

Towards Loosely-Coupling Knowledge Graph Embeddings and Ontology-based Reasoning

Zoi Kaoudi
Technische Universität Berlin
Germany

Abelardo Carlos Martinez
Lorenzo*
Sapienza University of Rome
Italy

Volker Markl
Technische Universität Berlin
Germany

ABSTRACT

Knowledge graph completion (a.k.a. link prediction), i.e., the task of inferring missing information from knowledge graphs, is a widely used task in many applications, such as product recommendation and question answering. The state-of-the-art approaches of knowledge graph embeddings and/or rule mining and reasoning are data-driven and, thus, solely based on the information the input knowledge graph contains. This leads to unsatisfactory prediction results which make such solutions inapplicable to crucial domains such as healthcare. To further enhance the accuracy of knowledge graph completion we propose to loosely-couple the data-driven power of knowledge graph embeddings with domain-specific reasoning stemming from experts or entailment regimes (e.g., OWL2). In this way, we not only enhance the prediction accuracy with domain knowledge that may not be included in the input knowledge graph but also allow users to plugin their own knowledge graph embedding and reasoning method. Our initial results show that we enhance the MRR accuracy of vanilla knowledge graph embeddings by up to 3x and outperform hybrid solutions that combine knowledge graph embeddings with rule mining and reasoning up to 3.5x MRR.

1 INTRODUCTION

Knowledge graphs (KGs) are extensively used in many application domains, such as search engines, product recommendation, and bioinformatics. Nodes in the graph denote entities and edges represent the relations between entities. KGs are very often accompanied by ontologies which specify the schema that a KG follows, i.e., the type of entities and the relationships between different types. As KGs are usually created from incomplete data sources, it is often the case that they are missing information. The task of *KG completion* (a.k.a. *link prediction*) aims to infer such missing information.

Two known methods used for the problem of KG completion is knowledge graph embeddings (KGE) and rule mining. KGEs are representations of entities and relations into a low-dimensional space which can be used to predict whether a missing link in the graph is true or not. Rule-mining methods mine logical rules from the KG and then apply them to reason about new information. Unfortunately, both methods fall short in providing good prediction results for any arbitrary KG. The reason for this is that they both rely on the data and patterns existing in the input KG and are thus unable to generalize in cases where there is inadequate information. For instance, KGEs work well in predicting links for popular entities and relations, i.e., dense parts of the graph [8]. Similarly, rule-mining methods work well when there is enough evidence supporting the patterns they mine. However, real-world KGs consist of both

dense and sparse areas and thus, current approaches fail to provide satisfactory prediction results in the general case and highly depend on the input data.

Recently there have been few proposals on hybrid solutions, combining KGEs with rule mining and reasoning [3, 12]. Although they achieve slightly better predictive performance than the monolithic approaches, they still fail to solve the problem as both their embedding and rule-mining components are data-driven. This means that in lack of sufficient evidence in the KG such hybrid methods are also unable to infer useful information. In addition, these solutions, are tightly coupled with specific KGE model and rule-based reasoning methods and require significant effort to adapt them to new, more promising ones, if possible at all.

A natural remedy to improve KG prediction performance is the inclusion of domain expertise. Domain knowledge can be encoded by ontologies and rules which are either user-defined or specified by an entailment regime (e.g., RDFS or OWL2). However, none of the aforementioned approaches leverages such information. In fact, [4] states that most existing KGEs are not capable of encoding ontological information. [5] use ontological information but only for improving the negative samples that KGEs require.

We propose a different approach where we combine the data-driven KGE approach with rule-based reasoning based on ontologies and domain experts and allow them to work in tandem: KGEs are used to extract plausible triples which are used together with the KG for the ontology-based reasoning. At the same time triples inferred by reasoning are used to enhance the prediction of KGEs. This chain continues until no more new information is extracted. Combining these two approaches is quite challenging for the following reasons: (i) The two approaches are completely different: KGE reasoning is inductive (bottom up), making generalizations over input data, while ontology-based rule reasoning is deductive (top-down), reaching to conclusions given general rules. Therefore, it is counter-intuitive on how these two can be combined. (ii) KGEs are not capable of generating information out of the box; they are only meant to be able to answer questions, such as whether a triple is true or not. It is thus, not trivial on how we can utilize the KGEs models to extract inferred triples. (iii) It is not straightforward on how the two components should communicate as they both require some time to generate information.

We are working towards building a framework which will overcome the above challenges. Our framework consists of two main components: a KGE engine and a reasoner. The KGE engine produces new triples based on the embeddings and the reasoner produces new triples based on the KG, the ontologies and the rules. The rules can be defined by domain experts or specified by different entailment regimes.

*Work done while conducting his master thesis at TU Berlin.

Table 1: Different KGE models. $Real()$ is a function that returns the real part of a complex number, \star denotes a circular correlation operation, and \circ denotes ...

KGE model	Scoring function
TransE [1]	$- e_i + r - e_j _1 \text{ or } 2$
DistMult [16]	$e_i^T \text{diag}(r) e_j$
ComplEx [15]	$Real(e_i^T \text{diag}(r) e_j)$
HolE [10]	$r^T (e_i \star e_j)$
RotatE [14]	$- e_i \circ r - e_j ^2$

The novelty of our framework over the previous hybrid approaches is twofold. First, it leverages the predictive power of the data-driven KGE approaches while at the same time is able to capture domain expert knowledge via deductive reasoning. This allows to infer information even for sparse regions of a KG. Second, it combines KGEs with reasoning in a loosely-coupled way and, thus, does not depend on any specific KGE model or reasoning algorithm: Users can easily input their own KGE or reasoning algorithm and the system will seamlessly work out of the box. Thanks to the loosely-coupled combination of KGEs and reasoning, it can achieve better predictive performance.

2 PRELIMINARIES

Before laying the foundations on KGEs and ontology-based reasoning, let us first formally introduce a knowledge graph.

A knowledge graph \mathcal{K} consists of a set of triples from $\mathcal{E} \times \mathcal{R} \times \mathcal{E}$, where \mathcal{E} is the set of entities and \mathcal{R} the set of relations. A triple $t = (e_i, r, e_j)$ is represented in the graph as two nodes e_i and e_j connected via a directed edge from e_i to e_j with label r . We refer to e_i as the subject and to e_j as the object of t . For example, the triple (Plato, bornIn, Athens) connects the two entities Plato and Athens via the labeled edge bornIn.

2.1 Knowledge graph embeddings

A KGE maps an entity or a relation to a d -dimensional vector (*embedding*). We denote with $e_i \in \mathbb{R}^d$ the embedding of an entity e_i and with $r \in \mathbb{R}^d$ the embedding of a relation r . A KGE model learns a scoring function $f(e_i, r, e_j)$ so that triples (e_i, r, e_j) that exist in \mathcal{K} are generally scored higher than triples that do not exist. The plausibility of a triple (e_i, r, e_j) is then determined by the score output by the function $f(e_i, r, e_j)$.

There have been many proposals on different scoring functions for training KGEs resulting in different models. Table 1 lists the most prominent ones. The objective of training is to optimize the function using some gradient-based algorithm so that it outputs high scores for the triples that exist in the knowledge graph and low scores for the triples that do not exist. The loss function (e.g. logistic or pairwise ranking loss) used for the optimization requires both positive and negative triples. Negative triples are typically constructed by corrupting an existing triple, i.e., replacing its subject or object entities with entities randomly sampled from the graph.

2.2 Ontology-based reasoning

Knowledge graphs are often accompanied by one or more ontologies. An ontology gives formal semantics to the knowledge graph

Table 2: A subset of RDFS rules. Relations **type**, **domain**, **range**, **subProperty**, and, **subClass** are pre-defined in the RDF/S vocabulary.

Rule name	Rule (condition \rightarrow consequence)
rdfs2	$(x, \text{domain}, c), (x, r, y) \rightarrow (x, \text{type}, c)$
rdfs3	$(x, \text{range}, c), (x, r, y) \rightarrow (y, \text{type}, c)$
rdfs5	$(r_1, \text{subProperty}, r_2), (r_2, \text{subProperty}, r_3) \rightarrow (r_1, \text{subProperty}, r_3)$
rdfs7	$(r_1, \text{subProperty}, r_2), (x, r_1, y) \rightarrow (x, r_2, y)$
rdfs9	$(c_1, \text{subClass}, c_2), (x, \text{type}, c_1) \rightarrow (x, \text{type}, c_2)$
rdfs11	$(c_1, \text{subClass}, c_2), (c_2, \text{subClass}, c_3) \rightarrow (c_1, \text{subClass}, c_3)$

by defining the concepts in which entities belong and the relations between these concepts. For example, the entity Plato belongs to the concept Philosopher, while the entity Athens belongs to the concept City, while concepts Philosopher and City are connected via a relation bornIn. Using ontologies not only adds semantics to the knowledge graphs but also enables reasoning via rules.

For example, given the rule

$$\forall x, y \in \mathcal{E} : (x, \text{hasCapital}, y) \rightarrow (y, \text{locatedIn}, x)$$

and the triple (Greece, hasCapital, Athens) we can infer that (Athens, locatedIn, Greece) even if this information is missing from the KG. By adding more rules and triples, such as

$$\forall x, y, z \in \mathcal{E} : (x, \text{bornIn}, y), (y, \text{locatedIn}, z) \rightarrow (x, \text{bornIn}, z)$$

and the triple (Plato, bornIn, Athens), we can infer that (Plato, bornIn, Greece).

Triples such as the latter one are called *intensional* (or implicit) triples, while triples that are already present in the initial KG are called *extensional* (or explicit). Rules can be specified by either a domain expert and/or an entailment regime, e.g., RDFS or OWL2. Table 2 shows the rules of a well-defined fragment [9] of the RDFS entailment regime expressed as Horn clauses.

3 RELATED WORK

Although there is a large body of literature on different KGE models [1, 10, 11, 15, 16] as well as works on scaling KGEs via parallel training [7, 17], there have been only few works on combining KGEs with logic rules. One of the first such work is KALE [3], which embeds first-order logic rules in the same mathematical framework of KGEs by devising a new loss function. In particular, triples are modelled as atomic formulas and are mapped to vectors based on the translation model of TransE [1], while rules are modelled as t-norm fuzzy logics which define the truth value of a complex formula as a composition of the truth values of its constituents. KALE then tries to minimize the global loss function of both the complex and atomic formulas. The problem with this approach is that the embedding and reasoning processes are tailored to a specific KGE scoring function which makes the system difficult to adapt to other more efficient and more scalable KGE models. pLogicNet [12] is based on Markov Logic Network (MLN), which is a rule-based approach utilizing probabilistic graphical models to infer new information and at the same time handle uncertainty. The MLN in pLogicNet is trained with the variational EM algorithm and leverages KGEs to infer missing triples during the inference step (E-step) which are then used in the learning step (M-step) until convergence. However, by relying on MLNs, pLogicNet inherits the inefficiency and

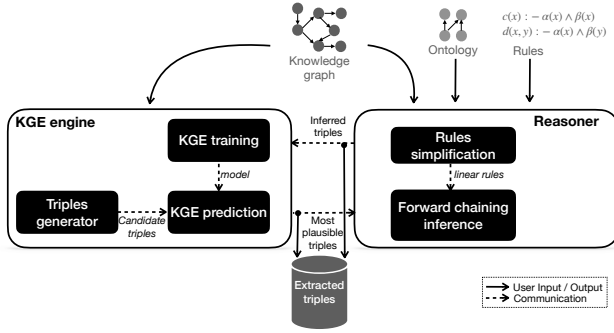


Figure 1: Overview of proposed framework.

inability to scale to large KGs. Finally, the authors of [2] impose two simple constraints in KGE models: (i) vectors of entities should be positive values meaning that we need only positive statements about entities and (ii) approximate rules between two relations, e.g., *bornIn* relation implies *nationality* relation, which they mine from the data. These constraints are then embedded into the score function, e.g., of ComplEx, and the KGE model is learned. However, this work is coupled with a specific KGE model and the used constraints are very restrictive being unable to incorporate domain knowledge.

4 UNIFIED FRAMEWORK

Our goal is to combine KGEs with ontology-based reasoning in a loosely-coupled way so that we can infer missing information, i.e., extract triples that do not exist in a KG. Our framework consists of two main components: the KGE engine and the reasoner. Figure 1 shows the internals of the two components and how they interact with each other. Each component passes its output (i.e., inferred triples) as input to the other component in an iterative manner until no new information can be inferred.

4.1 KGE engine

The KGE engine is responsible for the embedding-based learning. It receives as input the initial KG together with any triples inferred by the reasoner and trains a KGE model. The model is then able to predict with certain probability whether a given test triple, not contained in the training input set, is true or not. The challenge we had to overcome is how to generate the test triples. Ideally, we would like to pass through the KGE prediction the complement of the input KG, i.e., all triples that do not exist. Although theoretically it is possible to create the complement of the input KG by taking all pairs of nodes that are not directly connected and all possible edge labels, it is infeasible in practice as the number of test triples would be $O(N^2 \times R)$ for a KG with N entities and R relations. For this reason, we build the triples generator module inside the KGE engine that is responsible to output a reduced set of candidate triples that have a good chance of being true.

The triples generator is built in a generic manner to be able to support various methods for generating candidate triples. The simplest method is using uniform random sampling to draw from the

set of entities and relations. This strategy, however, leads to triples that are with high probability not true as connecting randomly entities with random relations creates meaningless triples. An intuitive strategy is to explore sparse regions of the graph as entities that are densely connected are less probable to have missing true relations. For this we exploit the cluster coefficient of the entities (nodes in the graph), i.e., the fraction of triangles passing through a node w.r.t. its degree. We then use the cluster coefficient as a weight to sample entities for each type of relation in KG. Finally, the triples generator makes sure that the produced triples are not included in the training set used to build the model.

Once the triples generator module outputs candidates triples, it passes them to the KGE prediction module which outputs whether a triple is true with a certain probability. The KGE engine outputs the triples that are true with a probability above a certain threshold.

The goal of the KGE engine is to be independent of any KGE model and triples generation strategy. In other words, users can plug their own implementations. To do so, our framework exposes the following primitives:

$$\text{model} = \text{fit}(X)$$

$$Y = \text{generateTriples}(X)$$

$$Y' = \text{predict}(Y, \text{model})$$

Function *fit* receives an array X with the input triples and returns the fitted KGE model. Function *generateTriples* takes as input an array X of triples and outputs another array of candidate triples Y which are not contained in X . Function *predict* receives an array of triples Y and a model and outputs the input triples with an extra column that specifies the probability of the triple being true. Our implementation currently includes the following KGE models: TransE, HolE, ComplEx, and DistMult.

4.2 Inference engine

The inference engine is responsible to infer new triples given (i) an initial set of triples (e.g., initial KG plus triples output by the KGE engine), (ii) an ontology, and (iii) a set of rules. Rules can be user-defined or can be coming from an entailment regime, such as RDFS or OWL2. Our framework can use any forward chaining reasoner that exhaustively applies the given rules to the input and inferred triples until a fixpoint is reached, i.e., no new triples can be inferred. This can be easily achieved by adding the following simple primitive as a layer on top of the reasoner: $Y = \text{infer}(X, \text{ontology}, \text{rules})$. The input is the input triples X , the ontology and the rules.

The type of rules supported strongly depends on the underlying reasoner used by the framework. The only requirement is that they should be monotonic, i.e., adding new rules or triples never results in removing or contradicting already output results. In our current implementation, we use Pellet [13] and Hermit¹ OWL2 DL reasoners.

A problem that may arise with the input rules is that they are redundant, i.e., they produce redundant triples. Although this does not change the correctness of the results, it adds a significant overhead on the runtime of the reasoning process. Even the rules included in the RDFS entailment regime are redundant and can lead to a large number of redundant triples [6]. For this reason, we have built a

¹<http://www.hermit-reasoner.com/>

Dataset	Entities	Relations	Triples (total)
DBPedia20k	20,143	12	120,000
LUBM-2	53,860	32	268,136

Table 3: Datasets statistics.

component that simplifies the input rules. For example, this module can transform a bilinear (rule `rdfs11` Table 2) to a linear rule.

5 VALIDATION

We now evaluate our proposal by using both synthetic and real world KGs. The questions we want to answer is (i) how link prediction can be improved by incorporating reasoning with KGEs in a decoupled manner and (ii) what is the training time overhead that reasoning incurs to the KGE computation.

Datasets. We use datasets one real-world KG that is accompanied with ontology, namely DBPedia, and the popular synthetic KG generator, LUBM. As the original real KG is too large for KGE models to handle, we use a subset of it: we extracted a well-connected subgraph that contains the 12 most popular relations and information around them. Table 3 summarizes the characteristics of our datasets.

Baselines. We compare our proposal against the following vanilla KGE models: TransE, DistMult, ComplEx, HolE, and ConvE. In addition, we compare against KALE [3] and plogNet [12] which use a different approach to incorporate reasoning in KGEs. For each dataset and each KGE model, we use the same hyperparameter settings for the vanilla KGEs and our approach. For KALE and plogNet, we use the hyperparameters recommended in their repository.

Hardware and Software. We ran all our experiments in a machine with 32x2.3 GHz AMD Opteron(tm) processor 6376, 62GB RAM memory, a GPU NVIDIA-SMI 440.33.01, the driver version 440.33.01 and the CUDA version 10.0. We used Python 3.7 and Tensorflow 1.15. For the baselines plogNet and KALE, we used Python 3.7 with PyTorch 1.5 and Java 1.8.0, respectively.

Metrics. We evaluate the quality of each method by computing the filtered mean reciprocal rank (MRR), and the Hits@1, Hits@3, and Hits@10 for the link prediction task. These are the standard metrics used to compare different KGE methods. In addition, we measure training time.

5.1 Model performance

We first compare the model performance our method achieves compared to vanilla KGE models. Table 4 shows the results for both datasets. We observe that for both LUBM and DBPedia20k our approach almost always improves the quality of the model (shown by the underlined times), by up to 3x in MRR for the case of HolE in the LUBM dataset. In particular, we improve the MRR, Hits@1, Hits@3, and Hits@10 of HolE [10] by a factor of 3. The best model for LUBM scored an MRR of 0.35 with DistMult* which includes the DistMult [16] KGE model with our reasoner. For DBpedia20k, ComplEx* scored the best MRR with value of 0.29. As the best KGE

Table 4: Quality results among different approaches. Our approach (denoted by an asterisk to the used KGE model) performs better than vanilla KGEs for both LUBM and DBPedia20k. It also significantly outperforms the hybrid approaches of plogNet [12] and KALE [3].

LUBM				
Method	MRR	Hits@1	Hits@3	Hits@10
TransE	0.28	0.28	0.26	0.31
TransE*	<u>0.24</u>	<u>0.22</u>	<u>0.24</u>	<u>0.27</u>
ComplEx	0.24	0.22	0.24	0.27
ComplEx*	<u>0.32</u>	<u>0.27</u>	<u>0.32</u>	<u>0.41</u>
DistMult	0.28	0.26	0.28	0.32
DistMult*	0.35	0.32	0.36	0.40
HolE	0.09	0.08	0.09	0.10
HolE*	<u>0.27</u>	<u>0.25</u>	<u>0.27</u>	<u>0.29</u>
plogNet	0.10	0.09	0.11	0.14
KALE	0.01	0	0	0.05

DBPedia20k				
Method	MRR	Hits@1	Hits@3	Hits@10
TransE	0.11	0.09	0.12	0.15
TransE*	<u>0.14</u>	<u>0.10</u>	<u>0.15</u>	<u>0.20</u>
ComplEx	0.25	0.18	0.27	0.38
ComplEx*	0.29	0.22	0.33	0.43
DistMult	0.25	0.20	0.29	0.34
DistMult*	<u>0.28</u>	0.20	<u>0.33</u>	<u>0.41</u>
HolE	0.13	0.11	0.14	0.17
HolE*	0.13	0.11	0.14	<u>0.18</u>
plogNet	0.20	0.16	0.24	0.31
KALE	0.18	0	0	0.25

model differs for each dataset, it is fundamental to give the opportunity to users to use any KGE model. This is achieved through our decoupled framework.

In addition, we observe that our approach significantly outperforms the other hybrid baselines, namely plogNet [12] and KALE [3]. Our approach achieves up to 3.5x better MRR performance than plogNet and 1.5x better MRR than KALE. This is not only because plogNet and KALE deeply embed reasoning with KGEs and thus may lose some information but also because our approach can effortlessly use any KGE model and improve over it.

5.2 Training time

We now evaluate what is the overhead of adding the reasoning process when training KGEs. Figure 2 shows the training time in hours for (i) different vanilla KGE models compared to their counterpart KGE* and (ii) the baselines compared to the KGE* model that yields the best prediction accuracy. Figure 2(a), we observe that the overhead posed by the reasoning engine is very small for the LUBM dataset (up to 34% for DistMult), while for TransE the inferred triples resulting from the inference lead to faster convergence of the algorithm. Similar results we observe for the DBpedia20k dataset in Figure 2(b).

Finally, Figure 2(c) shows the difference in training time between KGE* and the hybrid baselines KALE and plogNet. For both datasets, KGE* is faster to converge than the baselines that need significant much more time.

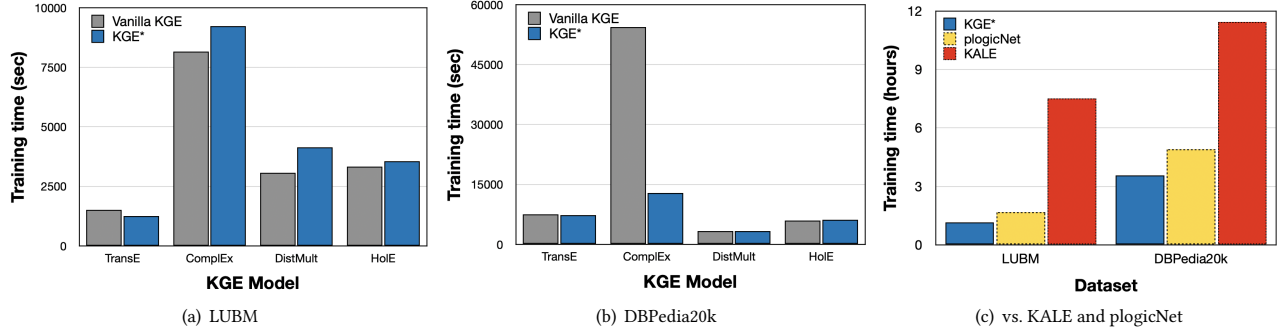


Figure 2: (a) and (b): Training times for vanilla KGE and improved with reasoning KGE (KGE*) for different datasets: KGE* keeps the reasoning overhead low for most of the cases. Adding reasoning may result to faster convergence and thus lower training times (see e.g., ComplEx for DBpedia). (c): KGE* is much faster than plogiNet and KALE for both LUBM and DBPedia20k datasets.

6 CONCLUSION

We proposed a hybrid framework that loosely decouples KGEs with ontology-based reasoning. In contrast to state-of-the-art approaches that rely solely on the information available in the input knowledge graph, we can incorporate domain expertise in the KG completion task which leads to increased accuracy. The loose decoupling also allows users to plug any KGE or reasoning algorithm they like allowing for further optimizations. Our results showed that our proposed framework can improve the accuracy of prediction by up to 3x while keeping the training time low.

REFERENCES

- [1] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS '13)*. 2787–2795.
- [2] Boyang Ding, Quan Wang, Bin Wang, and Li Guo. 2018. Improving Knowledge Graph Embedding Using Simple Constraints. In *ACL*.
- [3] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. 2016. Jointly Embedding Knowledge Graphs and Logical Rules. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 192–202.
- [4] Victor Gutierrez-Basulto and Steven Schockaet. 2018. From Knowledge Graph Embedding to Ontology Embedding? An Analysis of the Compatibility between Vector Space Representations and Rules. In *Knowledge Representation and Reasoning (KR)*. 379–388.
- [5] Nitisha Jain, Trung-Kien Tran, Mohamed H. Gad-Elrab, and Daria Stepanova. 2021. Improving Knowledge Graph Embeddings with Ontological Reasoning. In *The Semantic Web – ISWC 2021*. 379–388.
- [6] Zoi Kaoudi and Manolis Koubarakis. 2013. Distributed RDFS Reasoning over Structured Overlay Networks. *Journal of Data Semantics* (2013).
- [7] Adrian Kochsiek and Rainer Gemulla. 2021. Parallel Training of Knowledge Graph Embedding Models: A Comparison of Techniques. *PVLDB* (2021).
- [8] Aisha Mohamed, Shameem Puthiya Parambath, Zoi Kaoudi, and Ashraf Abou-naga. 2020. Popularity Agnostic Evaluation of Knowledge Graph Embeddings. In *UAI*. 1059–1068.
- [9] Sergio Muñoz, Jorge Pérez, and Claudio Gutierrez. 2009. Simple and efficient minimal RDFS. *Journal of Web Semantics* (2009).
- [10] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. 2016. Holographic Embeddings of Knowledge Graphs. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI '16)*. 1955–1961.
- [11] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A Three-way Model for Collective Learning on Multi-relational Data. In *Proceedings of the 28th International Conference on Machine Learning (ICML '11)*. 809–816.
- [12] Meng Qu and Jian Tang. 2019. Probabilistic Logic Neural Networks for Reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*. 7710–7720.

- [13] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. 2007. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics* (2007).
- [14] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *ICLR (Poster)*.
- [15] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *Proceedings of the 33rd International Conference on Machine Learning (ICML '16)*. 2071–2080.
- [16] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR '15)*.
- [17] Denghui Zhang, Manling Li, Yantao Jia, Yuanzhuo Wang, and Xueqi Cheng. 2017. Efficient Parallel Translating Embedding For Knowledge Graphs. In *Proceedings of the International Conference on Web Intelligence*. 460–468.