conclude related work in this topic as well.

Relational data mining. The problem of learning relational rules has been traditionally addressed in the field of *inductive logic programming* (ILP) [15]. These methods often view the task of completing a missing triple as a query $q(h,t)$ where they learn a probability as confidence score for each rule between the query entity and answer entity. Among these studies, Path-Ranking Algorithm (PRA) [19] investigated the framework of random walk inference, where they select a relational path under a set of constraints and perform maximum-likelihood classification. An RNN model was developed by [20] to compose the semantics of relations for arbitrary-length reasoning. Chain-Reasoning proposed by [21], enabling multi-hop reasoning through a neural attention mechanism, reveals logical rules across all relations and entities. Although ILP models are capable of mining interpretable rules for humans, these models typically take both positive and negative examples for training and suffer from a potentially large version space, which is a critical shortage since most modern KGs are huge and contain only positive instances.

Neural logic programming. In recent years, models borrowing the idea of logic rule learning in a deep manner have emerged as successful approaches for link prediction task. Extending the idea of TensorLog that tackles the problem of rule-based logic reasoning through sparse matrix multiplication, Neural LP [16] is the first end-to-end differentiable approach to simultaneously learn continuous parameters and discrete structure of rules. Some recent methods [22–24] have improved the framework done by Neural LP [16] in different manners. DRUM [22] introduces tensor approximation for optimization and reformulate Neural LP to support rules of varied lengths. Neural-Num-LP [23] extends Neural LP to learn numerical relations like *age* and *weight* with dynamic programming and cumulative sum operation. NLIL [24] proposes a multi-hop reasoning framework for general ILP problem through a divide-and-conquer strategy as well as decomposing the search space into three subspaces. However, the existing methods ignore the effects caused by the entities while reasoning over a specific relational path, thus witness a more obvious failure where heterogeneous entities and relations are involved in the KGs.

Representation learning. Capturing their semantic information by learning low-dimensional embeddings of entities and relations, also known as *knowledge graph embedding*, is a vital research issue in KGC, and we term those models as embedding-based models. Newly proposed methods, including RotatE [25], ConvE [26] and TuckER [27], predict missing links by learning embedding vectors from various perspectives of the problem. Specifically, the work of RotatE [25] focuses on inferring patterns such as symmetry and inversion, where they proposed a rotational model that rotates the relation from the subject to the object in the complex space as $e_o = e_s \circ r$ where the $\circ$ denotes the element-wise Hadamard product. ConvE introduces a highly parameter efficient model, which uses 2D convolution over embeddings and multiple layers of nonlinear features to express semantic information. TuckER, inspired by the Tucker decomposition [28] that factorizes a tensor into a core tensor along with a set of matrices, is a linear model for link prediction that has good expressive power. Unfortunately, the biggest problem is that these sort of methods can hardly be comprehended by humans, but we relate to these methods for their ability to capture latent information of entities and relations through embedding.

We also notice that there are methods trying to establish a connection between learning logical rules and learning embedding vectors [29, 30], where they explore new triplets from the existing ones in the KG using pre-defined logical rules to deal with the sparsity problem, which differs from our goal to learn in heterogeneous graphs.

3. Preliminaries

In this section, we introduce background concepts of logic rule learning for knowledge graph reasoning as well as the definition of topological structure of KGs.

3.1. Knowledge graph reasoning

Knowledge graph can be modeled as a collection of factual triples $\mathcal{G} = \{(e_s, r, e_o) \mid e_s, e_o \in \mathcal{E}, r \in \mathcal{R}\}$, with $\mathcal{E}, \mathcal{P}$ representing the set of entities and binary relations (*a.k.a.* predicates) respectively in the knowledge graph, and (e_s, r, e_o) the triple $(\texttt{subject}, \texttt{relation}, \texttt{object})$ in form of $e_s \xrightarrow{r} e_o$. During reasoning over KGs, each triple is usually presented in the form $\texttt{r}(e_s, e_o)$. The subgraph regarding a specific relation $\texttt{r}_i$ is described as a subset of $\mathcal{G}$ containing all triples with $\texttt{r}_i$ being the connection between the subject and object: $\mathcal{G}(\texttt{r}_i) = \{(e_s, r, e_o) \mid e_s, e_o \in \mathcal{E}, \texttt{r}_i \in \mathcal{R}, r = \texttt{r}_i\}$.

Logic rule reasoning. We perform reasoning on KGs by learning a *confidence* score $\alpha \in [0, 1]$ for a first-order logic *rule* in the form

$$\texttt{r}_1(x, z_1) \wedge \cdots \wedge \texttt{r}_l(z_{l-1}, y) \Rightarrow q(x, y) \ : \ \alpha, \quad (1)$$

$\mathbf{r(x, y)} \Rightarrow q(x, y)$ for short, with $r_1, \ldots, r_l, q \in \mathcal{R}$, $z_1, \ldots, z_{l-1} \in \mathcal{E}$, where $\mathbf{r} = \wedge_i r_i$, is called a *rule pattern*. For example, the rule $\texttt{brotherOf}(x, z) \wedge \texttt{fatherOf}(z, y) \Rightarrow \texttt{uncleOf}(x, y)$ intuitively states that if x is the brother of z and z is the father of y, then we can conclude that x is the uncle of y. All rule patterns of length l $(l \geqslant 2)$ can be formally defined as a set of relational tuples $\mathcal{H}^l = \{(r_1, r_2, \ldots, r_l) \mid r_i \in \mathcal{R}, 1 \leqslant i \leqslant l\} = \mathcal{R}^l$, and the set of patterns no longer than L is denoted as $\mathbb{H}^L = \bigcup_{l=2}^{L} \mathcal{H}^l$. A *rule path* $\mathbf{p}$ is an instance of a certain pattern $\mathbf{r}$ via different sequences of entities, which can be denoted as $\mathbf{p} \triangleright \mathbf{r}$, *e.g.*, $(r_a(x, z_1), r_b(z_1, y))$ and $(r_a(x, z_2), r_b(z_2, y))$ are different paths of the same pattern.

The link prediction task here is considered to contain a variety of queries, each of which is composed of a $\texttt{query}$ body $q \in \mathcal{R}$, an entity $\texttt{head}$ h which the query is about, and an entity $\texttt{tail}$ t that is the answer to the query such that $(h, q, t) \in \mathcal{G}$. Finally we want to find the most possible logic rules $h \xrightarrow{r_1} \cdots \xrightarrow{r_l} t$ to predict the link q. Thus, given maximum length L, we assign a single *confidence* score (*i.e.* probability) to a set of rule paths $\mathbf{p}$'s adhering to the same pattern $\mathbf{r}$ that connects h and t:

$$\{\mathbf{p}_i(h, t) \Rightarrow q(h, t) \mid \mathbf{p}_i \triangleright \mathbf{r}, \mathbf{r} \in \mathbb{H}^L\} \ : \ \alpha \qquad (2)$$

During inference, given an entity h, the unified score of the answer t can be computed by adding up the confidence scores of all rule paths that infer $q(h, t)$, and the model will produce a ranked list of entities where higher the score implies higher the ranking.

3.2. Graph structure

Definition 1 (Directed Labeled Multigraph)**.** *A* ***directed labeled multigraph*** *G is a tuple $G = (V, E)$, where V denotes the set of vertices, and $E \subseteq V \times V$ is a multiset of directed, labeled vertex pairs (i.e. edges) in the graph G.*

Because of its graph structure, a knowledge graph can be regarded as a directed labeled multigraph [31]. In this paper, "graph" is used to refer to "directed labeled multigraph" for the sake of simplicity. $G(\texttt{r}) = (V(\texttt{r}), E(\texttt{r}))$ is the corresponding topological structure of $\mathcal{G}(\texttt{r})$. $m = |V|$ and $n = |E|$ stand for the **number of vertices** and **number of edges** respectively for a graph G. Particularly in a KG, $|\mathcal{E}| = m$ and the total number of triplets (e_s, r, e_o) equals the number of edges, *i.e.* $|\mathcal{G}| = n$.

Formally, in a graph $G = (V, E)$, the **degree** of a vertex $v \in V$ is the number of edges incident to it. When it comes to directed graphs, **in-degree** and **out-degree** of a vertex v is usually distinguished, which are defined as

$$deg^+(v) = |\{(u, v) \mid u \in V, (u, v) \in E\}| \qquad (3)$$

$$deg^-(v) = |\{(v, u) \mid u \in V, (v, u) \in E\}| \qquad (4)$$

But in this paper, we use "degree" to represent the edges incident to a specific node v for conciseness.

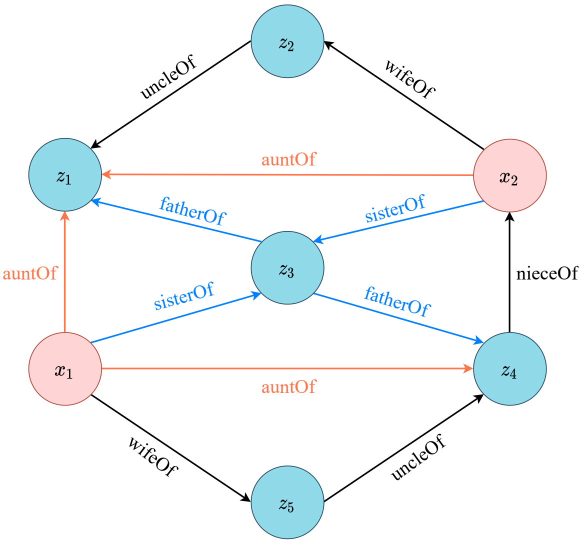


Fig. 2. A KG example of family members and the relations between them.

4. Methodology

Capable of simultaneously learning representations and logical rules, Neural LP [16] is the first differentiable neural system for knowledge graph reasoning that combines structure learning and parameter learning. Our work follows the work of Neural LP and extensive studies based on it to consider the problem of reasoning in heterogeneous KGs, from the view of mining intrinsic properties of the entities in KGs.

4.1. *Neural LP for logic reasoning*

4.1.1. *TensorLog*

The work of TensorLog [32, 33] successfully simulates the reasoning process using first-order logic rules by performing sparse matrix multiplication, based on which, Neural LP [16] proposed a fully differentiable reasoning system. In the following, we will first introduce the TensorLog operator. In a KG involving a set of entities $\mathcal{E}$ and a set of relations $\mathcal{R}$, factual triplets *w.r.t.* the relation r_k are restored in a binary matrix $\mathrm{M}_{r_k} \in \{0,1\}^{|\mathcal{E}|\times|\mathcal{E}|}$. M_{r_k}, an adjacency matrix, is called a TensorLog operator meaning that (e_i, r_k, e_j) is in the KG if and only if the (i,j)-th entry of M_{r_k} is 1. Let $\mathrm{v}_{e_i} \in \{0,1\}^{|\mathcal{E}|}$ be the one-hot encoded vector of entity e_i, then $s^\top = \mathrm{v}_{e_i}^\top \mathrm{M}_{r_1}\mathrm{M}_{r_2}\mathrm{M}_{r_3}$ is the *path features vector* [24], where the j-th entry counts the number of unique rule paths following the pattern $r_1 \wedge r_2 \wedge r_3$ from e_i to e_j [34].

For example, every KG entity $e \in \mathcal{E}$ in Fig. 2 is encoded into a 0-1 vector of length $|\mathcal{E}| = 7$. For every relation $r \in \mathcal{R}$ and every pair of entities $e_i, e_j \in \mathcal{E}$, the TensorLog operator relevant to r is define as a sparse matrix M_r with its (i,j)-th element being 1 if $(e_i, r, e_j) \in \mathcal{G}$. Considering the KG in Fig. 2, for the relation $r = \texttt{auntOf}$ we have

$$\mathrm{M}_r = \begin{array}{ccccccccc} & x_1 & x_2 & z_1 & z_2 & z_3 & z_4 & z_5 & \\ \left[\right. & 0 & 0 & {\color{red}1} & 0 & 0 & {\color{red}1} & 0 & \left.\right] x_1 \\ & 0 & 0 & {\color{red}1} & 0 & 0 & 0 & 0 & x_2 \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & z_1 \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & z_2 \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & z_3 \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & z_4 \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & z_5 \end{array}$$

And the rule $\texttt{sisterOf}(\mathrm{X}, \mathrm{Z}) \wedge \texttt{fatherOf}(\mathrm{Z}, \mathrm{Y}) \Rightarrow \texttt{auntOf}(\mathrm{X}, \mathrm{Y})$ can be simulated by performing the following sparse matrix multiplication:

$$\mathrm{M}_{\mathrm{r}'} = \mathrm{M}_{\mathrm{sisterOf}}\,\mathrm{M}_{\mathrm{fatherOf}} =$$

$$\begin{array}{cccccccc} 0 & 0 & {\color{red}1} & 0 & 0 & {\color{red}1} & 0 & x_1 \\ 0 & 0 & {\color{red}1} & 0 & 0 & {\color{red}1} & 0 & x_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & z_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & z_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & z_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & z_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & z_5 \\ x_1 & x_2 & z_1 & z_2 & z_3 & z_4 & z_5 & \end{array}$$

By setting $\mathrm{v}_{x_1} = [1,0,0,0,0,0,0]^\top$ as the one-hot vector of x_1 and multiplying by $\mathrm{v}_{x_1}^\top$ on the left, we obtain $s^\top = \mathrm{v}_{x_1}^\top \cdot \mathrm{M}_{\mathrm{r}'} = [0,0,1,0,0,1,0]$. The resultant $s^\top$ selects the row in $\mathrm{M}_{\mathrm{r}'}$ actually identified by x_1. By operating right-hand side multiplication with v_{z_1}, we get the number of unique paths following the pattern $\texttt{sisterOf} \wedge \texttt{fatherOf}$ from x_1 to z_1: $s^\top \cdot \mathrm{v}_{z_1} = 1$.

4.1.2. *Neural LP*

Neural LP [16] inherits the idea of TensorLog. Given a query $q(h,t)$, after L steps of reasoning, the score of the query induced through rule pattern $\mathbf{r}_s$ of length L is computed as

$$\mathrm{score}(t \mid q, h, \mathbf{r}_s) = \mathrm{v}_h^\top \prod_{l=1}^{L} \mathrm{M}^l \cdot \mathrm{v}_t, \tag{5}$$

where M^l is the adjacency matrix of the relation used at the l-th hop.

The operators above are used to learn for query q by calculating the weighted sum of all possible patterns:

$$\sum_s \alpha_s \prod_{k\in\beta_s} \mathrm{M}_{r_k}, \tag{6}$$

where s indexes over all potential patterns with maximum length of L, α_s is the confidence score associated with the rule $\mathbf{r}_s$ and β_s is the ordered list of relations appearing in $\mathbf{r}_s$.

To summarize, we update the score function in Eq. (5) by finding an appropriate α in

$$\varphi(t \mid q, h) = \mathrm{v}_h^\top \sum_s \alpha_s \cdot \left(\prod_{k\in\beta_s} \mathrm{M}_{r_k} \cdot \mathrm{v}_t \right), \tag{7}$$

and the optimization objective is

$$\max_{\alpha_s} \sum_{(h,q,t)\in\mathcal{G}} \varphi(t \mid q, h), \tag{8}$$

where α_s is to be learned.

Whereas the searching space of learnable parameters is exponentially large, *i.e.* $O(|\mathcal{R}|^L)$, direct optimization of Eq. (8) may fall in the dilemma of over-parameterization. Besides, it is difficult to apply gradient-based optimization. This is because each variable α_s is bound with a specific rule pattern, and it is obviously a discrete work to enumerate rules. To overcome these defects, the parameter of rule $\mathbf{r}_s$ can be reformulated by distributing the confidence to its inclusive relations at each hop, resulting in a differentiable score function:

$$\phi_L(t \mid q, h) = \left(\mathrm{v}_h^\top \prod_{l=1}^{L} \sum_{k=0}^{|\mathcal{R}|} a_k^l \mathrm{M}_{r_k} \right) \cdot \mathrm{v}_t, \tag{9}$$

where L is a hyperparameter denoting the maximum length of patterns and $|\mathcal{R}|$ is the number of relations in KG. M_{r_0} is an identity matrix I that enables the model to include all possible rule patterns of length L or smaller [22].

To perform training and prediction over the Neural LP framework, we should first construct a KG from a large subset of all triplets. Then we remove the edge (h, t) from the graph when facing the query (h, q, t), so that the score of t can get rid of the influence imposed from the head entity h directly through the edge (h, t) for the correctness of reasoning.

4.2. Our DegreEmbed model

In this section, we propose our *DegreEmbed* model based on Neural LP [16] as a combination of models relying on knowledge graph embedding and ILP models where the potential properties of individual entities are considered through a technique we call degree embedding. We discover that the attributes of nodes in a KG can make a difference via observation on their degrees. In Fig. 1, we notice that Mike is a male because he is a nephew of someone, hence it is incorrect indeed to reason by a rule containing a female-type relation starting from Mike. Also, the in-degree (*i.e.* `studyIn`) of entity THU proves its identity as a university. Besides, as illustrated in Section 4.1.1, all knowledge of a KG is stored in the relational matrices, which is our aim to reconstruct for harboring type information of entities. For a query $q(h, t)$, the final score is a scalar obtained by Eq. (9), where the *path feature vector* is $s^\top = \mathrm{v}_h^\top \prod_{l=1}^{L} \sum_{k=0}^{|\mathcal{R}|} a_k^l \mathrm{M}_{r_k}$, and v_t selects the t-th element of $s^\top$ through matrix multiplication. In fact, the vector $s^\top \in \mathbb{R}^{|\mathcal{E}|}$ is a row of matrix $\prod_{l=1}^{L} \sum_{k=0}^{|\mathcal{R}|} a_k^l \mathrm{M}_{r_k}$, each value of which is the "influence" passed from head entity h to the regarding entity. As a result, we can consider the attributes of the entity e_i by changing the i-th row of adjacency matrices from the perspective of the type of degrees of e_i.

For any entity $e \in \mathcal{E}$, we collect the ones of unique types among all of its in and out degrees separately to form a d-dimensional *degree feature vector*, where d is the number of unique degrees. Then we project the vector onto a semantic space by looking up in a row-vector embedding matrix $\mathbf{E}^{|\mathcal{R}|\times m}$, and the result is number of d vectors arranged in a matrix $\mathbf{M} \in \mathbb{R}^{d\times m}$, where m is the embedding dimension. The embedding vectors are input into a bidirectional LSTM [35] at different time steps. Finally, we perform attention operation on the hidden state of BiLSTM at the last time step to get the $|\mathcal{R}|$-dimensional attention vector of e for $1 \leqslant i \leqslant d$:

$$\mathbf{h}_i, \mathbf{h}'_{d-i+1} = \mathrm{BiLSTM}(\mathbf{h}_{i-1}, \mathbf{h}'_{d-i}, \mathbf{M}), \tag{10}$$

where $\mathbf{h}$ and $\mathbf{h}'$ are the hidden states of the forward and backward path LSTMs, with the subscripts denoting their time step, and H, **the actual embedding vector** of entity e, is obtained by concatenating $\mathbf{h}_d$ and $\mathbf{h}'_1$.

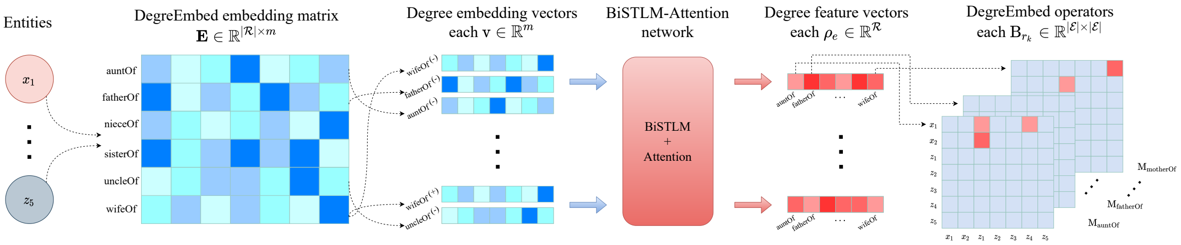

Entities
DegreEmbed embedding matrix
$\mathbf{E} \in \mathbb{R}^{|\mathcal{R}|\times m}$
Degree embedding vectors
each $\mathrm{v} \in \mathbb{R}^m$
BiSTLM-Attention
network
Degree feature vectors
each $\rho_e \in \mathbb{R}^{\mathcal{R}}$
DegreEmbed operators
each $\mathrm{B}_{r_k} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{E}|}$
x_1
z_5
auntOf
fatherOf
nieceOf
sisterOf
uncleOf
wifeOf
$\mathrm{wifeOf}^{(-)}$
$\mathrm{fatherOf}^{(-)}$
$\mathrm{auntOf}^{(-)}$
$\mathrm{wifeOf}^{(+)}$
$\mathrm{uncleOf}^{(-)}$
BiSTLM
+
Attention
auntOf
fatherOf
...
wifeOf
x_1 x_2 z_1 z_2 z_3 z_4 z_5
$\mathrm{M}_{\mathrm{motherOf}}$
$\mathrm{M}_{\mathrm{fatherOf}}$
$\mathrm{M}_{\mathrm{auntOf}}$

Fig. 3. An illustration of computing the DegreEmbed operators for the KG shown in Fig. 2. Superscripts (+) and (-) of the labels of degree embedding vectors denote their in and out direction. All DegreEmbed operators are initialized to zero matrices.

To compute the attention value on each relation imposed by entity e, we have

$$\rho_e = \mathrm{softmax}(WH + b) \tag{11}$$

Elements in $\rho_e \in \mathbb{R}^{|\mathcal{R}|}$ can be viewed as the weights for relations. At last, we replace the elements that are in the row identified by e and equal 1 in each adjacency matrix M_{r_k} by the k-th value of ρ_e. By following the same procedure for the other entities in the KG, we construct a new set of relational matrices $B_{r_1}, \dots, B_{r_{|\mathcal{R}|}}$, which are called *DegreEmbed* operators. The score function shown in Eq. (9) is updated accordingly as follows:

$$\phi'_L(t \mid q, h) = \left(\mathrm{v}_h^\top \prod_{l=1}^{L} \sum_{k=0}^{|\mathcal{R}|} a_k^l \mathrm{B}_{r_k} \right) \cdot \mathrm{v}_t, \tag{12}$$

where the $\mathrm{B}_{r_1}, \dots, \mathrm{B}_{r_{\mathcal{R}}}$ is our new DegreEmbed operators, and B_{r_0} is still the identity matrix. The whole process to compute the operators makes it possible to incorporate the information of entities for rule learning models, where the degree feature vector ρ_e can be viewed as the identification of the entity e. Remarkably, the DegreEmbed operators can be pre-trained due to its belonging to the inner attribute of a KG, thus resulting in a model that can be easily deployed in similar tasks. An overview of computing the DegreEmbed operators is illustrated in Fig. 3

Finally, the confidence scores are learned over the bidirectional LSTM followed by the attention using Eqs (13) and (14) for the temporal dependency among several consecutive steps. The `input` in Eq. (13) is `query` embedding from another lookup table. For $1 \leqslant i \leqslant L$ we have

$$\mathbf{h}_i, \mathbf{h}'_{L-i+1} = \mathrm{BiLSTM}(\mathbf{h}_{i-1}, \mathbf{h}'_{L-i}, \mathrm{input}), \tag{13}$$

$$[a_{i,1}, \dots, a_{i,|\mathcal{R}|}] = f_\theta\left([\mathbf{h}_i \mid\mid \mathbf{h}'_{L-i}] \right), \tag{14}$$

where $[a_{i,1}, \dots, a_{i,|\mathcal{R}|}]$ is the attention vector obtained by performing a linear transformation over concatenated forward and backward hidden states, followed by a softmax operator: $f_\theta(H) = \mathrm{softmax}(WH + b)$.

4.3. Optimization of the model

Loss construction. In general, this task of link prediction is treated as a classification of entities to build the loss. For each query $q(h, t)$ in a KG, we first split the objective function Eq. (12) into two parts: target vector v_t and prediction vector

$$s^\top = \mathrm{v}_h^\top \prod_{l=1}^{L} \sum_{k=0}^{|\mathcal{R}|} a_k^l \mathrm{B}_{r_k}, \tag{15}$$

and then our goal is to minimize the cross-entropy loss between v_t and $\mathrm{s}^\top$:

$$\ell_q(h, t) = -\sum_{i=1}^{|\mathcal{E}|} \{ \mathrm{v}_t[i] \cdot \log\left(\mathrm{s}[i]\right) + \\ (1 - \mathrm{v}_t[i]) \cdot \log\left(1 - \mathrm{s}[i]\right) \},$$

where i indexes elements in vector v_t and s.

Low-rank approximation. It can be shown that the final confidences obtained by expanding ϕ'_L are a rank one estimation of the *confidence value tensor* [22], and low-rank approximation is a popular method for tensor approximation. Hence we follow the work of [22] and rewrite Eq. (12) using rank of T approximation, as shown in Eq. (16).

$$\Phi_L(t \mid q, h) = \left(\mathrm{v}_h^\top \sum_{j=1}^{T} \prod_{l=1}^{L} \sum_{k=0}^{|\mathcal{R}|} a_{j,k}^l \mathrm{B}_{r_k} \right) \cdot \mathrm{v}_t, \tag{16}$$

Table 1

Statistics of datasets.

Dataset	# Relation	# Entity	# Triplets	# Facts	# Train	# Validation	# Test
FB15K-237	237	14541	310116	204087	68028	17535	20466
WN18	18	40943	151442	106088	35353	5000	5000
Family	12	3007	28356	17615	5868	2038	2835
Kinship	25	104	10686	6375	2112	1099	1100
UMLS	46	135	6529	4006	3009	569	633

More concretely, we update Eqs. (13) and (14), as is shown in Eqs. (17) and (18), by deploying number of T BiLSTMs of the same network structure, each of which can extract features from various dimensions.

$$\mathbf{h}_i^{(j)}, \mathbf{h}_{L-i+1}^{'(j)} = \text{BiLSTM}_j(\mathbf{h}_{i-1}^{(j)}, \mathbf{h}_{L-i}^{'(j)}, \text{input}) \quad (17)$$

$$[a_{i,1}^{(j)}, \dots, a_{i,|\mathcal{R}|}^{(j)}] = f_\theta\left([\mathbf{h}_i^{'(j)} \mid\mid \mathbf{h}_{L-i}^{'(j)}]\right), \quad (18)$$

where the superscripts of the hidden states identify their bidirectional LSTM.

5. Experiment

In this section, we report the evaluation results of our model on a knowledge graph completion task, where we compare the effectiveness of our model against the state-of-the-art learning systems for link prediction. Meanwhile, as DegreEmbed takes advantage in the interpretability in contrast to embedding-based methods, we also examine the rules mined by DegreEmbed with the help of the indicator *saturation*, which assesses the quality of rules from the corresponding topological structure of a KG. We show that the top-scored rules mined by our method coincide with those of high saturation scores, which in turn reflect the interpretability of our model. To this end, we use ablation study to show how different components of our model contribute to its performance.

The knowledge graph completion task we use is a canonical one as described in [16]. When training the model, the `query` and `head` are part of incomplete triplets for training, and the goal is to find the most possible entity as the answer `tail`. For example, if `nephewOf(Mike, Steve)` is missing from the knowledge graph, our goal is to learn rules for reasoning over the existing KG and retrieve `Steve` when presented with the query `nephewOf(Mike`, ?).

5.1. Experiment setting

5.1.1. Datasets

To evaluate our method for learning logic rules in heterogeneous KGs, we select the following datasets for knowledge graph completion task:

- *FB15K-237* [36], a more challenging version of FB15K [11] based on Freebase [8], a growing knowledge graph of general facts.
- *WN18* [26], a subset of knowledge graph WordNet [37, 38] constructed for a widely used dictionary.
- *Medical Language System (UMLS)* [39], from biomedicine, where the entities are biomedical concepts (*e.g.* `organism`, `virus`) and the relations consist of `affects` and `analyzes`, etc.
- *Kinship* [39], containing kinship relationships among members of a Central Australian native tribe.
- *Family* [39], containing individuals from multiple families that are biologically related.

Statistics about each dataset used in our experiments are presented in Table 1. All datasets are randomly split into 4 files: *facts*, *train*, *valid* and *test*, and the ratio is 6:2:1:1. The *facts* file contains a relatively large proportion of the triplets for constructing the KG. The *train* file is composed of query examples $q(h,t)$. The *valid* and *test* files both contain queries $q(h,t)$, in which the former is used for early stopping and the latter is for testing. Unlike the case of learning embeddings, our method does not necessarily require the entities in *train*, *valid* and *test* to overlap.

5.1.2. Evaluation metrics

During training on the task of knowledge graph completion, for each triplet (h,q,t), two queries are designed as $(h,q,?)$ and $(?,q,t)$ with answers t and h for data augmentation. During evaluation, for each query, we manually remove the edge (h,t) from KG for the correctness of reasoning results and the score is com-

Table 2

Saturations of the Family dataset. $\gamma_q^{\mathbf{p}_l}$, $\delta_q^{\mathbf{p}_l}$, $\eta_q^{\mathbf{p}_l}$ are *macro*, *micro* and *comprehensive* saturations. The results relating to a specific relation are sorted by the comprehensive saturation in descending order.

Rule	⇒	Relation	$\gamma_q^{\mathbf{p}_l}$	$\delta_q^{\mathbf{p}_l}$	$\eta_q^{\mathbf{p}_l}$
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{brotherOf}} Y$	⇒	$X \xRightarrow{\texttt{brotherOf}} Y$	.86	.14	.12
$X \xRightarrow{\texttt{nephewOf}} Z \xRightarrow{\texttt{uncleOf}} Y$	⇒		.77	.13	.10
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{sisterOf}} Y$	⇒		.81	.13	.10
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{fatherOf}} Y$	⇒		1.00	.08	.08
$X \xRightarrow{\texttt{nephewOf}} Z \xRightarrow{\texttt{auntOf}} Y$	⇒		.68	.11	.08
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{motherOf}} Y$	⇒		.98	.07	.07

puted for each entity, as well as the rank of the correct answer. For the computed ranks from all queries, we report the Mean Reciprocal Rank (MRR) and Hit@k under the filtered protocol [11]. MRR averages the reciprocal ranks of the answer entities and Hit@k computes the percentage of how many desired entities are ranked among top k.

When evaluating the interpretability of rules, we choose a set of indicators called *macro*, *micro* and *comprehensive* saturations that measure the probability of a rule pattern occurring in a certain relational subgraph $\mathcal{G}(r)$ from different angles. More specifically, these computational methods analyze the reasoning complexity from the inherent attributes of the graph structure G *w.r.t.* a KG $\mathcal{G}$.

Definition 2 (Macro Reasoning Saturation)**.** *Given a query $q \in \mathcal{R}$ and the maximum length L of a rule pattern $\mathbf{r}_l \in \mathbb{H}^L$, the **macro reasoning saturation** of $\mathbf{r}_l$ in relation to relation q, i.e. $\gamma_q^{\mathbf{r}_l}$, is the percentage of triples (h_i, q, t_j) in subgraph $\mathcal{G}(q)$ such that $\mathbf{r}_l(h_i, t_j) \Rightarrow q(h_i, t_j)$.*

We can compute the macro reasoning saturation $\gamma_q^{\mathbf{r}_l}$ using the following equation:

$$\gamma_q^{\mathbf{r}_l} = \frac{|U^{\mathbf{r}_l}|}{n^q}, \tag{19}$$

with $U^{\mathbf{r}_l}$ being the set $U^{\mathbf{r}_l} = \{(h,q,t) \mid (h,q,t) \in \mathcal{G}(q), \mathbf{r}_l(h,t) \Rightarrow q(h,t)\}$ that collects the factual triplets in $\mathcal{G}(q)$ as the reasoning candidates of rule $\mathbf{r}_l$, and $n^q = |\mathcal{G}(q)|$ being the number of edges (*i.e.* the number of triples) in $G(q)$. We can reasonably say that the larger $\gamma_q^{\mathbf{r}_l}$ grows, the more likely $\mathbf{r}_l$ can be as a proper inference of the query q. When $\gamma_q^{\mathbf{r}_l}$ equals 1, it means we can reason out every factual triple in $\mathcal{G}(q)$ through at least one rule path following the pattern $\mathbf{r}_l$.

Definition 3 (Micro Reasoning Saturation)**.** *Given the maximum length L of a rule pattern, we define the micro reasoning saturation of pattern $\mathbf{r}_l \in \mathbb{H}^L$ as following. Firstly, for a specific triple $\texttt{tri} = (h,q,t) \in \mathcal{G}$, i.e. $\delta^{\mathbf{r}_l}_{\texttt{tri}}$, is the percentage of the paths $\mathbf{p}_{l_i} \rhd \mathbf{r}_l$ such that $\mathbf{r}_l(h,t) \Rightarrow q(h,t)$ as to all paths from h to t.*

The equation to compute $\delta^{\mathbf{r}_l}_{\texttt{tri}}$ is

$$\delta^{\mathbf{r}_l}_{\texttt{tri}} = \frac{|V^{\mathbf{r}_l}|}{|V^L|} \tag{20}$$

where $V^{\mathbf{r}_l} = \{\mathbf{p}_{l_i} \mid \mathbf{p}_{l_i} \rhd \mathbf{r}_l, \mathbf{r}_l(h,t) \Rightarrow q(h,t)\}$, $V^L = \{\mathbf{p}_{k_j} \mid \mathbf{p}_{k_j} \rhd \mathbf{r}_k, \forall \mathbf{r}_k \in \mathbb{H}^L, \mathbf{r}_k(h,t) \Rightarrow q(h,t)\}$. $V^{\mathbf{r}_l}$ denotes the set of rule paths derived from the pattern $\mathbf{r}_l$ that is able to infer the fact (h,q,t), and V^L involves all the rules with their lengths no longer that L.

Then, we average $\delta^{\mathbf{r}_l}_{\texttt{tri}}$ over all triples $(h,q,t) \in \mathcal{G}(q)$ and get the **micro reasoning saturation** of the pattern $\mathbf{r}_l \in \mathbb{H}^L$ for query q:

$$\delta_q^{\mathbf{r}_l} = \frac{1}{n^q} \sum_{\texttt{tri} \in \mathcal{G}(q)} \delta^{\mathbf{r}_l}_{\texttt{tri}} \tag{21}$$

In Eqs. (19) and (21), $\gamma_q^{\mathbf{r}_l}$ and $\delta_q^{\mathbf{r}_l}$ assess how the probability to infer q following the pattern $\mathbf{r}_l$ respectively from a macro and a micro perspective. The higher the two indicators are, the easier for models to gain the inference $\mathbf{r}_l(h,t) \Rightarrow q(h,t)$. In order to obtain an overall result, we define the comprehensive reasoning saturation $\eta_q^{\mathbf{r}_l}$ by combining the two indicators through multiplication, as revealed in Eq. (22).

$$\eta_q^{\mathbf{r}_l} = \gamma_q^{\mathbf{r}_l} \times \delta_q^{\mathbf{r}_l} \tag{22}$$

We can imagine that the computation of comprehensive saturation on a certain logical rule $\mathbf{r}_l$ to infer the relation q involves two procedures: (1) select the triplets (h, q, t) in subgraph $\mathcal{G}(q)$ that imply $\mathbf{r}_l(h, t) \Rightarrow q(h, t)$ and (2) for each selected triplets, calculate the percentage of rule paths following the pattern $\mathbf{r}_l$ within all possible paths that imply $q(h, t)$.

We can take the relation $q = \texttt{auntOf}$ and the rule $\mathbf{r}_l = \texttt{sisterOf} \wedge \texttt{fatherOf}$ in Fig. 2 as an example to show the computation of saturations. In subgraph $\mathcal{G}(q)$, there are totally three triples (presented in red), thus $n^q = 3$. For the triple $(x_2, \texttt{auntOf}, z_2)$, two rule paths can contribute to its inference: $\texttt{wifeOf}(x_2, z_1) \wedge \texttt{uncleOf}(z_2, z_1)$ and $\texttt{sisterOf}(x_2, z_3) \wedge \texttt{fatherOf}(z_3, z_1)$. In the same way, we can see there are one and two rule paths for $(x_1, \texttt{auntOf}, z_1)$ and $(x_1, \texttt{auntOf}, z_4)$ respectively. The rule $\mathbf{r}_l = \texttt{sisterOf} \wedge \texttt{fatherOf}$ appears as an inference among all these three triples, therefore the macro saturation is $\gamma_q^{\mathbf{r}_l} = 3/n^q = 100\%$. More detailed information can be extracted through computing the micro saturation. The rule $\mathbf{r}_l$ takes a percentage of 50% among all paths for the triple $(x_2, \texttt{auntOf}, z_1)$, while 100% and 50% for the other two triples. Thus, the micro saturation of $\mathbf{r}_l$ for q is $\delta_q^{\mathbf{r}_l} = (0.5 + 1 + 0.5)/n^q = 67\%$. Finally, we can compute the comprehensive saturation $\eta_q^{\mathbf{r}_l} = \gamma_q^{\mathbf{r}_l} \times \delta_q^{\mathbf{r}_l} = 67\%$.

We show a small subset of saturations computed from Family dataset in Table. 2 for joint evaluation with logical rules mined by our model. More results can be obtained in App. C.

5.1.3. Comparison of algorithms

In experiment, the performance of our model is compared with that of the following algorithms:

- Rule learning algorithms. Since our model is based on neural logic programming, we choose Neural LP and a Neural LP-based method DRUM [22]. We also consider a probabilistic model called RNNLogic [40].
- Embedding-based algorithms. We choose several embedding-based algorithms for comparison of the expressive power of our model, including TransE [11], DistMult [41], ComplEx [12], TuckER [27] and RotatE [25].

The implementations of the above models we use are available at the links listed in App. A.

5.1.4. Model configuration

Our model is implemented using PyTorch [42] and the code will be publicly available. We use the same hyperparameter setting during evaluation on all datasets. The query and entity embedding have the dimension 128 and are both randomly initialized. The hidden state dimension of BiLSTM(s) for entity and degree embedding are also 128. As for optimization algorithm, we use mini-batch ADAM [43] with the batch size 128 and the learning rate initially set to 0.001. We also observe that the whole model tends to be more trainable if we use the normalization skill.

Note that Neural LP [16], DRUM [22] and our method all conform to a similar reasoning framework. Hence, to reach a fair comparison, we ensure the same hyperparameter configuration during experiments on these models, where the maximum rule length L is 2 and the rank T is 3 for DRUM and DegreEmbed, because Neural LP is not developed using the low-rank approximation method.

5.2. Results on KGC task

We compare our DegreEmbed to several baseline models on the KGC benchmark datasets as stated in the Section 5.1.1 and Section 5.1.3. Our results on the

Table 3

Knowledge graph completion performance comparison. Hit@k (H@k) is in %.

	Family				Kinship				UMLS			
	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
TransE	.34	7	53	86	.26	1	42	76	.57	28	84	96
DistMult	.58	39	71	91	.51	36	57	87	.73	63	81	90
ComplEx	.83	72	94	98	.61	44	71	92	.79	**69**	87	95
TuckER	.43	28	52	72	.60	46	70	86	.73	63	81	91
RotatE	.90	85	95	99	.65	50	76	93	.73	64	82	94
RNNLogic	.93	**91**	95	99	.64	50	73	93	.75	63	83	92
Neural LP	.91	86	95	99	.62	48	69	91	.75	62	86	92
DRUM	.94	90	98	99	.58	43	67	90	**.80**	66	**94**	97
DegreEmbed	**.95**	**91**	**99**	**100**	**.70**	**57**	**79**	**94**	**.80**	65	**94**	**98**

Table 4

Top rules without reverse queries mined by DegreEmbed on the Family dataset.

Rule	⇒	Relation	Confidence
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{sisterOf}} Y$	⇒	$X \xRightarrow{\texttt{brotherOf}} Y$	1.00
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{brotherOf}} Y$	⇒		0.81
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{motherOf}} Y$	⇒		0.55
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{fatherOf}} Y$	⇒		0.18

Table 5

Top rules without reverse queries mined by DRUM on the Family dataset.

Rule	⇒	Relation	Confidence
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{motherOf}} Y$	⇒	$X \xRightarrow{\texttt{brotherOf}} Y$	1.00
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{brotherOf}} Y$	⇒		0.52
$X \xRightarrow{\texttt{mohterOf}} Z \xRightarrow{\texttt{motherOf}} Y$	⇒		0.50
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{fatherOf}} Y$	⇒		0.48
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{sisterOf}} Y$	⇒		0.35
$X \xRightarrow{\texttt{motherOf}} Z \xRightarrow{\texttt{fatherOf}} Y$	⇒		0.13

selected benchmark datasets are summarized in Table 3 and App. B.

We notice that except that ComplEx [12] produces the best result among all methods on UMLS under the evaluation of Hit@1, all models are outperformed by DegreEmbed with a clear margin in Table 3, especially on the dataset Kinship where we can see about 10% improvement on some metrics. As expected, incorporating entity embedding enhances the expressive power of DegreEmbed and thus benefits to reasoning on heterogeneous KGs. In Table 7, our model achieves state-of-the-art performance on WN18. It is intriguing that embedding-based methods provide better predictions on FB15K-237 dataset, with rule based methods, including RNNLogic, Neural LP, DRUM and ours, left behind. As pointed in [26], there are inverse relations from the training data present in the test data in FB15K, which is called the problem of test set leakage, resulting in the variant FB15K-237 where inverse relations are removed. No wonder that methods depending on logic rule learning fails on this dataset.

Notably, DegreEmbed not only is capable of producing state-of-the-art results on KGC task thanks to the degree embedding of entities, but also maintains the advantage of logic rule learning that enables our model to be interpretable to humans, which is of vital significance in current research of intelligent systems. We will show the experiment results on the interpretability of our DegreEmbed model later.

5.3. Interpretability of our model

To demonstrate the interpretability of our method, we first report the logical rules mined by our model and compare them with those by DRUM [22]. Then we visualize the embedding vectors learned through the proposed technique degree embedding to prove its expressive power.

5.3.1. Quality of mined rules

Apart from reaching state-of-the-art performance on KGC task which is largely thanks to the mechanism of entity embedding, our DegreEmbed, as a knowledge graph reasoning model based on logic rule mining, is of excellent interpretability as well. Our work follows the Neural LP [16] framework, which successfully combines structure learning and parameter learning to generate rules along with confidence scores.

In this section, we report evaluation results on explanations of our model where some of the rules learned by DegreEmbed and DRUM are shown. As for evaluation metrics, we use the indicator *saturations* to objectively assess the quality of mined rules in a computable manner. We conduct two separate KGC experiments for generating the logical rules where the only difference is whether the inverted queries are learned. For better visualization purposes, experiments are done on the Family dataset, while other datasets such as UMLS produce similar results.

We sort the rules by their normalized confidence scores, which are computed by dividing by the maximum confidence of rules for each relation, and show top rules mined by our DegreEmbed and DRUM without augmented queries respectively in Table 4 and Table 5. Saturations of rules according to specific relations are shown in Table 2. For more results of saturations, learned rules *w.r.t.* both pure and augmented queries, please refer to the appendix.

By referring to the results given by computing saturations, we can see the rules mined by our model solidly agree with the ones with high saturation level. Meanwhile, our model obviously gets rid of the noises rendered by the heterogeneousness of the dataset through blending entity attributes (*e.g.* gender of entities) into rule learning. The rules mined for predicting the relation `brotherOf`, such as `brotherOf` ∧ `sisterOf` and `brotherOf` ∧ `brotherOf`, all show up with a male-type relation at the first hop. However, there are logically incorrect rules mined by DRUM which are highlighted by red in Table 5. We think this is mainly because DRUM does not take entity attributes in to account. In this case, our DegreEmbed model is capable of learning meaningful rules, which indeed proves the interpretability of our model.

5.3.2. *Learned entity embeddings*

To explain the learned degree embedding, we visualize the embeddings vectors of some entities from the Family dataset. We use t-SNE [44] to project the embeddings to two-dimensional space and plot them in Fig. 4. In order to obtain the entity embeddings, we first train our DegreEmbed model on Family with the same hyperparameter settings mentioned in Section 5.1.4, and store the entire entity embedding matrix given by Eq. 10. Then, we classify the entities according to their *degree feature vector* proposed in Section 4.2 and choose top ten most populated clusters marked with various colors to plot in Fig. 4.

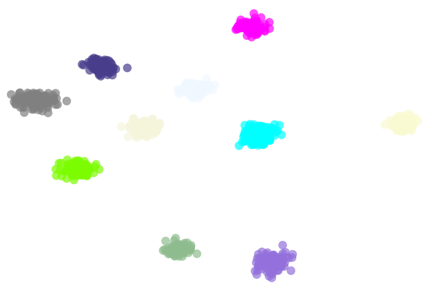


Fig. 4. A t-SNE plot of the entity embedding of our trained model on Family dataset. Node colors denote their classes (*i.e.* degree feature vectors).

Note that, we use a logarithmic scale for the embedding plot to get better visualization results. In fact, the representation of entities exhibits localized clustering in the projected 2D space, which verifies the capability of our model to encode latent features of entities in heterogeneous KGs through their degrees.

5.4. *Ablation study*

To study the necessity of each component of our method, we gradually change the configuration of each component and observe how the model performance varies.

Degree embedding. In this work, degree embedding is proposed as a new technique of entity embedding for incorporating heterogeneous information in KGs. We successively replace this component with learned entity embeddings from five pre-trained embedding-based models listed in Section 5.1.3 on three datasets. We measure the Hit@1, Hit@3 and Hit@10 metrics and show the results on Family in Fig. 5. Results on another two datasets are placed in the appendix. In summary, the original model using degree embedding to encode entities produces the best results among all variants. We hypothesis that this is due to the fact that many inner attributes of entities are lost in the embeddings of those variants while DegreEmbed can learn to utilize these features implicitly.

Low-rank approximation. Tensor approximation of rank T enables our model to learn latent features from various dimensions, as show Eqs. (17) and (18). We conduct experiments on three datasets and show how model behavior differs with rank ranging from 1 to 4 in Fig. 6. Training curves in Fig. (6a) imply that model may converge faster with lower training loss as rank goes up. However, Fig. 6b demonstrates that

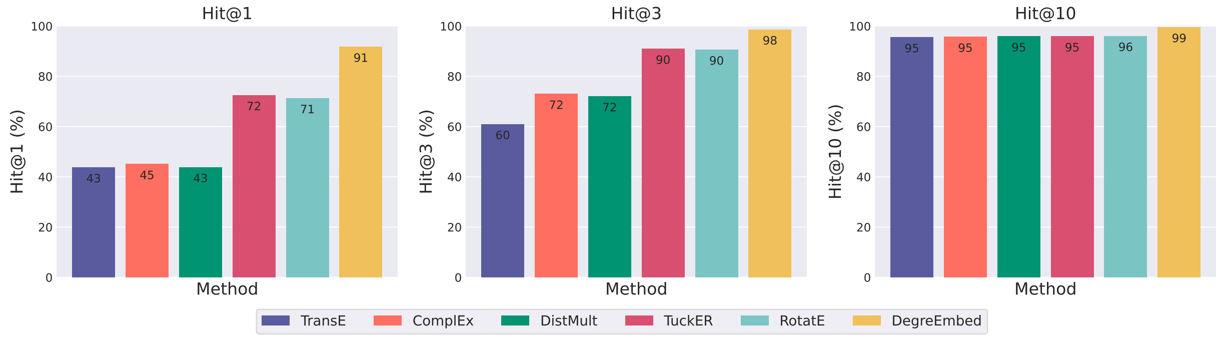
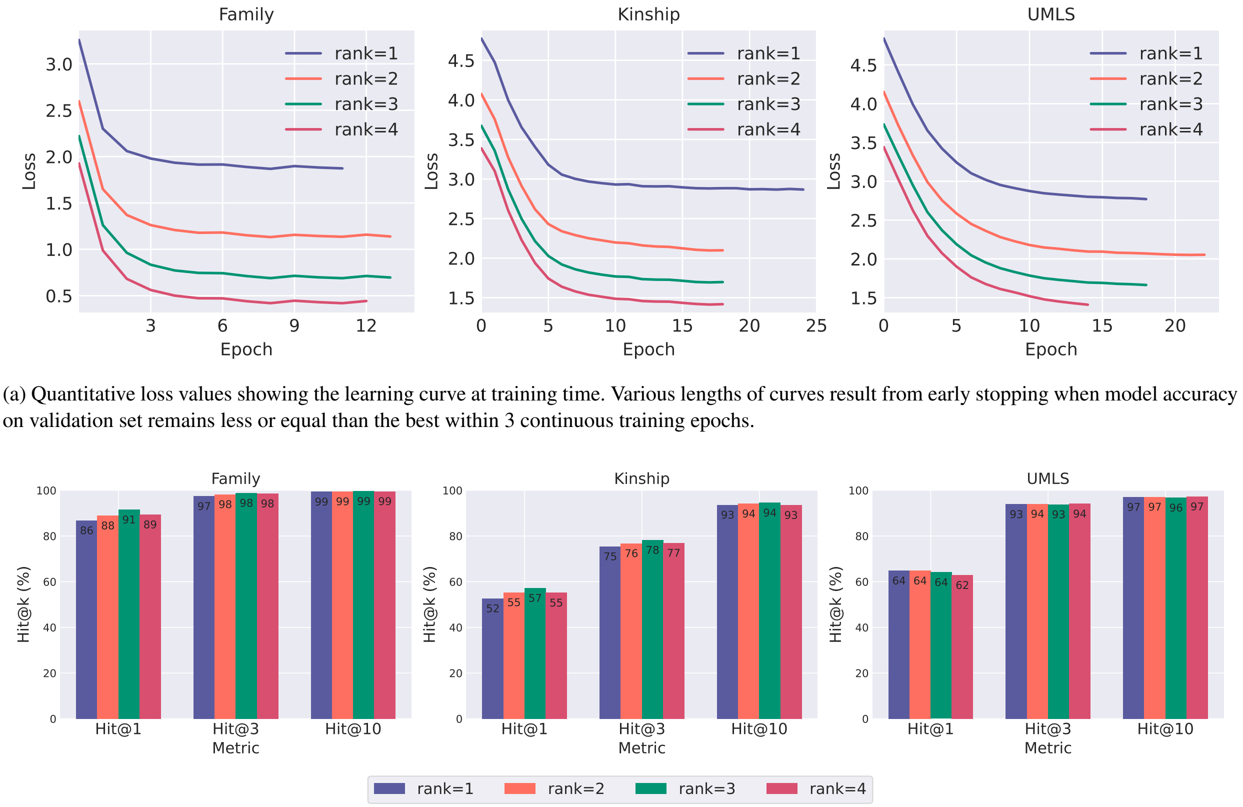


Fig. 5. Model performance on Family with the original entity embeddings replaced by pre-trained ones from embedding-based methods. Hit@k is in %. The number inside each bar indicates its Hit@k value.



(a) Quantitative loss values showing the learning curve at training time. Various lengths of curves result from early stopping when model accuracy on validation set remains less or equal than the best within 3 continuous training epochs.

(b) Model performance under the evaluation of Hit@k on test datasets.

Fig. 6. Comparison among DegreEmbed variants with different ranks on three benchmark datasets.

higher rank does not necessarily bring better test results. We conjecture that this is because the amount of learnable features of distinct dimensions varies from dataset to dataset, where the choice of rank matters a lot. An intriguing insight can be obtained by combining Fig. (6a) and Fig. (6b): training loss degrades as model rank increases while it barely contributes to results on test sets, which provides a view of over-fitting.

6. Conclusions

In this paper, a logic rule learning model called DegreEmbed has been proposed for reasoning more effectively in heterogeneous knowledge graphs, where there exist entities and relations of different types. Based on mining logic rules, DegreEmbed simultaneously leverages latent knowledge of entities by learning embedding vectors for them, where the degrees of the entities are closely observed. Experiment results show that our model benefits from the advantages of both embedding-based methods and rule learning systems, as one can see DegreEmbed outperforms the state-of-the-art models with a clear margin, and it produces high-quality rules with great interpretability. In the future, we would like to optimize the way of entity embedding to increase the expressive power of logic rule learning models for knowledge graph reasoning.

Acknowledgements

The work of this paper is supported by the "National Key R&D Program of China" (2020YFB2009502), "the Fundamental Research Funds for the Central Universities" (Grant No. HIT.NSRIF.2020098).

References

[1] Z. Liu, Z.-Y. Niu, H. Wu and H. Wang, Knowledge aware conversation generation with explainable reasoning over augmented graphs, *arXiv preprint arXiv:1903.10245* (2019).

[2] S. Moon, P. Shah, A. Kumar and R. Subba, Opendialkg: Explainable conversational reasoning with attention-based walks over knowledge graphs, in: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 845–854.

[3] C. Xiong, R. Power and J. Callan, Explicit semantic ranking for academic search via knowledge graph embedding, in: *Proceedings of the 26th international conference on world wide web*, 2017, pp. 1271–1279.

[4] R.T. Sousa, S. Silva and C. Pesquita, Evolving knowledge graph similarity for supervised learning in complex biomedical domains, *BMC bioinformatics* **21**(1) (2020), 1–19.

[5] S.K. Mohamed, V. Nováček and A. Nounu, Discovering protein drug targets using knowledge graph embeddings, *Bioinformatics* **36**(2) (2020), 603–610.

[6] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun and W. Zhang, Knowledge vault: A web-scale approach to probabilistic knowledge fusion, in: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 601–610.

[7] D. Krompaß, S. Baier and V. Tresp, Type-constrained representation learning in knowledge graphs, in: *International semantic web conference*, Springer, 2015, pp. 640–655.

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

[9] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak and Z. Ives, Dbpedia: A nucleus for a web of open data, in: *The semantic web*, Springer, 2007, pp. 722–735.

[10] S. Ji, S. Pan, E. Cambria, P. Marttinen and S.Y. Philip, A survey on knowledge graphs: Representation, acquisition, and applications, *IEEE Transactions on Neural Networks and Learning Systems* (2021).

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

[12] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier and G. Bouchard, Complex embeddings for simple link prediction, in: *International conference on machine learning*, PMLR, 2016, pp. 2071–2080.

[13] Y. Chen, S. Goldberg, D.Z. Wang and S.S. Johri, Ontological pathfinding, in: *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 835–846.

[14] L. Galárraga, C. Teflioudi, K. Hose and F.M. Suchanek, Fast rule mining in ontological knowledge bases with AMIE

$$+$$

+, *The VLDB Journal* **24**(6) (2015), 707–730.

[15] S. Muggleton and L. De Raedt, Inductive logic programming: Theory and methods, *The Journal of Logic Programming* **19** (1994), 629–679.

[16] F. Yang, Z. Yang and W.W. Cohen, Differentiable learning of logical rules for knowledge base reasoning, *arXiv preprint arXiv:1702.08367* (2017).

[17] M. Qu and J. Tang, Probabilistic logic neural networks for reasoning, *arXiv preprint arXiv:1906.08495* (2019).

[18] Y. Zhang, X. Chen, Y. Yang, A. Ramamurthy, B. Li, Y. Qi and L. Song, Efficient probabilistic logic reasoning with graph neural networks, *arXiv preprint arXiv:2001.11850* (2020).

[19] N. Lao and W.W. Cohen, Relational retrieval using a combination of path-constrained random walks, *Machine learning* **81**(1) (2010), 53–67.

[20] A. Neelakantan, B. Roth and A. McCallum, Compositional vector space models for knowledge base completion, *arXiv preprint arXiv:1504.06662* (2015).

[21] R. Das, A. Neelakantan, D. Belanger and A. McCallum, Chains of reasoning over entities, relations, and text using recurrent neural networks, *arXiv preprint arXiv:1607.01426* (2016).

[22] A. Sadeghian, M. Armandpour, P. Ding and D.Z. Wang, Drum: End-to-end differentiable rule mining on knowledge graphs, *arXiv preprint arXiv:1911.00055* (2019).

[23] P.-W. Wang, D. Stepanova, C. Domokos and J.Z. Kolter, Differentiable learning of numerical rules in knowledge graphs, in: *International Conference on Learning Representations*, 2019.

[24] Y. Yang and L. Song, Learn to explain efficiently via neural logic inductive learning, *arXiv preprint arXiv:1910.02481* (2019).

[25] Z. Sun, Z.-H. Deng, J.-Y. Nie and J. Tang, Rotate: Knowledge graph embedding by relational rotation in complex space, *arXiv preprint arXiv:1902.10197* (2019).

[26] T. Dettmers, P. Minervini, P. Stenetorp and S. Riedel, Convolutional 2d knowledge graph embeddings, in: *Thirty-second AAAI conference on artificial intelligence*, 2018.

[27] I. Balažević, C. Allen and T.M. Hospedales, Tucker: Tensor factorization for knowledge graph completion, *arXiv preprint arXiv:1901.09590* (2019).

[28] L.R. Tucker, Some mathematical notes on three-mode factor analysis, *Psychometrika* **31**(3) (1966), 279–311.

[29] W. Zhang, B. Paudel, L. Wang, J. Chen, H. Zhu, W. Zhang, A. Bernstein and H. Chen, Iteratively learning embeddings and rules for knowledge graph reasoning, in: *The World Wide Web Conference*, 2019, pp. 2366–2377.

[30] P. Wang, D. Dou, F. Wu, N. de Silva and L. Jin, Logic Rules powered knowledge graph embedding, *arXiv:1903.03772 [cs]* (2019).

[31] F.N. Stokman and P.H. de Vries, Structuring knowledge in a graph, in: *Human-computer interaction*, Springer, 1988, pp. 186–206.

[32] W.W. Cohen, Tensorlog: A differentiable deductive database, *arXiv preprint arXiv:1605.06523* (2016).

[33] W.W.C.F.Y. Kathryn and R. Mazaitis, Tensorlog: Deep learning meets probabilistic databases, *Journal of Artificial Intelligence Research* **1** (2018), 1–15.

[34] K. Guu, J. Miller and P. Liang, Traversing knowledge graphs in vector space, *arXiv preprint arXiv:1506.01094* (2015).

[35] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural computation* **9**(8) (1997), 1735–1780.

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

[37] G.A. Miller, WordNet: a lexical database for English, *Communications of the ACM* **38**(11) (1995), 39–41.

[38] G.A. Miller, *WordNet: An electronic lexical database*, MIT press, 1998.

[39] S. Kok and P. Domingos, Statistical predicate invention, in: *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 433–440.

[40] M. Qu, J. Chen, L.-P. Xhonneux, Y. Bengio and J. Tang, Rnnlogic: Learning logic rules for reasoning on knowledge graphs, *arXiv preprint arXiv:2010.04029* (2020).

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

[42] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., Pytorch: An imperative style, high-performance deep learning library, *Advances in neural information processing systems* **32** (2019), 8026–8037.

[43] D.P. Kingma and J. Ba, Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980* (2014).

[44] L. Van der Maaten and G. Hinton, Visualizing data using t-SNE., *Journal of machine learning research* **9**(11) (2008).

Appendix A. Algorithm URLs

Table 6

Available links to the models used in this work.

Algorithm	Link
TransE, DistMult and ComplEx	https://github.com/Accenture/AmpliGraph
TuckER	https://github.com/ibalazevic/TuckER
RotatE	https://github.com/liyirui-git/KnowledgeGraphEmbedding_RotatE
RNNLogic	https://github.com/DeepGraphLearning/RNNLogic
Neural LP	https://github.com/fanyangxyz/Neural-LP
DRUM	https://github.com/alisadeghian/DRUM
DegreEmbed (ours)	https://github.com/lirt1231/DegreEmbed

Appendix B. Results on FB15K-237 and WN18

Table 7

Knowledge graph completion results on FB15K-237 and WN18. Hit@k is in %.

	FB15K-237				WN18			
	MRR	Hit@1	Hit@3	Hit@10	MRR	Hit@1	Hit@3	Hit@10
TransE	.15	5	19	25	.36	4	63	81
DistMult	.25	17	28	42	.71	56	83	93
ComplEx	.26	17	29	44	.90	88	92	94
TuckER	**.36**	**27**	**39**	**46**	.94	93	94	95
RotatE	.34	24	38	53	**.95**	**94**	**95**	96
RNNLogic	.29	21	31	43	.94	93	94	96
Neural LP	.25	19	27	37	.94	93	94	95
DRUM	.25	19	28	38	.54	49	54	66
DegreEmbed	.25	19	27	38	**.95**	**94**	**95**	**97**

Appendix C. Extension to Table 2: more saturations of Family

Table 8

Saturations of the Family dataset. $\gamma_q^{\mathbf{p}_l}$, $\delta_q^{\mathbf{p}_l}$, $\eta_q^{\mathbf{p}_l}$ are *macro*, *micro* and *comprehensive* saturations. The results relating to a specific relation are sorted by the comprehensive saturation in descending order.

Rule	⇒	Relation	$\gamma_q^{\mathbf{p}_l}$	$\delta_q^{\mathbf{p}_l}$	$\eta_q^{\mathbf{p}_l}$
$X \xRightarrow{\texttt{nephewOf}} Z \xRightarrow{\texttt{brotherOf}} Y$	⇒	$X \xRightarrow{\texttt{nephewOf}} Y$	.86	.25	.21
$X \xRightarrow{\texttt{nephewOf}} Z \xRightarrow{\texttt{sisterOf}} Y$	⇒		.79	.22	.17
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{nephewOf}} Y$	⇒		.79	.21	.16
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{nieceOf}} Y$	⇒		.72	.17	.12
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{brotherOf}} Y$	⇒		.64	.10	.06
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{sisterOf}} Y$	⇒		.36	.05	.02
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{sonOf}} Y$	⇒	$X \xRightarrow{\texttt{daughterOf}} Y$	.68	.25	.17
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{daughterOf}} Y$	⇒		.61	.20	.12
$X \xRightarrow{\texttt{daughterOf}} Z \xRightarrow{\texttt{husbandOf}} Y$	⇒		.46	.15	.07
$X \xRightarrow{\texttt{daughterOf}} Z \xRightarrow{\texttt{wifeOf}} Y$	⇒		.46	.14	.06
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{uncleOf}} Y$	⇒	$X \xRightarrow{\texttt{auntOf}} Y$	.89	.26	.23
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{auntOf}} Y$	⇒		.85	.22	.19
$X \xRightarrow{\texttt{auntOf}} Z \xRightarrow{\texttt{brotherOf}} Y$	⇒		.83	.21	.17
$X \xRightarrow{\texttt{auntOf}} Z \xRightarrow{\texttt{sisterOf}} Y$	⇒		.75	.18	.13
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{fatherOf}} Y$	⇒		.66	.09	.06
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{motherOf}} Y$	⇒		.34	.05	.02
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{brotherOf}} Y$	⇒	$X \xRightarrow{\texttt{sisterOf}} Y$	.89	.15	.13
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{sisterOf}} Y$	⇒		.84	.14	.12
$X \xRightarrow{\texttt{nieceOf}} Z \xRightarrow{\texttt{uncleOf}} Y$	⇒		.78	.13	.10
$X \xRightarrow{\texttt{auntOf}} Z \xRightarrow{\texttt{nephewOf}} Y$	⇒		.67	.12	.08
$X \xRightarrow{\texttt{daughterOf}} Z \xRightarrow{\texttt{fatherOf}} Y$	⇒		1.00	.07	.07
$X \xRightarrow{\texttt{daughterOf}} Z \xRightarrow{\texttt{motherOf}} Y$	⇒		.99	.07	.07
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{sonOf}} Y$	⇒	$X \xRightarrow{\texttt{sonOf}} Y$	.64	.24	.15
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{daughterOf}} Y$	⇒		.56	.19	.10
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{husbandOf}} Y$	⇒		.46	.16	.08
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{wifeOf}} Y$	⇒		.46	.14	.06
$X \xRightarrow{\texttt{nephewOf}} Z \xRightarrow{\texttt{brotherOf}} Y$	⇒		.39	.12	.05

Appendix D. Extension to Table 4: top rules obtained by our model

Table 9

Top rules without reverse queries mined by DegreEmbed on the Family dataset.

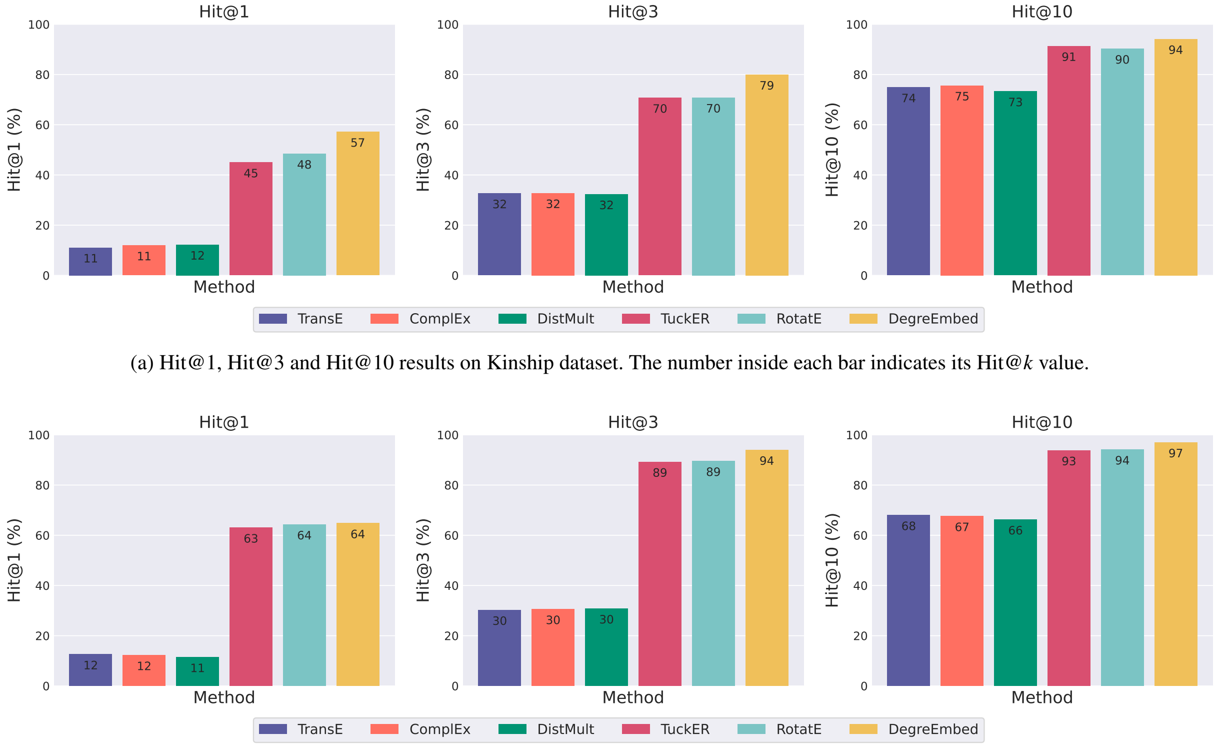
Rule	⇒	Relation	Confidence
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{nephewOf}} Y$	⇒	$X \xRightarrow{\texttt{nephewOf}} Y$	1.00
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{nieceOf}} Y$	⇒		0.88
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{sisterOf}} Y$	⇒		0.34
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{brotherOf}} Y$	⇒		0.16
$X \xRightarrow{\texttt{nephewOf}} Z \xRightarrow{\texttt{sisterOf}} Y$	⇒		0.13
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{sonOf}} Y$	⇒	$X \xRightarrow{\texttt{daughterOf}} Y$	1.00
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{daughterOf}} Y$	⇒		0.84
$X \xRightarrow{\texttt{daughterOf}} Z \xRightarrow{\texttt{wifeOf}} Y$	⇒		0.72
$X \xRightarrow{\texttt{daughterOf}} Z \xRightarrow{\texttt{husbandOf}} Y$	⇒		0.24
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{motherOf}} Y$	⇒	$X \xRightarrow{\texttt{auntOf}} Y$	1.00
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{fatherOf}} Y$	⇒		0.77
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{auntOf}} Y$	⇒		0.77
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{uncleOf}} Y$	⇒		0.31
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{sisterOf}} Y$	⇒	$X \xRightarrow{\texttt{sisterOf}} Y$	1.00
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{brotherOf}} Y$	⇒		0.90
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{motherOf}} Y$	⇒		0.39
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{sonOf}} Y$	⇒	$X \xRightarrow{\texttt{sonOf}} Y$	1.00
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{daughterOf}} Y$	⇒		0.67
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{husbandOf}} Y$	⇒		0.56
$X \xRightarrow{\texttt{sonOf}} Z \xRightarrow{\texttt{wifeOf}} Y$	⇒		0.39

Table 10

Top rules with reverse queries mined by DegreEmbed on the Family dataset.

Rule	⇒	Relation	Confidence
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{inv\_uncleOf}} Y$	⇒	$X \xRightarrow{\texttt{nephewOf}} Y$	1.00
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{inv\_auntOf}} Y$	⇒		0.39
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{inv\_auntOf}} Y$	⇒		0.36
$X \xRightarrow{\texttt{inv\_brotherOf}} Z \xRightarrow{\texttt{inv\_auntOf}} Y$	⇒		0.32
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{nieceOf}} Y$	⇒		0.29
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{inv\_uncleOf}} Y$	⇒		0.22
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{inv\_motherOf}} Y$	⇒	$X \xRightarrow{\texttt{daughterOf}} Y$	1.00
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{inv\_fatherOf}} Y$	⇒		0.67
$X \xRightarrow{\texttt{inv\_brotherOf}} Z \xRightarrow{\texttt{inv\_fatherOf}} Y$	⇒		0.31
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{inv\_fatherOf}} Y$	⇒		0.26
$X \xRightarrow{\texttt{inv\_brotherOf}} Z \xRightarrow{\texttt{inv\_motherOf}} Y$	⇒		0.18
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{inv\_motherOf}} Y$	⇒		0.17
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{inv\_sisterOf}} Y$	⇒	$X \xRightarrow{\texttt{brotherOf}} Y$	1.00
$X \xRightarrow{\texttt{inv\_brotherOf}} Z \xRightarrow{\texttt{inv\_brotherOf}} Y$	⇒		0.57
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{brotherOf}} Y$	⇒		0.55
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{sisterOf}} Y$	⇒		0.37
$X \xRightarrow{\texttt{inv\_brotherOf}} Z \xRightarrow{\texttt{inv\_sisterOf}} Y$	⇒		0.20
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{motherOf}} Y$	⇒	$X \xRightarrow{\texttt{auntOf}} Y$	1.00
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{fatherOf}} Y$	⇒		0.26
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{inv\_nephewOf}} Y$	⇒		0.26
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{inv\_nieceOf}} Y$	⇒		0.23
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{inv\_daughterOf}} Y$	⇒		0.18
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{sisterOf}} Y$	⇒	$X \xRightarrow{\texttt{sisterOf}} Y$	1.00
$X \xRightarrow{\texttt{sisterOf}} Z \xRightarrow{\texttt{inv\_brotherOf}} Y$	⇒		0.72
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{inv\_sisterOf}} Y$	⇒		0.51
$X \xRightarrow{\texttt{inv\_brotherOf}} Z \xRightarrow{\texttt{inv\_sisterOf}} Y$	⇒		0.16
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{inv\_brotherOf}} Y$	⇒		0.10
$X \xRightarrow{\texttt{inv\_brotherOf}} Z \xRightarrow{\texttt{inv\_motherOf}} Y$	⇒	$X \xRightarrow{\texttt{sonOf}} Y$	1.00
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{inv\_fatherOf}} Y$	⇒		0.43
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{inv\_motherOf}} Y$	⇒		0.37
$X \xRightarrow{\texttt{inv\_sisterOf}} Z \xRightarrow{\texttt{inv\_motherOf}} Y$	⇒		0.31
$X \xRightarrow{\texttt{brotherOf}} Z \xRightarrow{\texttt{inv\_fatherOf}} Y$	⇒		0.28
$X \xRightarrow{\texttt{inv\_brotherOf}} Z \xRightarrow{\texttt{inv\_fatherOf}} Y$	⇒		0.27

Appendix E. Ablation results



(a) Hit@1, Hit@3 and Hit@10 results on Kinship dataset. The number inside each bar indicates its Hit@k value.

(b) Hit@1, Hit@3 and Hit@10 results on UMLS dataset. The number inside each bar indicates its Hit@k value.

Fig. 7. Model performance on Kinship and UMLS with the original entity embeddings replaced by pre-trained ones from embedding-based methods. Hit@k is in %.