Knowledge Graph Embedding for Link Prediction: A Comparative Analysis

ANDREA ROSSI, Department of Engineering, Roma Tre University
DENILSON BARBOSA, Department of Computing Science, University of Alberta
DONATELLA FIRMANI, Department of Engineering, Roma Tre University
ANTONIO MATINATA, Department of Engineering, Roma Tre University
PAOLO MERIALDO, Department of Engineering, Roma Tre University

Knowledge Graphs (KGs) have found many applications in industrial and in academic settings, which in turn, have motivated considerable research efforts towards large-scale information extraction from a variety of sources. Despite such efforts, it is well known that even the largest KGs suffer from incompleteness; Link Prediction (LP) techniques address this issue by identifying missing facts among entities already in the KG. Among the recent LP techniques, those based on *KG embeddings* have achieved very promising performance in some benchmarks. Despite the fast-growing literature on the subject, insufficient attention has been paid to the effect of the design choices in those methods. Moreover, the standard practice in this area is to report accuracy by aggregating over a large number of test facts in which some entities are vastly more represented than others; this allows LP methods to exhibit good results by just attending to structural properties that include such entities, while ignoring the remaining majority of the KG. This analysis provides a comprehensive comparison of embedding-based LP methods, extending the dimensions of analysis beyond what is commonly available in the literature. We experimentally compare the effectiveness and efficiency of 18 state-of-the-art methods, consider a rule-based baseline, and report detailed analysis over the most popular benchmarks in the literature.

CCS Concepts: • **Computing methodologies** → **Knowledge representation and reasoning**; • **Information systems** → **Information integration**.

Additional Key Words and Phrases: Knowledge Graphs, Link Prediction, Knowledge Graph Embeddings, Comparative Analysis

ACM Reference Format:
Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. 2020. Knowledge Graph Embedding for Link Prediction: A Comparative Analysis. *ACM Trans. Knowl. Discov. Data.* 1, 1, Article 1 (January 2020), 47 pages. https://doi.org/10.1145/3424672

1 INTRODUCTION

Knowledge Graphs (KGs) are structured representations of real world information. In a KG *nodes* represent entities, such as people and places; *labels* are types of relations that can connect them; *edges* are specific facts connecting two entities with a relation. Due to their capability to model structured data in a machine-readable way, KGs are used in countless

Authors' addresses: Andrea Rossi, Department of Engineering, Roma Tre University, Via della Vasca Navale, 79, Rome, Italy, 00146, andrea.rossi3@uniroma3.it; Denilson Barbosa, Department of Computing Science, University of Alberta, 2-21 Athabasca Hall, Edmonton, Alberta, AB T6G 2E8, denilson@ualberta.ca; Donatella Firmani, Department of Engineering, Roma Tre University, Via della Vasca Navale, 79, Rome, Italy, 00146, donatella.firmani@uniroma3.it; Antonio Matinata, Department of Engineering, Roma Tre University, Via della Vasca Navale, 79, Rome, Italy, 00146, ant.matinata@stud.uniroma3.it; Paolo Merialdo, Department of Engineering, Roma Tre University, Via della Vasca Navale, 79, Rome, Italy, 00146, paolo.merialdo@uniroma3.it.

domains, ranging from question answering to content-based recommendation, and they are vital to any semantic web project [26]. Notable examples are FreeBase [5], WikiData [70], DBPedia [3], Yago [58] and – in industry – Google KG, Microsoft Satori and Facebook Graph Search. These massive KGs can contain millions of entities and billions of facts.

Despite such volumes of data, it is well known that even the richest KGs suffer from *incompleteness*: for instance, in FreeBase over 70% of person entities have no known place of birth, and over 99% have no known ethnicity [11, 76]. This has led researchers to investigate how to identify new facts to add to KGs [53]. This task, commonly known as *Knowledge Graph Completion* or *Knowledge Graph Augmentation*, can be performed by either extracting new facts from external sources, such as Web corpora, or by inferring them from those already in the KG. The latter approach, called *Link Prediction* (LP), is the focus of our analysis.

LP has been an increasingly active area of research, and it has recently benefited from the explosion of machine learning and deep learning techniques. The vast majority of LP models nowadays leverage the original KG elements to learn low-dimensional representations dubbed *Knowledge Graph Embeddings*, that are subsequently used to infer new facts. Inspired by a few seminal works such as RESCAL [50] and TransE [6], in just a few years researchers have developed dozens of models based on very different architectures.

One aspect that is common to the vast majority of papers in this area, but nevertheless also problematic, is that they report results aggregated over a large number of test facts in which few entities are over-represented: LP methods can report good results on these benchmarks by just attending to such entities, while ignoring the others. Moreover, the limitations of the current best-practice can make it difficult to understand how the papers in this literature fit together, and to picture what research directions are worth pursuing. In other words the strengths, weaknesses and limitations of the current techniques are still unknown and the circumstances allowing models to perform better have been hardly investigated. Roughly speaking, we still do not really know what makes a fact easy or hard to learn and predict.

In order to mitigate these issues we carry out an extensive comparative analysis on a representative set of LP models based on KG embeddings. We specifically focus on models that only leverage the KG structure to learn their embeddings; we privilege state-of-the-art systems, and consider works belonging to a wide range of architectures. We train and tune such systems from scratch and provide experimental results beyond what is available in the original papers, by proposing new and informative evaluation practices. Specifically:

- We take into account 18 embedding-based models belonging to diverse architectures, and adopt as a baseline an additional rule-based state-of-the-art system. We provide a detailed description of the approaches considered for experimental comparison and a summary of related literature, together with a taxonomy for Knowledge Graph Embedding techniques.
- We consider the 5 most commonly employed datasets as well as the most popular metrics currently used for benchmarking; we analyze in detail their features and peculiarities.
- For each model, we provide quantitative results for efficiency and effectiveness on every dataset.
- We define a set of structural features in the training data, and we measure how they affect the predictive performance of each model on each test fact.

We find that, across all datasets, models strongly respond to the presence of specific graph features in the training set. Their performance seems especially sensitive to prediction peers (Section 4.3.1), a novel graph structure introduced in this work, and to the logical support provided by relation paths, that we estimate with an original metric (Section 4.3.2). Furthermore, we observe and investigate how different models can display very different behaviours when dealing with specific relation types. We also find that, in some datasets, the processes applied to represent relations with cardinality

beyond 2 has largely altered the graph structure and its semantics, also affecting the performance of models. All in all, our findings indicate which scenarios are easy to handle for LP models, and which ones are still in need for further advancements. We finally show that, due to low-level methodological differences, the results published for some models are not directly comparable to the others; we explain how such policies affect the evaluation process, and we identify the most responsible architectural components.

The datasets, the code and all the resources used in our work are publicly available through our GitHub repository.[1]

Outline. The paper is organized as follows. Section 2 provides background on KG embeddings and LP. Section 3 introduces the models included in our work, presenting them in a taxonomy to facilitate their description. Section 4 describes the analysis directions and approaches we follow in our work. Section 5 reports our results and observations. Section 6 provides lessons learned and future research directions. Section 7 discusses related works, and Section 8 provides concluding remarks.

2 THE LINK PREDICTION PROBLEM

This section provides a detailed outline of the LP task on KGs, introducing key concepts that we refer to in our work.

We define a KG as a labeled, directed multi-graph, that is, a graph in which multiple differently labeled edges can connect the same nodes: $KG = (\mathcal{E}, \mathcal{R}, \mathcal{G})$:

- $\mathcal{E}$: a set of nodes representing *entities*;
- $\mathcal{R}$: a set of labels representing *relations*;
- $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$: a set of edges representing *facts* connecting pairs of entities. Each fact is a triple $\langle h, r, t \rangle$, where h is the *head*, r is the *relation*, and t is the *tail*.

Link Prediction (LP) is the task of exploiting the existing facts in a KG to infer missing ones. This amounts to guessing the correct entity that completes $\langle h, r, ? \rangle$ in tail prediction, or $\langle ?, r, t \rangle$ in head prediction. More generally, in any prediction we call the known entity *source* entity, and the one to predict *target* entity.

In time, numerous approaches to LP have been studied. Some are based on observable features, relying on techniques such as Rule Mining [1, 17, 18, 41] or the Path Ranking Algorithm [34, 35]. With the rise of Machine Learning, researchers have recently started experimenting on capturing latent features of the graph by learning vectorized representations of its elements, dubbed KG embeddings.

In general, *embeddings* are numerical vectors that can be used to represent any kind of element (e.g., depending on the domain: words, people, products...). They are learned automatically, based on how the corresponding elements occur and interact with each other in datasets representative of the real world. For instance, word embeddings have become a standard way to represent words based on their co-occurrence in text. KG embeddings are used to represent entities and relations in a KG; they embody the semantics of the original graph, and can thus be used to identify new links in it. In the following we indicate KG elements in $italics$ and the corresponding embeddings in $\boldsymbol{bold}$, e.g we may use e to refer to to a generic entity in the graph, and $\boldsymbol{e}$ to indicate its embedding. When indicating embeddings, we use lowercase letters for monodimensional ones, and uppercase letters when they have 2 or more dimensions.

LP datasets are typically obtained by sampling real-world KGs; each dataset can therefore be seen as a small KG with its own sets of entities $\mathcal{E}$, relations $\mathcal{R}$ and facts $\mathcal{G}$. In order to facilitate research, $\mathcal{G}$ is further split into a training set $\mathcal{G}_{train}$, a validation set $\mathcal{G}_{valid}$ and a test set $\mathcal{G}_{test}$.

[1] https://github.com/merialdo/research.lpca. For each model and dataset, we also share CSV files containing, for each test prediction, the rank and the list of all the entities predicted up to the correct one.

All LP models based on embeddings, in order to to estimate the plausibility of any fact $\langle h,\ r,\ t\rangle$ using the embeddings of its elements, define a scoring function $\phi(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})$. In this paper, unless otherwise specified, we are going to assume that the higher the values of ϕ, the more plausible the facts.

2.1 Loss Functions

In training, embeddings are usually initialized randomly, and then optimized with algorithms such as back-propagation with gradient descent. Models may also learn additional *shared parameters* not specific to any KG element (e.g. weights of neural layers). The training process ultimately aims at finding for embeddings and shared parameters the values that maximize the plausibility of true facts while minimizing it for false facts. Therefore, loss functions need to gather the scores of all facts in $\mathcal{G}_{train}$. Furthermore, such positive samples are often corrupted to generate a set of presumably false triples $\mathcal{G}^{corr}_{train}$, that are included to the loss function too with the goal of minimizing their scores. The combination of positive and negative samples often corresponds to implementing a triplet loss function. Corrupting a fact amounts to replacing its head or tail with a random entity; in time, other policies have been proposed in this regard, based on Bernouilli distribution sampling [74], self-adversarial algorithms [59], or Noise Contrastive Estimation (NCE) [20]. Due to the inherent incompleteness of KGs, however, models need to work under the *Open World Assumption*, and any unseen triples, even the ones obtained with corruption, cannot be considered false with certainty.

In the following we briefly report the most popular loss functions for LP[2]; for a vertical study on this topic see [45]. Before being used in a loss function, scores may need to be normalized, most often through a sigmoid function; some losses enforce this step explicitly in their formulation, e.g. the Self-adversarial Negative Sampling Loss [59].

Negative Log-Likelihood, or *NLL*, is a standard loss and it is a very popular choice:

$$\mathcal{L} = \sum_{\langle h,r,t\rangle \in \mathcal{G}_{train} \cup \mathcal{G}^{corr}_{train}} \log(1 + e^{-y_{hrt}\phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t})}), \tag{1}$$

where $y_{hrt} = +1$ if $\langle h, r, t\rangle \in \mathcal{G}_{train}$, and $y_{hrt} = -1$ if $\langle h, r, t\rangle \in \mathcal{G}^{corr}_{train}$. In this formulation high-scoring positive samples lower $\mathcal{L}$, whereas high-scoring negative ones increase it. Therefore, minimizing $\mathcal{L}$ means maximizing the scores of training facts, while minimizing the others.

Multiclass Negative Log-Likelihood is a NLL version for multi-class classification proposed by Toutanova and Chen [60]. It handles head-corrupted samples $\mathcal{G}^{corr}_{train}[head]$ and tail-corrupted samples $\mathcal{G}^{corr}_{train}[tail]$ in separate addends:

$$\begin{aligned}
\mathcal{L} &= \mathcal{L}_{head} + \mathcal{L}_{tail},\\
\mathcal{L}_{head} &= \sum_{\langle h,r,t\rangle \in \mathcal{G}_{train}} \left(-\phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t}) \ + \ log\left(\sum_{\langle h',r,t\rangle \in \mathcal{G}^{corr}_{train}[head][\langle h,r,t\rangle]} e^{\phi(\boldsymbol{h'},\boldsymbol{r},\boldsymbol{t})} \right) \right),\\
\mathcal{L}_{tail} &= \sum_{\langle h,r,t\rangle \in \mathcal{G}_{train}} \left(-\phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t}) \ + \ log\left(\sum_{\langle h,r,t'\rangle \in \mathcal{G}^{corr}_{train}[tail][\langle h,r,t\rangle]} e^{\phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t'})} \right) \right).
\end{aligned} \tag{2}$$

It has been applied with fruitful results by Kadlec *et al.* [29] and in the Ampligraph project [9]. The remarkable ComplEx implementation with N3 regularization [33] uses a "full" Multiclass NLL: rather than generating a restricted number of negative samples, for each training fact the head and the tail are always replaced with all entities in $\mathcal{E}$.

[2]We report all formulations under the assumption that high scores correspond to better plausibility; this may lead to different plus/minus signs from the original papers. For the sake of simplicity, we do not include regularization terms.

Pairwise Ranking Loss, sometimes called *Margin-Based ranking loss*, is a common choice as well:

$$\mathcal{L} = \sum_{\langle h,r,t\rangle \in \mathcal{G}_{train}} \sum_{\langle h',r,t'\rangle \in \mathcal{G}^{corr}_{train}[\langle h,r,t\rangle]} [\gamma - \phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t}) + \phi(\boldsymbol{h'},\boldsymbol{r},\boldsymbol{t'})]_+, \tag{3}$$

where γ is a margin hyperparameter; $\mathcal{G}^{corr}_{train}[\langle h, r, t\rangle]$ is the set of facts obtained corrupting $\langle h, r, t\rangle$; and $[x]_+ = max(0, x)$. In contrast to NLL, this loss enforces negative samples to just have lesser scores than positive ones, better adhering to the Open World Assumption [71]. Logistic losses seem superior when ϕ is based on tensor decomposition, whereas Pairwise Ranking appears more suitable for geometric transformations [66], although it may be more prone to overfitting [65].

Binary Cross-Entropy, or *BCE*, has recently become a popular choice, especially across models with shared parameters:

$$\mathcal{L} = \sum_{\langle h,r,t\rangle \in \mathcal{G}_{train}} \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} -y_{hre} log(\phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{e})) + (1 - y_{hre}) log(1 - \phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{e})). \tag{4}$$

With BCE, models do not extract a set of negative samples; rather, for each training fact they replace the target with all entities in $\mathcal{E}$, resulting in samples that may be either positive (if included in $\mathcal{G}_{train}$) or negative; y_{hre} will have value +1 in the former case, and 0 for the latter. This can be seen as a way to include in $\mathcal{L}$ all possible negative samples.

Self-adversarial Negative Sampling Loss [59] gives to each negative sample a weight based on the score it obtains:

$$\sum_{\langle h,r,t\rangle \in \mathcal{G}_{train}} \left(-\log(\sigma(\phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t}) - \gamma)) - \sum_{\langle h',r,t'\rangle \in \mathcal{G}^{corr}_{train}[\langle h,r,t\rangle]} weight_{\langle h',r,t'\rangle} \log(\sigma(\gamma - \phi(\boldsymbol{h'},\boldsymbol{r},\boldsymbol{t'}))) \right), \tag{5}$$

where $weight_{\langle h',r,t'\rangle}$ is computed with a softmax across the scores of all negative samples, thus making the highest-scoring negative samples more relevant. This loss relies on a temperature parameter α, and it is a variant of the Negative Sampling Loss used in Word2Vec [44], which gives identical weights to all negative samples.

2.2 Evaluation Metrics

In the prediction phase, given an incomplete triple $\langle h, r, ?\rangle$, the missing tail is inferred as the entity that, completing the triple, results in the highest score:

$$t = \underset{e \in \mathcal{E}}{\operatorname{argmax}}\, \phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{e}). \tag{6}$$

Head prediction is performed analogously.

Evaluation is carried out by performing head and tail prediction on all test triples in $\mathcal{G}_{test}$, and computing every time how the target entity ranks against the others. Ranks can be computed in two largely different settings, called raw and filtered scenarios, depending on how other valid answers outranking the target one are handled. For instance, when predicting the tail for $\langle$ *Barack Obama*, *parent*, *Natasha Obama* $\rangle$, a model may associate a higher score to *Malia Obama* than to *Natasha Obama*. More generally, if a predicted entity produces a fact contained in $\mathcal{G}$, it may still be considered as a valid answer, even if it is different from the target one:

- In *Raw Scenario*, valid entities outscoring the target one are considered as mistakes, and they do contribute to the rank computation. Given a test fact $\langle h,\ r,\ t\rangle \in\ \mathcal{G}_{test}$, the raw rank r_t of the target tail t is computed as:

$$r_t = |\{e \in \mathcal{E} \setminus \{t\}\ :\ \phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{e}) > \phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t})\}| + 1. \tag{7}$$

- In *Filtered Scenario*, valid entities outscoring the target one are not considered as mistakes, and they are skipped in the rank computation. Given a test fact $\langle h,\ r,\ t\rangle \in \mathcal{G}_{test}$, the filtered rank r_t of the target tail t is computed as:

$$r_t = |\{e \in \mathcal{E} \setminus \{t\} \ :\ \phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{e}) > \phi(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t}) \ \wedge\ \langle\ h,\ r,\ e\ \rangle \notin \mathcal{G}\}| + 1. \tag{8}$$

Raw and filtered ranks in head prediction can be computed analogously.

Rank computation also requires defining the strategy to apply when multiple entities obtain the same score as the target one. This event is called a *tie* and it can be handled with different *tie-breaking policies*:

- *min*: the target is given the lowest (best) rank among the entities in the tie. This is the most permissive policy, and it may result in artificially boosting benchmark results, as we show with our experiments in Section 5.5.
- *average*: the target is given the average rank among the entities in the tie.
- *random*: the target is given a random rank among the entities in the tie. On large test sets, this should globally be equivalent to the average policy.
- *ordinal*: the entities in the tie are given ranks based on the order in which they are seen, which should be independent from their scores: therefore this should globally correspond to the random policy.
- *max*: the target is given the highest (worst) rank among the entities in the tie. This is the most strict policy.

The ranks Q obtained from test predictions are usually employed to compute standard global metrics. The most common metrics in LP are:

Mean Rank (MR). It is the average of the obtained ranks:

$$MR = \frac{1}{|Q|}\sum_{q\in Q} q. \tag{9}$$

It is always between 1 and $|\mathcal{E}|$; the lower it is, the better the result. As noted by Nickel *et al.* [52] it is very sensitive to outliers (e.g. spurious overranked facts), so it is progressively being dismissed in favour of Mean Reciprocal Rank.

Mean Reciprocal Rank (MRR). It is a popular metric in Information Retrieval, computing the average of the inverse of the obtained ranks [69]:

$$MRR = \frac{1}{|Q|}\sum_{q\in Q} \frac{1}{q}. \tag{10}$$

It is always between 0 and 1; the higher it is, the better the result.

Hits@K (H@K). It is the ratio of predictions for which the rank is equal or lesser than a threshold K [40]:

$$H@K = \frac{|\{q \in Q : q \leq K\}|}{|Q|}. \tag{11}$$

It is always between 0 and 1; the higher it is, the better the result. Common values for K are 1, 3, 5, 10; when $K = 1$, H@K is the ratio of test facts where the target is predicted on the first try. H@1 and MRR are often closely related, because such predictions correspond to the most relevant addends in the MRR formula.

All the aforementioned metrics can be computed either separately for subsets of predictions (e.g. considering separately head and tail predictions) or considering all test predictions altogether.

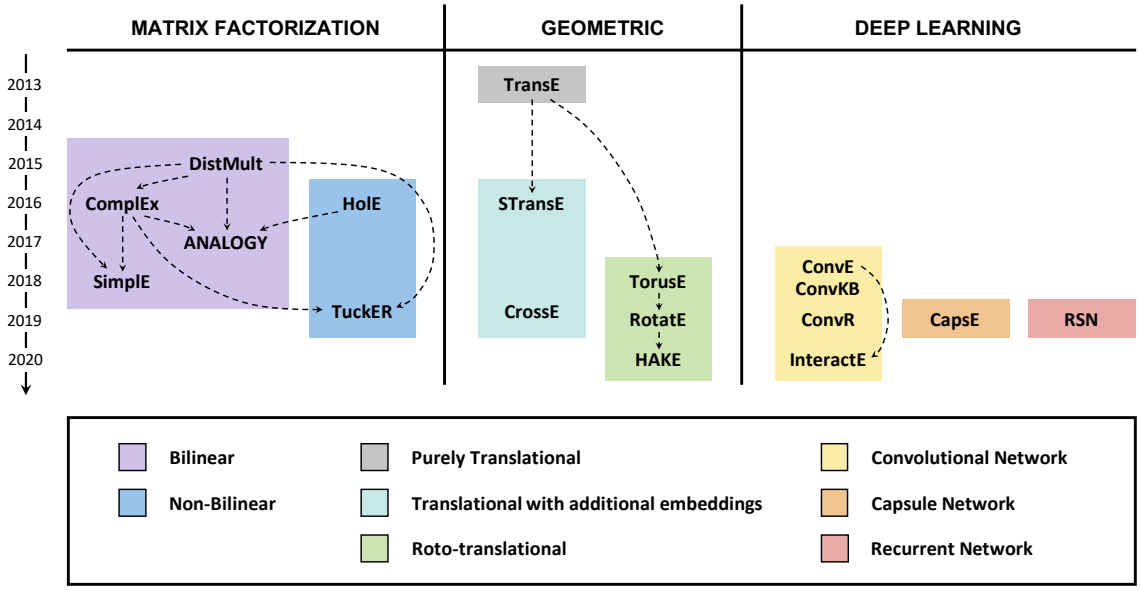


Fig. 1. Taxonomy for the LP models included in our analysis. Dotted arrows indicate that the target method builds on the source method by either generalizing or specializing the definition of its scoring function. The included models are: DistMult [77]; ComplEx [66]; ANALOGY [38]; SimplE [30]; HolE [52]; TuckER [4]; TransE [6]; STransE [47]; CrossE [78]; TorusE [12]; RotatE [59]; HAKE [80]; ConvE [10]; ConvKB [48]; ConvR [28]; InteractE [68]; CapsE [49]; RSN [20].

3 OVERVIEW OF LINK PREDICTION TECHNIQUES

In this section we survey the main embedding-based approaches to LP and discuss the corresponding models included in our work. LP systems can rely on a large variety of methods depending on how they model the optimization problem; in order to overview their highly diverse characteristics we propose a novel taxonomy illustrated in Figure 1.

We define three main families of models; each of them is further split into smaller groups identified by unique colours. For each group, we include in our study the most valid representative models, prioritizing those reaching state-of-the-art performances and, whenever possible, with publicly available implementations. The result is a set of 18 models, that we employ in the experimental sections of our comparative analysis. For each model we also report the influences it has received from the others. We believe that this taxonomy facilitates the understanding of these models and of the experiments carried out in our work. Further information on the included models, e.g. their scoring and loss functions and their space complexity, is reported in Table 1.

In our analysis we focus on the body of literature for systems that learn from the KG structure only. Researchers have been also investigating how to combine these approaches with different sources of information, e.g textual captions [2, 61, 73], or pre-computed rules [21]. The widespread of multi-modal KGs has recently led to systems leveraging an even wider variety of data sources. For instance, MKBE [54] uses separate neural encoders for handling text, images, and numerical values, and combines them with structure-based LP models; the work of Trisedya *et al.* [63] uses textual or numeric attributes of entities to learn attribute character embeddings, and uses them to perform entity alignment across multiple KGs. These systems are not directly comparable to models relying on the KG structure only, so we do not include them in our analysis; see the work of Gesese *et al.* [19] for a survey focusing on those approaches.

We identify three main families of models: 1) *Tensor Decomposition*; 2) *Geometric*; 3) *Deep Learning*.

3.1 Tensor Decomposition Models

These models interpret LP as a task of tensor decomposition [31]. They implicitly see the KG as a 3D adjacency matrix (a 3-way tensor), made only partially observable by incompleteness. This tensor can be decomposed into a combination of low-dimensional vectors, i.e. the embeddings of entities and relations. In practice the embeddings are learnt *a posteriori* by leveraging the known facts only, but they should be able to generalize and associate high scores to unseen true facts as well. The score of a fact can be computed by applying the combination operation on the embeddings of its elements. These models tend to employ few or no shared parameters at all; this makes them particularly light and easy to train.

3.1.1 Bilinear Models. Given the head and tail embeddings $\boldsymbol{h} \in \mathbb{R}^d$, and $\boldsymbol{t} \in \mathbb{R}^d$, these models represent the relation embedding as a bidimensional matrix $\boldsymbol{R} \in \mathbb{R}^{d \times d}$. The scoring function is the bilinear product:

$$\phi(\boldsymbol{h}, \boldsymbol{R}, \boldsymbol{t}) = \boldsymbol{h}^T \ \boldsymbol{R} \ \boldsymbol{t}. \tag{12}$$

These models do not rely on shared parameters, and differ from one another by introducing specific additional constraints on the embeddings. We refer to the following representative models:

DistMult [77] forces all relation embeddings to be diagonal matrices, significantly reducing the space of parameters. As a consequence the scoring function is commutative, with $\phi(\boldsymbol{h}, \boldsymbol{R}, \boldsymbol{t}) = \phi(\boldsymbol{t}, \boldsymbol{R}, \boldsymbol{h})$: this amounts to treating all relations as symmetric. Despite this, when carefully tuned DistMult has been shown to still reach state-of-the-art performance [29].

ComplEx [66] uses the same diagonal constraint as DistMult, but extends its formulation in the complex space: $\boldsymbol{h} \in \mathbb{C}^d$, $\boldsymbol{t} \in \mathbb{C}^d$, $\boldsymbol{R} \in \mathbb{C}^{d \times d}$. The bilinear product thus becomes a Hermitian product, where in lieu of $\boldsymbol{t}$, its conjugate-transpose $\bar{\boldsymbol{t}}$ is used. The Hermitian product is not commutative, enabling ComplEx to successfully model asymmetric relations.

Analogy [38] uses the general bilinear formulation, enforcing constraints that model analogical inductive structures: (*i*) $\boldsymbol{R}$ must be a normal matrix: $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{R}^T\boldsymbol{R}$; (*ii*) the composition of any pair of relations R_1, R_2 must be commutative: $\boldsymbol{R_1} \circ \boldsymbol{R_2} = \boldsymbol{R_2} \circ \boldsymbol{R_1}$. The authors show that normal matrices in Analogy can successfully learn asymmetric relations.

SimplE [30] uses the same diagonal constraint as DistMult. It models each fact in a direct and an inverse form, averaging their bilinear products to compute the overall score. To represent such forms, it embeds each entity e in separate head and tail vectors $\boldsymbol{e_h}$ and $\boldsymbol{e_t}$, and each relation r in separate direct and inverse vectors $\boldsymbol{R}$ and $\boldsymbol{R_{-1}}$. It is fully expressive, and can successfully model asymmetric relations; Lacroix *et al.* simultaneously proposed an analogous model called CP.

3.1.2 Non-bilinear Models. These models combine the head, relation and tail embeddings using formulations different from the strictly bilinear product.

HolE [52] computes the circular correlation (denoted by $\star$ in Table 1) of the head and tail embeddings and multiplies it by the relation embedding; this can be seen as a compression of the full matrix product, making HolE lighter than unconstrained bilinear models. HolE has been proved theoretically equivalent to ComplEx [23]; we include both models in our work, as their different formulations lead to different time complexities [65] and ultimately to different results.

TuckER [4] relies on the Tucker decomposition [24]; it handles entity and relation embeddings of independent dimensions d_e and d_r, and jointly learns a shared core $\boldsymbol{W} \in \mathbb{R}^{d_e \times d_r \times d_e}$. $\boldsymbol{W}$ can be seen as a shared pool of prototype relation matrices. The scoring function is $\phi(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{t}) = \boldsymbol{W} \times_1 \boldsymbol{h} \times_2 \boldsymbol{r} \times_3 \boldsymbol{t}$, where $\times_i$ denotes tensor product along mode i.

This could be seen as a form of bilinear product, with $\boldsymbol{W}$ and $\boldsymbol{r}$ combining in a low-rank matrix $\boldsymbol{R}$ [79]; however $\boldsymbol{W}$ is shared across all KG embeddings, so we do not include TuckER among the strictly bilinear models in our taxonomy.

3.2 Geometric Models

These models interpret relations as geometric operations in the latent space. The head embedding undergoes a spatial transformation τ depending on the values of the relation embedding. The fact score is the distance between the resulting vector and the tail vector, and it is computed using a distance function δ (e.g. L1 or L2 norm).

$$\phi(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \delta(\tau(\boldsymbol{h}, \boldsymbol{r}), \boldsymbol{t}). \tag{13}$$

Depending on τ, the analytical form of geometric models may look similar to a tensor decomposition. In these cases, however, additional constraints are usually needed to make τ implement a spatial transformation; for instance, the rotation operated by RotatE can be formulated as a matrix product, but the rotation matrix would need to be diagonal and contain elements with modulus 1. Much like in Tensor Decomposition, Geometric models tend to avoid shared parameters, running back-propagation directly on the embeddings. We identify three groups in this family: *(i) Pure Translational Models*, *(ii) Translational Models with Additional Embeddings*, and *(iii) Roto-translational models*.

3.2.1 Pure Translational Models. These models interpret each relation as a *translation* in the latent space: the relation embedding is just added to the head embedding, and we expect to land in a position close to the tail embedding. These models thus represent entities and relations as vectors of same length.

TransE [6] was the first geometric LP model, inspired by the translational properties of Word2vec embeddings [44]. It explicitly enforces the tail vector to lie close to the sum of the head and relation ones, measuring the distance with L1 or L2 norm. TransE cannot model correctly one-to-many, many-to-one, symmetric or transitive relations.

3.2.2 Translational models with Additional Embeddings. These models learn more than one embedding for each KG element; they often use relation-specific embeddings for each entity or, vice versa, entity-specific embeddings for each relation. The use of such additional embeddings addresses some of the issues of purely translational methods at the cost of increasing the number of parameters; as noted by Kazemi and Poole [30], it does not imply full expressiveness.

STransE [47], in addition to the d-dimensional embeddings seen in TransE, maps each relation r to two additional $d \times d$ independent matrices $\boldsymbol{W}_r^{\boldsymbol{h}}$ and $\boldsymbol{W}_r^{\boldsymbol{t}}$. When computing the score of a fact $\langle h, r, t \rangle$, before the usual translation, $\boldsymbol{h}$ is pre-multiplied by $\boldsymbol{W}_r^{\boldsymbol{h}}$ and $\boldsymbol{t}$ by $\boldsymbol{W}_r^{\boldsymbol{t}}$; this amounts to using relation-specific embeddings for the head and tail, alleviating the issues with one-to-many and many-to-one relations.

CrossE [78] learns for each relation r an additional embedding $\boldsymbol{c_r}$ of same size. When scoring $\langle h, r, t \rangle$, it combines $\boldsymbol{h}$ and $\boldsymbol{r}$ with $\boldsymbol{c_r}$ using element-wise products (denoted by $\odot$ in Table 1). The resulting triple-specific interaction embeddings are then used for the translation. Similarly to deep learning models and TuckER, CrossE interposes operations with non-linear activation functions, such as *hyperbolic tangent* and *sigmoid* (denoted respectively by tanh and σ in Table 1).

3.2.3 Roto-Translational Models. These models include operations that are not directly expressible as pure translations: this often amounts to performing rotation-like transformations either in combination or instead of translations.

TorusE [12] was motivated by the observation that regularization in TransE, forcing entity embeddings to lie on a hypersphere, limits their capability to satisfy the translational constraint. To solve this, TorusE projects each point $\boldsymbol{x}$ of

the original space $\mathbb{R}^d$ into a point $[\boldsymbol{x}]$ on a torus $\mathbb{T}^d$. The authors define torus distances d_{L1}, d_{L2} and d_{eL2} corresponding to L1, L2 and squared L2 norm respectively (in Table 1 we report the scoring function with the extended form of d_{L1}).

RotatE [59] models relations as rotations in a complex latent space, with $\boldsymbol{h}$, $\boldsymbol{r}$ and $\boldsymbol{t}$ all belonging to $\mathbb{C}^d$. The $\boldsymbol{r}$ embedding is a rotation vector: in all its elements, the phase conveys the rotation along that axis, and the modulus is equal to 1. $\boldsymbol{h}$ is rotated by $\boldsymbol{r}$ through an element-wise product, and the distance from $\boldsymbol{t}$ is measured with the L1 norm. Rotation allows to model correctly numerous relational patterns, such as symmetry/anti-symmetry, inversion and composition.

HAKE [80] extends RotatE by having relations combine modulus scaling with rotation. HAKE maps each KG element e to separate modulus and phase vectors $\boldsymbol{e_m}$ and $\boldsymbol{e_p}$. When scoring $\langle h, r, t\rangle$, modulus $\boldsymbol{h_m}$ is scaled via dot product with $\boldsymbol{r_m}$, while phase $\boldsymbol{h_p}$ is added $\boldsymbol{r_p}$. Distances from $\boldsymbol{t_m}$ and $\boldsymbol{t_p}$ are L1 norm and *sin* respectively. The authors show that in hierarchical relations the modulus scaling prevails, whereas nonhierarchical relations just differ for their phases.

3.3 Deep Learning Models

Deep Learning Models use deep neural networks to perform LP. Neural Networks learn parameters such as weights and biases, and combine them with the input data to recognize significant patterns. Deep networks organize parameters into separate layers, generally interspersed with non-linear activation functions. In time, numerous types of layers have been developed; dense layers, for instance, will just combine the input data $\boldsymbol{X}$ with weights $\boldsymbol{W}$ and add a bias $\boldsymbol{b}$: $\boldsymbol{W} \times \boldsymbol{X} + \boldsymbol{b}$. For the sake of simplicity, in the following formulas we will not mention the use of bias, keeping it implicit. More advanced layers perform more complex operations, such as convolutional layers, that learn convolution kernels to apply to the input data, or recurrent layers, that handle sequential inputs in a recursive fashion.

In the LP field, KG embeddings are usually learned jointly with the weights and biases of the layers; these shared parameters make these models more expressive, but potentially longer to train and more prone to overfitting. We identify three groups in this family, based on the neural architecture they employ: *(i) Convolutional Neural Networks*, *(ii) Capsule Neural Networks*, and *(iii) Recurrent Neural Networks*.

3.3.1 Convolutional Neural Networks. These models rely on convolutional layers [36] to perform convolution on the input data (e.g. the embeddings of the KG elements in a training fact) applying low-dimensional filters $\boldsymbol{\omega}$. The result is a *feature map* that is usually then passed to additional dense layers in order to compute the fact score.

ConvE [10] represents entities and relations as one-dimensional d-sized vectors. When scoring $\langle h, r, t\rangle$, it concatenates and reshapes $\boldsymbol{h}$ and $\boldsymbol{r}$ into a unique input $[\boldsymbol{h}; \boldsymbol{r}]$, with dimensions $d_m \times d_n$. This input is passed through a convolutional layer with a set $\boldsymbol{\omega}$ of $m_\omega \times n_\omega$ filters, and then through a dense layer with d neurons and a set of weights $\boldsymbol{W}$. The output is finally combined with the tail embedding $\boldsymbol{t}$ using the dot product, resulting in the fact score.

ConvKB [48] models entities and relations as same-sized vectors as well; when scoring $\langle h, r, t\rangle$ it concatenates $\boldsymbol{h}$, $\boldsymbol{r}$ and $\boldsymbol{t}$ into a $d \times 3$ matrix $[\boldsymbol{h}; \boldsymbol{r}; \boldsymbol{t}]$. This input undergoes convolution by a set $\boldsymbol{\omega}$ of T filters of shape 1×3, resulting in a $T \times 3$ feature map. The feature map goes through a dense layer with one neuron and weights $\boldsymbol{W}$, resulting in the fact score.

ConvR [28] learns entity and relation embeddings as vectors of different dimensions d_e and d_r. When scoring $\langle h, r, t\rangle$, $\boldsymbol{h}$ is reshaped into a $d_{e_m} \times d_{e_n}$ matrix, and $\boldsymbol{r}$ is split into a set $\boldsymbol{\omega_r}$ of T $m_\omega \times n_\omega$ convolutional filters, that are then applied on $\boldsymbol{h}$. This amounts to an adaptive convolution with relation-specific filters. The resulting feature maps are passed to a dense layer with weights $\boldsymbol{W}$; as in ConvE, the score is computed as the dot product between the neural output and $\boldsymbol{t}$.

InteractE [68] expands and improves ConvE. When scoring $\langle h, r, t \rangle$, it generates multiple permutations p_i of $\boldsymbol{h}$ and $\boldsymbol{r}$, and concatenates them in couples $[\boldsymbol{h}_i; \boldsymbol{r}_i]$. The authors investigate how various reshaping practices highlight homogeneous and heterogeneous interactions between $\boldsymbol{h}$ and $\boldsymbol{t}$, and ultimately choose a chequered reshaping. InteractE stacks the reshaped matrices into a 3D tensor, that is then processed with depthwise circular convolution.

3.3.2 Capsule Neural Networks. Capsule networks [57] (CapsNets) are composed of groups of neurons called *capsules* that encode specific features of the input, e.g. the presence of a specific object in an image. CapsNets recognize such features without losing spatial information the way convolutional networks do. Each capsule sends its output to higher order ones, with connections decided by a dynamic routing process.

CapsE [49] embeds entities and relations into same-sized vectors, assuming that different embeddings encode homologous aspects in the same positions. It relies on a similar architecture to ConvKB, but replaces the final dense layer with a capsule layer in which a separate capsule is passed information regarding one aspect of the input fact. A second layer with one capsule is used to yield the triple score. In Table 1, we denote capsule layer operations with *capsnet*.

3.3.3 Recurrent Neural Networks (RNNs). These models employ one or multiple recurrent layers [25] to analyze entire paths (sequences of facts) extracted from the training set, instead of just processing individual facts separately.

Recurring Skipping Networks (RSNs) [20] are a type of RNNs specifically designed for LP. The authors observe that traditional RNNs may be unsuitable for LP due to the alternation of entities and relations, and propose a new RSN layer that actively takes this aspect into account. The resulting RSN model learns entire paths extracted from the training set with biased random walks, and uses a specially optimized type-based noise contrastive estimation loss. In Table 1 we dub the RSN operation as *rsn*; the layers in a cell as L; the weight matrices as k; the neurons in a layer as n.

4 METHODOLOGY

In this section we describe the procedures we employ to study the efficiency and effectiveness of the models mentioned in Section 3, as well as to investigate their behaviours.

4.1 Datasets

LP datasets are usually obtained by sampling real-world KGs and partitioning the extracted facts into training, validation and test sets. We run our analysis on the 5 best-established LP datasets; their core properties can be found in Table 2.

FB15k was generated by Bordes *et al.* [6] extracting all the FreeBase entities with more than 100 mentions and also featured in the Wikilinks database[3], as well as all the facts featuring them. The authors excluded literals (e.g. dates, proper nouns, etc), and converted all *Compound Value Type* (CVT) nodes conveying reified n-ary relations into cliques of binary edges; this operation has greatly affected the graph structure and semantics, as described in Section 4.3.4.

WN18, also introduced by Bordes *et al.* [6], was extracted from WordNet[4], a linguistic KG ontology meant to provide a dictionary/thesaurus to support NLP and automatic text analysis. In WordNet entities correspond to *synsets* (word senses) and relations represent their lexical connections (e.g. "hypernym"). In order to build WN18, the authors used WordNet as a starting point, and then iteratively filtered out entities and relationships with too few mentions.

[3] https://code.google.com/archive/p/wiki-links/
[4] https://wordnet.princeton.edu/

		Loss	Score	Constraints	Space Complexity
Tensor Decomposition models	**DistMult**	Pairwise Ranking	$\boldsymbol{h}^\top\ \boldsymbol{R}\ \boldsymbol{t}$	$\forall\ r \in \mathcal{R} : \boldsymbol{R}$ is diagonal	$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
	ComplEx	Negative Log-Likelihood	$\boldsymbol{h}^\top\ \boldsymbol{R}\ \overline{\boldsymbol{t}}$	$\boldsymbol{h} \in \mathbb{C}^d;\ \boldsymbol{R} \in \mathbb{C}^{d \times d};\ \boldsymbol{t} \in \mathbb{C}^d$ $\forall\ r \in \mathcal{R} : \boldsymbol{R}$ is diagonal	$\mathcal{O}(2\lvert\mathcal{E}\rvert d + 2\lvert\mathcal{R}\rvert d)$
	Analogy	Logistic (NLL variant)	$\boldsymbol{h}^\top\ \boldsymbol{R}\ \boldsymbol{t}$	$\forall\ r \in \mathcal{R} : \boldsymbol{R}\ \boldsymbol{R}^\top = \boldsymbol{R}^\top\ \boldsymbol{R}$ $\forall\ (r_1, r_2) \in \mathcal{R} \times \mathcal{R} : \boldsymbol{R}_1\ \boldsymbol{R}_2 = \boldsymbol{R}_2\ \boldsymbol{R}_1$	$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
	SimplE	Negative Log-Likelihood	$\frac{1}{2}(\boldsymbol{h}_h{}^\top\ \boldsymbol{R}\ \boldsymbol{t}_t + \boldsymbol{h}_t{}^\top\ \boldsymbol{R}_{-1}\ \boldsymbol{t}_r)$	$\forall\ r \in \mathcal{R} : \boldsymbol{R},\ \boldsymbol{R}_{-1}$ are diagonal	$\mathcal{O}(2\lvert\mathcal{E}\rvert d + 2\lvert\mathcal{R}\rvert d)$
	HolE	Negative Log-Likelihood	$\boldsymbol{r}^\top(\boldsymbol{h} \star \boldsymbol{t})$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
	TuckER	Binary Cross-Entropy	$\boldsymbol{W} \times_1 \boldsymbol{h} \times_2 \boldsymbol{r} \times_3 \boldsymbol{t}$		$\mathcal{O}(\lvert\mathcal{E}\rvert d_e + \lvert\mathcal{R}\rvert d_r + d_e d_r d_e)$
Geometric models	**TransE**	Pairwise Ranking	$\lVert\ \boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}\ \rVert$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
	STransE	Pairwise Ranking	$\lVert\ \boldsymbol{W}_r^h\ \boldsymbol{h} + \boldsymbol{r} - \boldsymbol{W}_r^t\ \boldsymbol{t}\ \rVert$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert(d + 2d^2))$
	CrossE	Logistic (NLL variant)	$tanh(\boldsymbol{h} \odot \boldsymbol{c}_r + \boldsymbol{r} \odot \boldsymbol{h} \odot \boldsymbol{c}_r + \boldsymbol{b})^\top\ \boldsymbol{t}$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + 2\lvert\mathcal{R}\rvert d)$
	TorusE	Pairwise Ranking	$\min_{(x,\ y)\ \in\ [h]\ +\ [r]\ \times\ [t]} \lVert\ \boldsymbol{x} - \boldsymbol{y}\ \rVert$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
	RotatE	Self-adversarial Negative Sampling	$-\lVert\ \boldsymbol{h} \odot \boldsymbol{r} - \boldsymbol{t}\ \rVert$	$\boldsymbol{h} \in \mathbb{C}^d;\ \boldsymbol{r} \in \mathbb{C}^d;\ \boldsymbol{t} \in \mathbb{C}^d$ $\forall\ r_i \in \boldsymbol{r} : \lvert r_i\rvert = 1$	$\mathcal{O}(2\lvert\mathcal{E}\rvert d + 2\lvert\mathcal{R}\rvert d)$
	HAKE	Self-adversarial Negative Sampling	$-\lambda_m \lVert\ \boldsymbol{h}_m \odot \boldsymbol{r}_m - \boldsymbol{t}_m \rVert_2 +$ $-\lambda_p \left\lVert\ sin\left(\frac{\boldsymbol{h}_p + \boldsymbol{r}_p - \boldsymbol{t}_p}{2}\right)\right\rVert_1$	$\boldsymbol{h} \in \mathbb{C}^d;\ \boldsymbol{r} \in \mathbb{C}^d;\ \boldsymbol{t} \in \mathbb{C}^d$	$\mathcal{O}(2\lvert\mathcal{E}\rvert d + 2\lvert\mathcal{R}\rvert d)$
Deep Learning models	**ConvE**	Binary Cross-Entropy	$g(\ g([\boldsymbol{h}\ ;\ \boldsymbol{r}] \circledast \boldsymbol{\omega})\ \boldsymbol{W} + \boldsymbol{b})\ \boldsymbol{t}$		$\mathcal{O}\left(\begin{matrix}\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d\ + T m_\omega n_\omega + \\ + T d(2d_m - m_\omega + 1)(d_n - n_\omega + 1)\end{matrix}\right)$
	ConvKB	Negative Log-Likelihood	$g([\boldsymbol{h}\ ;\ \boldsymbol{r}\ ;\ \boldsymbol{t}] \circledast \boldsymbol{\omega})\ \boldsymbol{W} + \boldsymbol{b}$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d\ + 4T)$
	ConvR	Binary Cross-Entropy	$g(\boldsymbol{W}\ g([\boldsymbol{h}] \circledast \boldsymbol{\omega}_r) + \boldsymbol{b})^\top\ \boldsymbol{t}$		$\mathcal{O}\left(\begin{matrix}\lvert\mathcal{E}\rvert d_e + \lvert\mathcal{R}\rvert d_r\ + T d_r + \\ + T d_e(2d_{e_m} - m_\omega + 1)(d_{e_n} - n_\omega + 1)\end{matrix}\right)$
	InteractE	Binary Cross-Entropy	$g(\ g(perm([\boldsymbol{h}\ ;\ \boldsymbol{r}]) \circledast \boldsymbol{\omega})\ \boldsymbol{W} + \boldsymbol{b})\ \boldsymbol{t}$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d\ + T m_\omega n_\omega + 2T p d^2)$
	CapsE	Negative Log-Likelihood	$\lVert\ capsnet(g([\boldsymbol{h}\ ;\ \boldsymbol{r}\ ;\ \boldsymbol{t}] \circledast \boldsymbol{\omega}))\ \rVert$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d\ + 3T + 3d)$
	RSN	Logistic (NLL variant for RNNs with NCE)	$rsn(\boldsymbol{h},\ \boldsymbol{r})^\top\ \boldsymbol{t}$		$\mathcal{O}(2\lvert\mathcal{E}\rvert d + 2\lvert\mathcal{R}\rvert d\ + Lknd)$

Table 1. Original loss Function, Scoring function, constraints and space complexity for each model included in our analysis. Legend: $\times_i$: matrix product along mode i; $\odot$: element-wise product; $\star$: circular correlation; $\circledast$: convolution; $\circledcirc$: circular convolution; $||\boldsymbol{x}||_i$: L_i Norm of $\boldsymbol{x}$ (if no i is specified, its value depends on the setting); $capsnet$: capsule network operation; rsn: RSN layer operation; σ: sigmoid function; g: generic activation function; $\overline{\boldsymbol{e}}$: conjugate-transpose of $\boldsymbol{e}$; $[\boldsymbol{a}; \boldsymbol{b}]$: concatenation and (potential) reshape of $\boldsymbol{a}$ and $\boldsymbol{b}$; $perm[\boldsymbol{a}; \boldsymbol{b}]$: generate and stack p permutations for the concatenated $\boldsymbol{a}$ and $\boldsymbol{b}$: $\boldsymbol{r}_{-1}$: embedding for the inverse relation of r; $\boldsymbol{e_h}$, $\boldsymbol{e_r}$: embeddings for entity e occurring as head and as tail respectively; $\boldsymbol{\omega}$: set of filters used in a convolutional layer; $\boldsymbol{\omega_r}$: convolutional filters obtained from $\boldsymbol{r}$ in $ConvR$; d: embedding size for entities and relations when they have same size; d_e, d_r: embedding sizes for entities and relations when they have different size; T: number of convolutional filters in a layer; m_ω, n_ω: dimensions of the convolutional filters in $\boldsymbol{\omega}$; d_m, d_n: dimensions of the reshaped $[\boldsymbol{h}; \boldsymbol{r}]$ in $ConvE$ ($d_m \times d_n = 2d$); d_{e_m}, d_{e_n}: dimensions of the reshaped $[\boldsymbol{h}]$ in $ConvR$ ($d_{e_m} \times d_{e_n} = d_e$); p: number of permutations in $InteractE$; L: number of RSN layers in a RSN cell; k: number of weight matrices in a RSN layer; n: number of neurons in a RSN layer. In bilinear models that enforce the relation embedding to be a diagonal matrix, $\boldsymbol{r}$ can be seen as a d-dimensional vector. This leads to the following alternative notations: DistMult: $\phi(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \langle \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \rangle$; Complex: $\phi(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = Re(\langle \boldsymbol{h}, \boldsymbol{r}, \overline{\boldsymbol{t}} \rangle)$; SimplE: $\phi(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \frac{1}{2}(\langle \boldsymbol{h_h}, \boldsymbol{r}, \boldsymbol{t_t} \rangle + \langle \boldsymbol{h_t}, \boldsymbol{r_{-1}}, \boldsymbol{t_h} \rangle)$.

FB15k-237 was built by Toutanova and Chen [60] after finding that FB15k suffers from *test leakage* (that is, test data being seen by models at training time) due to the presence of equivalent and inverse relations. They assessed the severity of this issue showing how a simple model based on observable features can reach state-of-the-art performance on FB15k. They have then further sampled FB15k to create a more challenging dataset, removing all equivalent or

	Entities	Relations	Triples			Reified Relations	Test Leakage	Multiple Domains
			Train	Valid	Test			
FB15k	14951	1345	483142	50000	50971	✗	✗	✗
WN18	40943	18	141442	5000	5000		✗	
FB15k-237	14541	237	272115	17535	20466	✗		✗
WN18RR	40943	11	86835	3034	3134			
YAGO3-10	123182	37	1079040	5000	5000			✗

Table 2. The 5 LP datasets included in our comparative analysis, and their general properties.

inverse relations, as well as the least mentioned ones. In order to filter away all trivial triples, they also ensured that none of the entities connected in the training set are also directly linked in the validation and test sets.

WN18RR is a WN18 subset built by Dettmers *et al.* [10] after observing in WN18 test leakage analogous to FB15k. They proved that a simple Inverse Model leveraging inverse relations reaches state-of-the-art results in both WN18 and FB15k, and built WN18RR by removing inverse relations from WN18. It has been recently observed that 212 entities in its test set are never mentioned in training, thus making about 6.7% test facts impossible to reasonably predict.

YAGO3-10 is a YAGO3 [39] sample also proposed by Dettmers *et al.* [10]. It was built by identifying all entities with at least 10 different relations in the KG, and by extracting all the corresponding facts. As stated by the authors, the majority of its triples deal with descriptive properties of people, such as citizenship or gender. The poor performance of the Inverse Model [10] on YAGO3-10 imply that it should not suffer from the same test leakage as FB15k and WN18.

4.2 Efficiency Analysis

For each model, we consider two main metrics for efficiency:

- *Training Time*: time required to learn optimal embeddings for all entities and relations.
- *Prediction Time*: time required to generate full rankings for one test fact, including both head and tail predictions.

Training time and prediction time mostly depend (*i*) on the model architecture (e.g. deep neural networks may require longer computations due to their shared parameters and inherently longer pipeline of operations); (*ii*) on the model hyperparameters, such as embedding size and number of negative samples for each positive one; (*iii*) on the dataset size, namely the number of entities and relations to learn and the number of training triples to process. We also investigate how the training times of the best performing models in each family scale when dealing with increasing volumes of data.

4.3 Effectiveness Analysis

We analyze the effectiveness of LP models based on the structure of the training graph. We define measurable structural features and we treat each of them as a separate research direction, investigating how it correlates to the predictive performance of each model in each dataset. We take into account 4 different structural features for each test fact:

- *Number of Peers*, namely the valid alternatives for the source and target entities;
- *Relational Path Support*, taking into account paths connecting the head and tail of the test fact;
- *Relation Properties* that affect both the semantics and the graph structure;

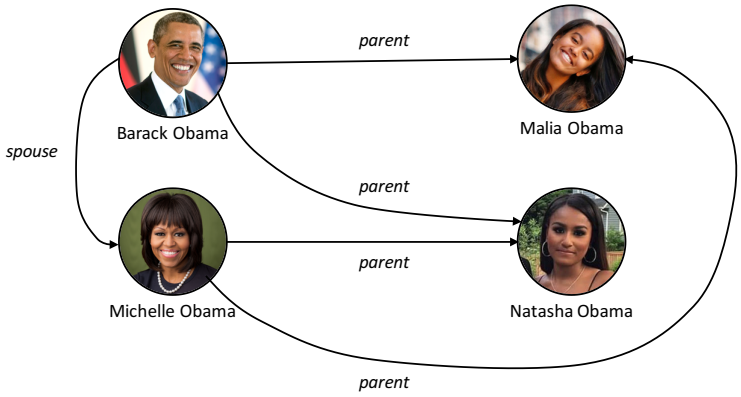


Fig. 2. Example of head peers and tail peers in a small portion of a KG.

- *Degree of the original reified relation*, for datasets generated from KGs using reification.

We address and explain these features in Sections 4.3.1, 4.3.2, 4.3.3, 4.3.4 respectively.

4.3.1 Number of Peers. Given any fact $\langle h, r, t \rangle$ we define peers with the following formulations:

- *head peers*: the set of entities $\{h' \in \mathcal{E} \mid \langle h', r, t \rangle \in \mathcal{G}_{train}\}$;
- *tail peers*: the set of entities $\{t' \in \mathcal{E} \mid \langle h, r, t' \rangle \in \mathcal{G}_{train}\}$.

In other words, the head peers are all the alternatives to h conditioned to having relation r and tail t; analogously, the tail peers are the known alternatives to t when the head is h and the relation is r. We only focus on peers seen in training because we are interested in how the structure of the training graph affects performance. Following the notation in Section 2 we name the peers for the source and the target entity of a prediction *source peers* and *target peers* respectively. We illustrate an example in Figure 2: in fact ⟨*Barack Obama*, *parent*, *Malia Obama*⟩, entity *Michelle Obama* is a peer to *Barack Obama* because it is *parent* to *Malia Obama* too. Analogously, *Natasha Obama* is a peer for *Malia Obama*. In head prediction *Malia Obama* is the source and *Barack Obama* is the target entity, so *Natasha Obama* is a source peer and *Michelle Obama* is a target peer. In tail prediction, source and target peers are just reversed.

Our intuition is that source and target peers may affect predictions with subtle, possibly unanticipated effects.

On the one hand, source peers can be seen as the training samples from which LP models directly learn how to predict the current target entity, given the current source entity and relation. When predicting the tail in ⟨*Barack Obama*, *nationality*, *USA*⟩, the source peers are all the other entities with *nationality USA* that the model gets to see in training: they provide examples from which the models can learn what makes a person have American citizenship.

On the other hand, target peers can be seen as the answers correctly satisfying this prediction seen during training. For instance, given the same fact as before ⟨*Barack Obama*, *nationality*, *USA*⟩, but performing head prediction this time, the other USA citizens are now target peers. Since all of them constitute valid alternatives to the target entity, a skewed distribution of target peers may intuitively lead models to confusion and performance degradation.

Our experimental results on source and target peers, reported in Section 5.3.1, confirm our hypothesis.

4.3.2 Relational Path Support. In a KG a *path* is a sequence of chained facts, in which the tail of each fact corresponds to the head of the next one. The *length* of the path is the number of facts it contains. A *relational path* is the sequence

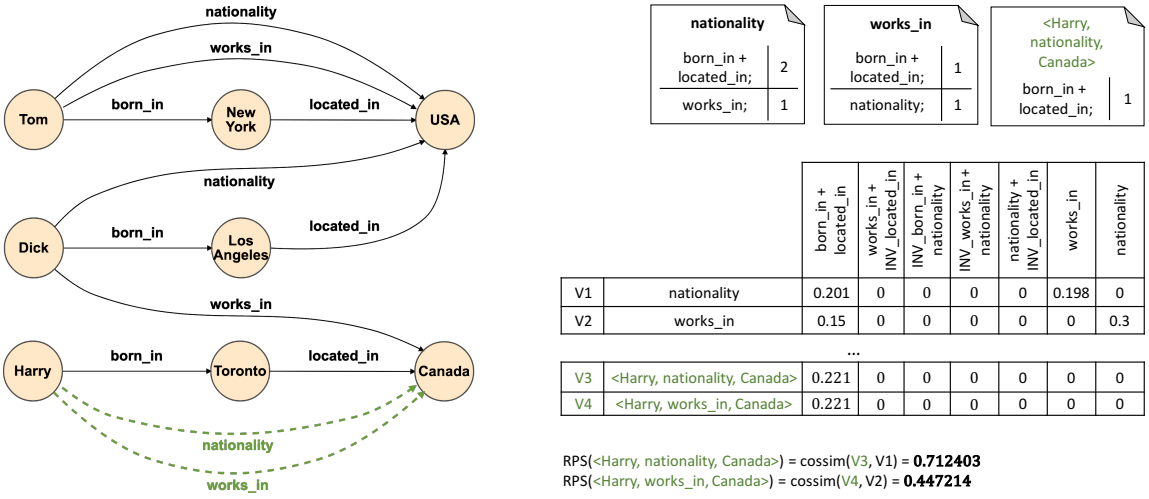
of relations in a path, ignoring its entities. Relational paths can embody logical patterns corresponding to specific relations: for instance, given ⟨*Barack Obama*, *place of birth*, *Honolulu*⟩ and ⟨*Honolulu*, *located in*, *USA*⟩, it may be possible to predict ⟨*Barack Obama*, *nationality*, *USA*⟩ via the path *place-of-birth-located-in*. Paths have been used for a long time in LP methods based on observable features, such as the Path Ranking Algorithm [34, 35]. On the other hand, the vast majority of embedding-based models just learn individual facts separately; notable exceptions are PTransE [37] or, in our analysis, RSN [20]. Some models do not employ paths in training but use them for additional tasks, such as the explanation approach proposed by CrossE [78].

Our intuition is that even models that train on individual facts acquire indirect knowledge of paths as they progressively scan the training set: leveraging these logical patterns can allow them to achieve better results. To verify this, we define a novel measure of *Relational Path Support* (RPS) that estimates for any fact how the relational paths connecting its head to its tail facilitate their prediction. In greater detail, the RPS value for a fact ⟨*h*, *r*, *t*⟩ measures how the relation paths connecting *h* to *t* match those most usually co-occurring with *r*. In models that heavily rely on relation patterns, a high RPS value should correspond to good predictions, whereas a low one should correspond to bad ones.

Our RPS metric is a variant of the Term Frequency–Inverse Document Frequency (TF-IDF) statistical measure [40] commonly used in Information Retrieval. The TF-IDF value of any word w in a document D of a collection C measures both how relevant and how specific w is to D, based respectively on the frequency of w in D and on the number of other documents in C including w. Any document and any keyword-based query can be modeled as a vector with the TF-IDF values of all words in the vocabulary. Given any query Q, a TF-IDF-based search engine will retrieve the documents with vectors most similar to the vector of Q.

In our scenario each relation path p is analogous to a word and each relation r is analogous to a document. When a relation path p co-occurs with a relation r (that is, it connects the head and tail of a fact featuring r) we interpret this as the word p belonging to the document r. We treat each test fact q as a query whose keywords are the relation paths connecting its head to the tail. In greater detail, this is the procedure we apply to compute our RPS measure:

(1) For each training fact $\langle h, r, t \rangle$ we extract from $\mathcal{G}_{train}$ all relational paths p leading from h to t. We invert any steps with the wrong orientation by prepending "INV" to their relation name. Our vocabulary V is the set of resulting relational paths. To keep computations feasible, we consider paths with length equal or lesser than 3.
(2) We merge the relation paths extracted for training facts featuring the same relation r. We obtain:
   - the number n_r of training facts featuring r;
   - for each relational path $p \in V$, the number n_{r_p} of times that p co-occurs with r. Of course, $\forall(r, p)\ n_r \geq n_{r_p}$.
(3) We compute *Document Frequencies* (DFs): $\forall p \in V \quad DF[p] = |\{r \in \mathcal{R} : n_{r_p} > 0\}|$.
(4) We compute *Term Frequencies* (TFs): $\forall r \in \mathcal{R},\ \forall p \in V,\ TF[r][p] = \frac{n_{r_p}}{\sum_{x \in V} n_{r_x}}$.
(5) We compute *Inverse Document Frequencies* (IDFs): $\forall p \in V \quad IDF[p] = log(\frac{|\mathcal{R}|}{DF[p]})$.
(6) For each relation we compute the *TF-IDF* vector: $\forall r \in \mathcal{R} \quad TFIDF_r = [\ \forall p \in V \quad TF[r][p] * IDF[p]\ ]$.
(7) For each test fact q we extract the set of relational paths in $\mathcal{G}_{train}$ connecting its head to its tail, as in point (1).
(8) For each q we apply the same formulas seen in points (3) - (6) to compute *DF*, *TF* and *IDF* and the whole *TF-IDF* vector; in all computations we treat each q as if it was an additional document.
(9) For each q we compute *RPS* as the cosine-similarity between its TF-IDF vector and the TD-IDF vector of its relation r_q: $RPS(q) = cossim(TF-IDF_q, TF-IDF_{r_q})$.



		born_in + located_in	works_in + INV_located_in	INV_born_in + nationality	INV_works_in + nationality	nationality + INV_located_in	works_in	nationality
V1	nationality	0.201	0	0	0	0	0.198	0
V2	works_in	0.15	0	0	0	0	0	0.3
	...							
V3	<Harry, nationality, Canada>	0.221	0	0	0	0	0	0
V4	<Harry, works_in, Canada>	0.221	0	0	0	0	0	0

RPS(<Harry, nationality, Canada>) = cossim(V3, V1) = **0.712403**
RPS(<Harry, works_in, Canada>) = cossim(V4, V2) = **0.447214**

Fig. 3. Example for Relational Path Support

All in all, the RPS of a test fact estimates how similar it is to training facts with the same relation in terms of co-occurring relation paths. This corresponds to measuring how much the relation paths suggest that, given the source and relation in the test fact, the target is indeed the right answer for prediction.

We stress that in any prediction, the relational path support is completely orthogonal to the number of peers discussed in Section 4.3.1. While the number of peers only depends on the local neighborhood of the source and target entity, RPS relies on paths whose length is typically greater than one. In other words, the number of peers can be seen as a form of information very close to the test fact, whereas RPS is more prone to take into account longer-range dependencies. As a matter of fact, it is not uncommon to observe predictions that, despite being facilitated by their peers, display low RPS values, or vice versa.

Example 4.1. The graph in Figure 3 represent training facts with black solid edges and test facts with green dashed edges. The collection of documents is $C = \{nationality, works\_in, born\_in, located\_in\}$. Our queries stem from test facts $\langle$ *Harry*, *nationality*, *Canada* $\rangle$ and $\langle$ *Harry*, *works_in*, *Canada* $\rangle$; note that such facts share the same head and tail, so their queries have the same keywords (the relational path *born_in + located_in*). We compute words and frequencies for each document and query, and obtain TF-IDF values for each word in each document following the above described procedure. For instance, for document *nationality* and word $born\_in \ + \ located\_in$:

- $TF(born\_in + located\_in,\ nationality) \ = \ \frac{co-occurrences\ of\ born\_in + located\_in\ with\ nationality}{co-occurrences\ of\ all\ relational\ paths\ with\ nationality} \ = \frac{2}{2+1} \ \simeq \ 0.67$
- $IDF(born\_in \ + \ located\_in) \ = \ \log_{10}(\frac{all\ documents}{documents\ containing\ born\_in + located\_in}) \ = \ \log_{10}(\frac{4}{2}) \ \simeq \ 0.3$
- $TFIDF(born\_in + located\_in,\ nationality) \ = \ TF \ \times \ IDF \ = \ 0.67 * 0.3 = 0.201$

Similarly, $TFIDF(\ born\_in \ + \ located\_in,\ works\_in\ ) = 0.15$ and $TFIDF(\ works\_in,\ nationality\ ) = 0.198$.
The TF-IDF value for each query can be computed analogously, except that the query must be included among the documents. The two queries our example share the same keywords, so they will result in identical vectors.

- $TF(born\_in + located\_in, test fact) \ = \ \frac{co-occurrences\ of\ born\_in + located\_in\ with\ test\ fact}{all\ relational\ paths\ co-occurring\ with\ test\ fact} \ = \frac{1}{1} \ = \ 1.0$
- $IDF(born\_in \ + \ located) \ = \ log_{10}(\frac{all\ documents}{documents\ containing\ born\_in + located\_in})) \ = \ log_{10}(\frac{4+1}{2+1}) \ \simeq \ 0.221$

- $TFIDF(born\_in + located\_in, test fact) = TF \times IDF = 1 * 0.221 = 0.221$

The RPS for ⟨ *Harry*, *nationality*, *Canada* ⟩ is the cosine-similarity between its vector and the vector of *nationality*, and it measures 0.712403; analogously, the RPS for ⟨ *Harry*, *works_in*, *Canada* ⟩ is the cosine-similarity with the vector of *nationality*, and it measures 0.447214. As expected, the former value is higher than the latter, because the paths connecting *Harry* with *Canada* are more similar to those usually co-occurring with *nationality* than with *works_in*. In other words, in our small example the relation path *born_in + located_in* supports *nationality* more than *works_in*.

Our experimental results on the analysis of relational path support are reported in Section 5.3.2.

4.3.3 Relation Properties. The semantics of relations give them properties that can heavily affect the structures and patterns they form in KGs. For instance symmetric relations are likely to connect couples of entities in both directions.

Relation properties can make facts easier or harder to learn and predict for a LP model, depending on its scoring function; some models may even be incapable of learning certain relation types at all. For instance, TransE [6] can only learn null vectors for symmetric and transitive relations due to the nature of translations; analogously, DistMult [77] can not handle anti-symmetric relations, because given any fact $\langle h, r, t \rangle$, it assigns the same score to $\langle t, r, h \rangle$ too.

In this regard, a model is defined as **fully expressive** [30] if, given any graph, there exists at least one embedding that allows the model to separate all correct triples from incorrect ones. In other words, a fully expressive model has the theoretical potential to correctly learn any graph, not being hindered by intrinsic limitations. Examples of fully expressive models are SimplE [30], TuckER [4], ComplEx [66] and HolE [65].

Being capable of learning certain relations, however, does not necessarily imply reaching good performance on them. Even fully expressive models may find certain properties harder than others. For instance, Meilicke *et al.* [41] have reported surprisingly bad results for HolE on symmetric relations in FB15K, despite HolE being fully expressive. Therefore, we lead a systematic analysis, defining a comprehensive set of relation properties and verifying how they affect the performance of all our models. We take into account the following properties:

- *Reflexivity*: by definition a reflexive relation connects each element with itself; this is not suitable for KGs, where each entity may only be involved with some relations based on its type. As a consequence, in our analysis we use the following definition: $r \in \mathcal{R}$ is reflexive if $\forall \langle h, r, t \rangle \in \mathcal{G}_{train} \implies \langle h, r, h \rangle \in \mathcal{G}_{train}$.
- *Irreflexivity*: $r \in \mathcal{R}$ is irreflexive if $\forall e \in \mathcal{E} \implies \langle e, r, e \rangle \notin \mathcal{G}_{train}$.
- *Symmetry*: $r \in \mathcal{R}$ is symmetric if $\forall \langle h, r, t \rangle \in \mathcal{G} \implies \langle t, r, h \rangle \in \mathcal{G}$.
- *Anti-symmetry*: $r \in \mathcal{R}$ is anti-symmetric if $\forall \langle h, r, t \rangle \in \mathcal{G} \implies \langle t, r, h \rangle \notin \mathcal{G}$.
- *Transitivity*: $r \in \mathcal{R}$ is transitive if $\forall \langle h, r, x \rangle \in \mathcal{G}, \langle x, r, t \rangle \in \mathcal{G} \implies \langle h, r, t \rangle \in \mathcal{G}$.

We do not consider other properties, such as Equivalence and Order (partial or complete), because we have observed that in the 5 datasets of our analysis they would only involve a negligible number of test facts.

On all datasets we use the following approach. For each relation r we extract the corresponding training facts and use them to identify the properties of r. Due to the inherent incompleteness of the training set, we assume that a property is verified if the ratio of training facts satisfying the corresponding condition exceeds a tolerance threshold, set to 0.5 in all our experiments. We bucketize test facts based on the properties of their relations; if a relation has multiple properties, its test facts will belong to multiple buckets. Finally, we compute how each model performs on each bucket.

We report our results regarding relation properties in Section 5.3.3.

4.3.4 Reified Relations. Some KGs support relations with cardinality greater than 2, that is, connecting more than two entities at a time. Depending on their semantics, some relations make more sense when modeled in this way, e.g. an

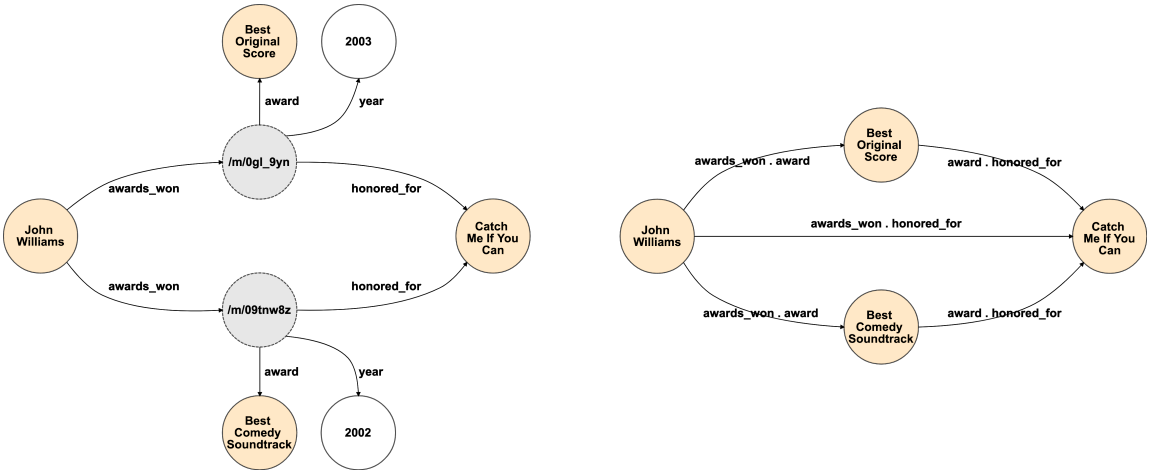


Fig. 4. Example of how the Star2Clique process operates on a small portion of a KG.

actor winning an award for their performance in a movie can be modeled as a unique fact connecting the actor, the award and the movie. KGs supporting these not-binary relations often handle them with one of these methods:

- using **hyper-graphs**: in a hyper-graph, each hyper-edge can link more than two nodes at a time by design. Hyper-graphs can not be directly expressed as sets of triples.
- using **reification**: each edge linking multiple entities is modeled as an intermediate node connected to those entities by binary edges. The relation cardinality thus becomes the degree of the reified nodes used to represent it. The graph can still be expressed as a set of triples.

FreeBase, from which FB15k and FB15k-237 have been extracted, employs reification, and denotes the intermediate reified nodes with type *Compound Value Type* (CVT). By extension, we refer to these nodes as CVTs.

When generating FB15k [6], CVTs have been removed and converted into cliques; the entities previously linked to the CVT have been connected to one another with new binary edges, whose labels have been obtained by concatenating the old ones. This also applies to FB15k-237, that is just a further sample of FB15k [60]. It has been pointed out that this conversion, dubbed "Star-to-Clique" (S2C), is irreversible [75]. In our study we observe further consequences:

- From a structural standpoint, S2C transforms a CVT with degree n into a clique with at most n^2 edges. Therefore, some parts of the graph artificially become much denser than in the KG. The generated edges are often redundant; in the filtering operated to create FB15k-237, many of them have been removed.
- From a semantic standpoint, the original meaning of the relations is vastly altered. As shown in Figure 4, after generating the cliques, edges are deduplicated: if the same two entities were linked with the same types of relations through multiple CVTs, e.g. an artist winning multiple awards for the same work, this information is lost. In other words, in the new semantics, each fact represents an event taking place *at least once*.

We hypothesize that the very dense, redundant and locally-consistent areas generated with S2C may influence predictive behaviours. Therefore, for FB15k and FB15k-237, we recover from FreeBase[5] the original CVT for each S2C-derived test fact, in order to correlate its degree with the predictive performance of our models. As already pointed

[5]We use the latest FreeBase dump, available at https://developers.google.com/freebase

out, S2C is not reversible due to deduplication, so this process often yields multiple CVTs for the same fact; in this circumstance we take into account the CVT with highest degree. For a few S2C-derived test facts no CVTs can be found in FreeBase at all; in this case, we apply the minimum possible degree, that is 2.

The results of our analysis on the correlation between the degree of the original reified relation and the predictive performance are reported in Section 5.4.

5 EXPERIMENTAL RESULTS

In this section we provide a detailed report of the experiments and comparisons carried out in our work.

5.1 Experimental set-up

We start with a brief overview of the environment we use in our analysis, and of the procedures we follow to train and evaluate all LP models. We also provide a description of the baseline model we use in all experiments.

5.1.1 Environment. All of our experiments, as well as the training and evaluation of each model, are performed on a server environment using 88 CPUs Intel Core(TM) i7-3820 at 3.60GH, 516GB RAM and 4 GPUs NVIDIA Tesla P100-SXM2, each with 16GB VRAM. The operating system is Ubuntu 17.10 (Artful Aardvark).

5.1.2 Training procedures. We train and evaluate from scratch all the models introduced in Section 3. In order to make our results reproducible we use, whenever possible, publicly available implementations. In this regard the only model with no implementations available online is ConvR [28]; we thank the authors for kindly sharing their code with us. When multiple implementations are available for the same model, we use the best performing one, with the goal of analyzing each model at its best. This results in the following choices:

- For TransE [6], DistMult [77] and HolE [52] we use the implementation provided by project Ampligraph [9];
- For ComplEx [66] we use the version with N3 regularization by Lacroix [33], sometimes called ComplEx-N3 [14];
- For SimplE [30] we use the fast implementation by Fatemi [15], as suggested by the authors of the model.

For all the other models we use the original implementations shared by the authors themselves in their repositories.

As shown by Kadlec *et al.* [29], LP models are very sensitive to hyperparameters, and a separate tuning is often required for each dataset. The authors of a model usually define the space of acceptable values for its hyperparameters, and find the best performing setting for each dataset through grid or random searches. When training a model on a dataset, we thus rely on the setting suggested by the authors, if provided; otherwise we explore the space of hyperparameters ourselves. This scenario is unfortunately quite common, and also considering the sheer size of the hyperparameter spaces (often with thousands of combinations) and the duration of each training (usually taking several hours), we find that even random searches soon become unfeasible. We thus resort to hand tuning to the best of our possibilities, using what is familiarly called a *panda* approach [32] (in contrast to a *caviar* approach where large batches of training processes are launched). We report in Appendix G the best hyperparameter combination we find for each model.

In AnyBURL [42], for datasets FB15k-237 and YAGO3-10 we use training time 1000 seconds while the original paper reports slightly better results when using 10000 seconds. This is because, due to our necessity to extract full rankings, models trained for 10000 seconds result in prohibitively long prediction times. Therefore, under suggestion of the authors, we resort to using the second best training time for these datasets. We also note that STransE [47], ConvKB [48] and CapsE [49] use transfer learning and require embeddings pre-trained on TransE [6]. For FB15k, FB15k-237, WN18

		FB15k				WN18				FB15k-237				WN18RR				YAGO3-10			
		H@1	H@10	MR	MRR	H@1	H@10	MR	MRR	H@1	H@10	MR	MRR	H@1	H@10	MR	MRR	H@1	H@10	MR	MRR
Matrix Decomposition Models	DistMult	73.61	86.32	173	0.784	72.60	94.61	675	0.824	22.44	49.01	199	0.313	39.68	50.22	5913	0.433	41.26	66.12	1107	0.501
	ComplEx	**82.32**	**91.04**	**33**	**0.855**	94.43	96.15	**190**	0.951	**27.18**	**55.80**	**144**	**0.367**	44.27	**58.06**	2867	0.489	**50.09**	**71.29**	**793**	**0.577**
	ANALOGY	65.59	83.74	126	0.726	92.61	94.42	808	0.934	12.59	35.38	476	0.202	35.82	38.00	9266	0.366	19.21	45.65	2423	0.283
	SimplE	66.13	83.63	138	0.726	93.25	94.58	759	0.938	10.03	34.35	651	0.179	38.27	42.65	8764	0.398	35.76	63.16	2849	0.453
	HolE	75.85	86.78	211	0.800	93.11	94.94	650	0.938	21.37	47.64	186	0.303	40.28	48.79	8401	0.432	41.84	65.19	6489	0.502
	TuckER	72.89	88.88	39	0.788	94.64	95.80	510	0.951	25.90	53.61	162	0.352	42.95	51.40	6239	0.459	46.56	68.09	2417	0.544
Geometric Models	TransE	49.36	84.73	45	0.628	40.56	94.87	279	0.646	21.72	49.65	209	0.31	2.79	49.52	3936	0.206	40.57	67.39	1187	0.501
	STransE	39.77	79.60	69	0.543	43.12	93.45	208	0.656	22.48	49.56	357	0.315	10.13	42.21	5172	0.226	3.28	7.35	5797	0.049
	CrossE	60.08	86.23	136	0.702	73.28	95.03	441	0.834	21.21	47.05	227	0.298	38.07	44.99	5212	0.405	33.09	65.45	3839	0.446
	TorusE	68.85	83.98	143	0.746	94.33	95.44	525	0.947	19.62	44.71	211	0.281	42.68	53.35	4873	0.463	27.43	47.44	19455	0.342
	RotatE	73.93	88.10	42	0.791	94.30	96.02	274	0.949	23.83	53.06	178	0.336	42.60	57.35	3318	0.475	40.52	67.07	1830	0.498
	HAKE	74.49	88.44	43	0.796	94.34	**96.19**	230	0.950	24.91	54.20	184	0.347	45.28	**58.06**	3300	**0.497**	46.27	69.54	2068	0.546
Deep Learning Models	ConvE	59.46	84.94	51	0.688	93.89	95.68	413	0.945	21.90	47.62	281	0.305	38.99	50.75	4944	0.427	39.93	65.75	2429	0.488
	ConvKB	11.44	40.83	324	0.211	52.89	94.89	202	0.709	13.98	41.46	309	0.230	5.63	52.50	3429	0.249	32.16	60.47	1683	0.420
	ConvR	70.57	88.55	70	0.773	94.56	95.85	471	0.950	25.56	52.63	251	0.346	43.73	52.68	5646	0.467	44.62	67.33	2582	0.527
	InteractE	72.58	88.68	623	0.786	94.63	95.55	403	0.950	26.35	53.73	185	0.355	42.72	51.96	5056	0.458	46.61	68.51	2182	0.543
	CapsE	1.93	21.78	610	0.087	84.55	95.08	233	0.890	7.34	35.60	405	0.160	33.69	55.98	**720**	0.415	0.00	0.00	60674	0.000
	RSN	72.34	87.01	51	0.777	91.23	95.10	346	0.928	19.84	44.44	248	0.280	34.59	48.34	4210	0.395	42.65	66.43	1339	0.511
	AnyBURL	81.47	87.96	217	0.837	**94.72**	95.62	358	**0.953**	26.92	51.83	276	0.353	**45.77**	57.63	4035	**0.497**	49.17	68.48	1818	0.560

Table 3. Global H@1, H@10, MR and MRR results for all LP models on each dataset. The best results of each metric for each dataset are marked in bold and underlined. The hyperparameters used for each model to obtain the reported results can be found in Appendix G.

and WN18RR we use TransE embeddings shared by the authors across their repositories; for YAGO3-10, which the authors have not considered, we use embeddings trained with the TransE implementation we use in our analysis [9].

Moreover, for the best-performing models in each family we run additional training processes on a selection of datasets; we extract the evaluation results and apply the Boostrap method [13] to identify confidence intervals for each evaluation metric. We find that all metrics are generally very stable across different trainings, with MRR being the most consistent one, and MR, unsurprisingly, the most variable one. Detailed results for this experiment can be found in Appendix H.

5.1.3 Evaluation Metrics. When evaluating predictive performance we always focus on the *filtered scenario*. We report in Table 3 global results for each model using the four most popular metrics: H@1, H@10, MR and MRR. In our finer-grained experiments we use H@1 and MRR, as relying on a H@K measure coupled with a mean-rank-based

measure is very popular in LP works. In particular, MRR is a very stable metric, unlike MR which is highly sensitive to outliers, and H@1 favours the emerging of behavioural differences among models, as pointed out by Kadlec *et al.* [29]. In this paper we mostly report H@1 results for our experiments, as we usually observe analogous trends with MRR.

As described in Section 2, different models may rely on different *tie-breaking policies*. We modify the implementation of each model to support evaluation with multiple policies, and we find that in a few systems *min* results are significantly different from the others. This proves that using different tie-breaking policies may make results not directly comparable to one another; therefore, unless differently specified, when a system originally relies on *min* policy we report its *average* results instead, to make them comparable to the results of the other models. We discuss further experiments and findings on this topic in Section 5.5.

5.1.4 Baseline. As a baseline we use AnyBURL [42], a popular LP model based on observable features. AnyBURL (acronym for Anytime Bottom-Up Rule Learning) treats each training fact as a compact representation of a very specific rule; it then tries to generalize it, in order to cover and satisfy as many training facts as possible. In greater detail, AnyBURL samples paths of increasing length from the training dataset. For each path of length n, it computes a rule of $n-1$ atoms, and stores it if some quality criteria are matched. AnyBURL keeps analyzing paths of same length n until a saturation threshold is exceeded; when this happens, it moves on to paths of length $n+1$. The AnyBURL version we use in our work is a very recent advancement that also relies on reinforcement learning to mine more robust rules [43].

AnyBURL is a very strong baseline for LP, outperforming most latent-features-based models. It is also computationally fast: depending on the settings, its training can take from a couple of minutes (*100s* setting) to about 3 hours (*10000s* setting). In evaluation, AnyBURL is designed to return the top-k scoring entities for each prediction in the test set. When used in this way, it is strikingly fast as well. In order to use it as a baseline in our experiments, however, we need to extract **full rankings** for each prediction, setting $k = |\mathcal{E}|$: this results in much longer computation times. As a side note, we observe that in AnyBURL it is not possible to score nd rank entities for which no rules have been extracted. This makes it possible to have even the target entity missing from the full ranking. In this very unlikely event we assume that all the entities not included in the ranking have identical score 0, and apply *avg* policy.

Code and documentation for AnyBURL are publicly available [67].

5.2 Efficiency

In this section we investigate the efficiency of LP models in terms of time required for training and for prediction, and we perform a scalability analysis on the best-performing models in each family.

In Figure 5 we illustrate, for each model and dataset, the times required for training, which range from around 1 to about 200-300 hours, with the largest dataset YAGO3-10 requiring the longest time in most models. In this regard, the baseline AnyBURL [42] is strikingly fast; it treats training time as a parameter, and it just requires 100s on FB15k and WN18, and 1000s on FB15k-237, WN18RR and YAGO3-10. As already mentioned, STransE [47], ConvKB [48] and CapsE [49] use embeddings pre-trained on TransE [6]; we do not include pre-training times in our measurements.

In Figure 6 we illustrate, for each model and dataset, the average prediction time, defined as the time to generate full head and tail rankings for one fact. Prediction times are mainly affected by the embedding dimension and the evaluation batch size. ConvKB [48] and CapsE [49] require multiple batches for running one prediction, which may negatively affect their prediction time; in our experiments we use size 2048, the maximum allowed in our setting. We find that ANALOGY [38] behaves particularly well in terms of prediction times; this may possibly depend on this model being implemented in C++. For the baseline AnyBURL [42], we compute prediction times a posteriori by dividing the time

required to perform all test predictions by the number of test facts. The obtained times are significantly higher than the ones for embedding-based methods. We stress that this depends on the fact that AnyBURL is not designed to generate full rankings: using top-k policy with lower k values would result in much faster computations.

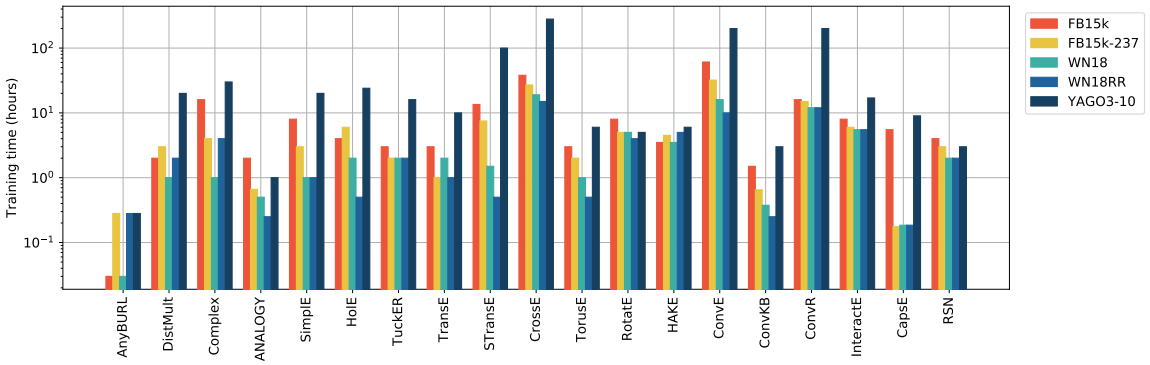


Fig. 5. Training times in hours for each LP model on each dataset. Y axis is in logscale.

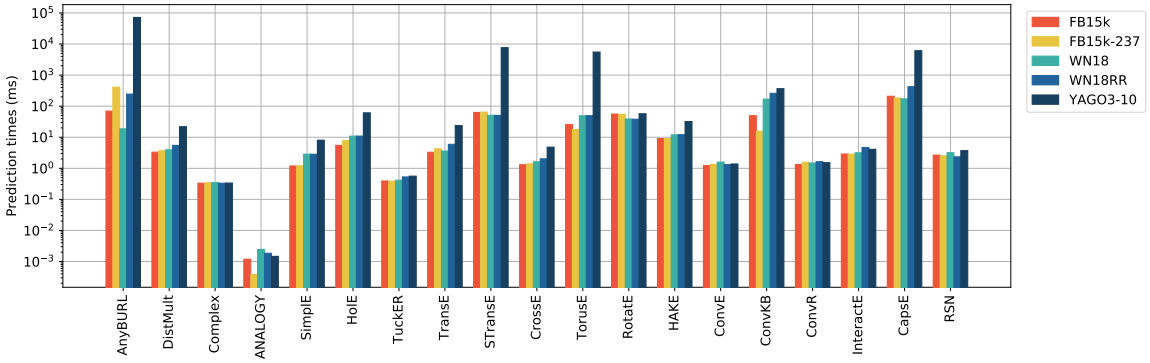


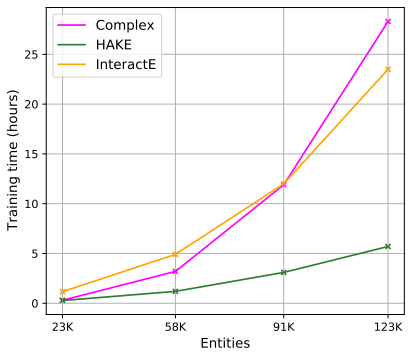
Fig. 6. Prediction times in milliseconds for each LP model on each dataset. Y axis is in logscale.

We finally investigate scalability by assessing the effect of dataset size on training times. We use differently sized samples of the largest dataset YAGO3-10, reporting their dimensions in Table 7c, and we run a dual experiment.

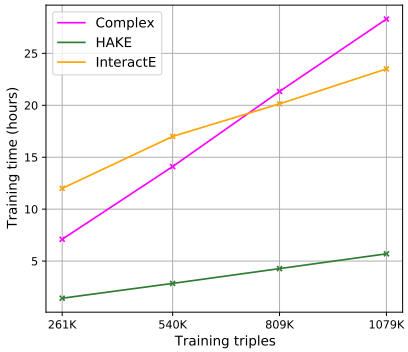
We show in Figure 7a a **node removal experiment**. We progressively remove 25%, 50% and 75% entities with stratified sampling, using 3 strata based on entity degree. Removing an entity implies deleting all its edges and, thus, all the other entities only linked to that one. In order to obtain robust results to the inherent randomness of stratified sampling, we extract 3 independent versions of each sample, and report average dimensions and results across them.

We show in Figure 7b an **edge removal experiment**. We progressively filter away 25%, 50% and 75% triples. Edge removal does not imply removing nodes, so we are able to keep the number of entities constant across all samples.

In both experiments, we report results for the 3 best-performing models of each family, that is, ComplEx [66] for bilinear family, HAKE [80] for the geometric family, and [68] for the Deep Learning family; we use the same hyperparameters and number of training epochs in each training process.



(a) Node Removal



(b) Edge Removal

		$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{G}_{train}\rvert$
FULL		123182	37	1079040
Node Removal	**1**	23112	37	59448
	2	57813	37	261233
	3	91109	37	618806
Edge Removal	**1**	123182	37	134880
	2	123182	37	269760
	3	123182	37	539520

(c) Dataset Samples

Fig. 7. Training times for ComplEx, HAKE and InteractE on multiple YAGO3-10 subsamples obtained with a node removal policy and with an edge removal policy.

Both in node removal and in edge removal, we observe strikingly similar patterns. In all sampled datasets, HAKE is always the least time-consuming model. ComplEx takes less time to train than InteractE in the smallest samples, but it exceeds its training time when taking into account the full YAGO3-10 dataset.

5.3 Effectiveness

In this section we report our results regarding the effectiveness of LP models in terms of predictive performance.

5.3.1 Peer Analysis. Our goal in these experiments is to analyze how predictive performance varies when taking into account test facts with different numbers of source and target peers. Our results are illustrated in Figures 8a, 8b, 8c. These plots show how performance, measured in terms of H@1, trends when increasing the number of source peers or target peers. The plots are incremental, meaning that for each number of source (target) peers we report H@1 for all predictions with source (target) peers equal or lesser than that number. In each chart, for each number of peers we also report the overall percentage of test facts involved: this provides the distribution of facts by peers. For the sake of readability, we only report here charts for FB15k-237, WN18RR and YAGO3-10, and only include the baseline AnyBURL and the best performing models for each family, i.e. ComplEx-N3 [33] for the tensor decomposition models, HAKE [80] for the geometric ones, and InteractE [68] for the deep learning ones. Complete versions of the plots for all datasets and including all models can be found in Appendix A.

Our observations are intriguing. First, we point out that, almost always, a greater number of source peers leads to better H@1 results. As mentioned in Section 4.3.1, this can be explained by considering that, given any prediction with a certain source entity, relation and target entity, the source peers are all the other entities connected to the same target through the same relation in $\mathcal{G}_{train}$. Therefore, they provide examples of other cases in which the target entity plays the same role as in the prediction, thus making it easier to infer it.

Second, we observe that very often a greater number of target peers leads to worse H@1 results. As mentioned in Section 4.3.1, the reason for this behaviour is that, in any prediction with a certain source entity, relation and target entity, target peers are all the other entities connected to the same source through the same relation in $\mathcal{G}_{train}$. They represent the valid alternatives to the target entity of the current prediction: if too many target peers are present,

models are more likely to get confused and slip wrong entities among the answers, thus worsening H@1. We underline that such a degradation is not caused by the target peers just outscoring the target entity: we are working in filtered scenario, therefore target peers, being valid answers to our predictions, do not contribute to the rank computation.

The correlations between numbers of peers and performance are particularly evident in datasets FB15K and FB15K-237. Albeit to a lesser extent, they are also visible in YAGO3-10, especially regarding the source peers. In WN18RR these trends seem much less evident. This is probably due to the very skewed dataset structure: more than 60% predictions involve less than 1 source peer or target peer. In WN18, where the distribution is very skewed as well, models show pretty balanced behaviours. Most of them reach almost perfect results, above 90% H@1.

Summary: Across all datasets and models, a higher number of source peers leads to better predictions, whereas a higher number of target peers produces worse predictions.

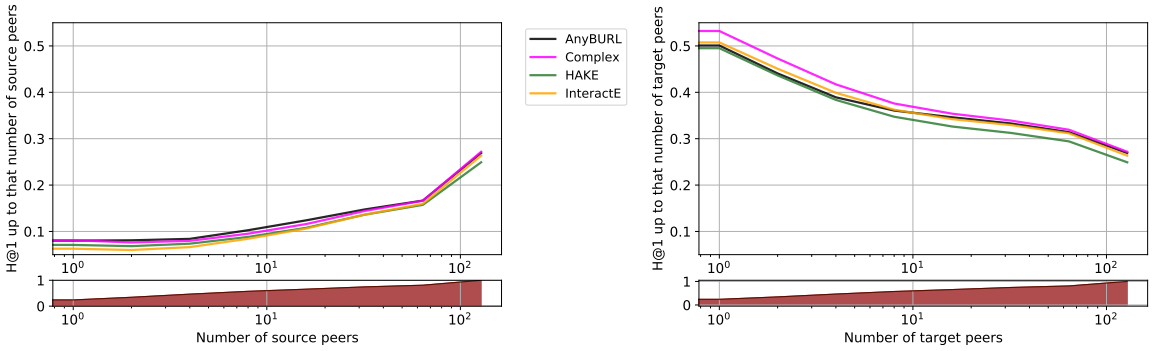
5.3.2 Relational Path Support. Our goal in these experiments is to analyze how predictive performance is influenced by different degrees of relational path support, estimated with the RPS metric introduced in Section 4, using relational paths with length up to 3. Similarly to the peer experiments in Section 5.3.2, we report in Figures 9a, 9b, 9c the results of AnyBURL [42], ComplEx-N3 [33], HAKE [80] and InteractE [68] on datasets FB15k-237, WN18RR and YAGO3-10, measuring performance with H@1. Once more we use incremental metrics, showing for each RPS value the percentage and the H@1 of all the facts with support up to that value. Results for all models and datasets can be found in Appendix B.

In our experiments we observe that, for almost all models, greater RPS values lead to better performance. This proves that such models, to a certain extent, are capable of benefiting from longer-range dependencies. This correlation is clearly visible in all datasets; we find it particularly evident in WN18, WN18RR and YAGO3-10, and to a slightly lesser extent in FB15k-237. We point out that in FB15k-237 and WN18RR a significant percentage of facts displays a very low path support (less than 0.1). This is likely due to the filtering process employed to generate these datasets: removing facts breaks paths in the graph, thus making relational patterns less frequently observable.

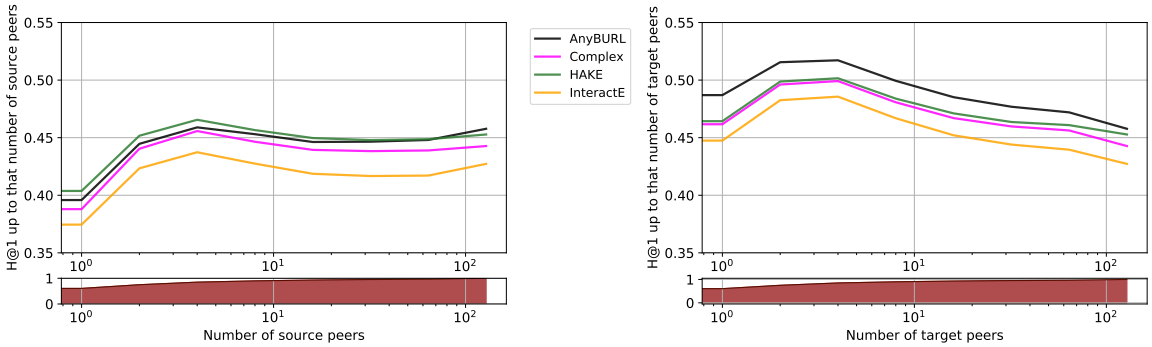
FB15k is the dataset in which the correlation between RPS and performance, albeit clearly observable, seems weakest; we see this as a consequence of the severe test leakage in FB15k. We also find evidence that, in presence of many relations with equivalent or inverse meaning, models tend to focus on shorter dependencies for predictions, ignoring longer relational paths. We show this by replicating our experiment computing RPS values with paths of maximum 1 and 2 instead of 3. In FB15k and WN18, well known for their test leakage, the correlation with performance becomes evidently stronger, while in FB15k-237, WN18RR and YAGO3-10 it weakens, meaning that 3-step relational paths are actually associated with correct predictions in these datasets. We include charts for these findings in Appendix C.

As an additional experiment we use RPS as the scoring function of a standalone LP model based on observable features. We apply the same evaluation pipeline described in Section 2, with some limitations due to computational constraints. The results we obtain show that the RPS metric successfully identifies the presence of valuable logical patterns supporting test facts. Our results, unsurprisingly, are particularly good in dataset featuring inverse relations: they exceed 50% H@1 in FB15k, and approach state-of-the-art performance on WN18, with almost 94% H@1. The experiment setup and the resulting metrics are reported in greater detail in Appendix D.

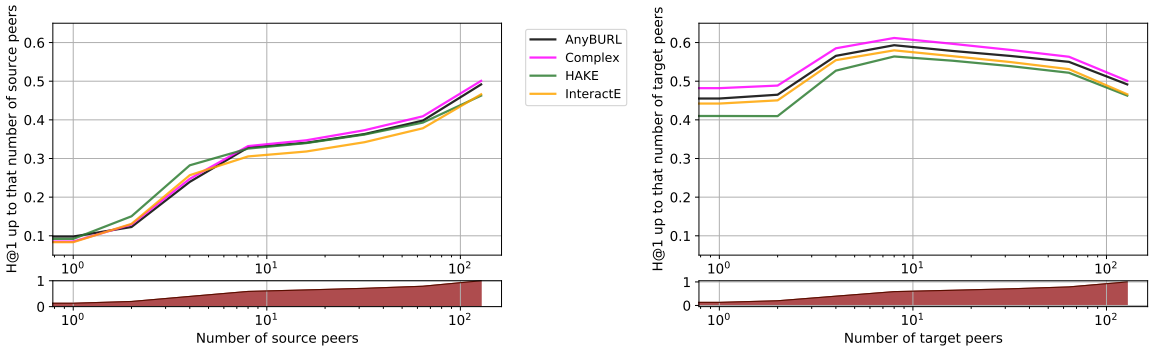
Summary: Higher RPS leads to better predictions across all models and datasets, proving that even models that learn training facts individually can leverage relational paths. In FB15k and WN18, due to test leakage, shorter relational paths are by far the most relevant, whereas in the other datasets LP models rely on longer-range dependencies too.



(a) FB15k-237



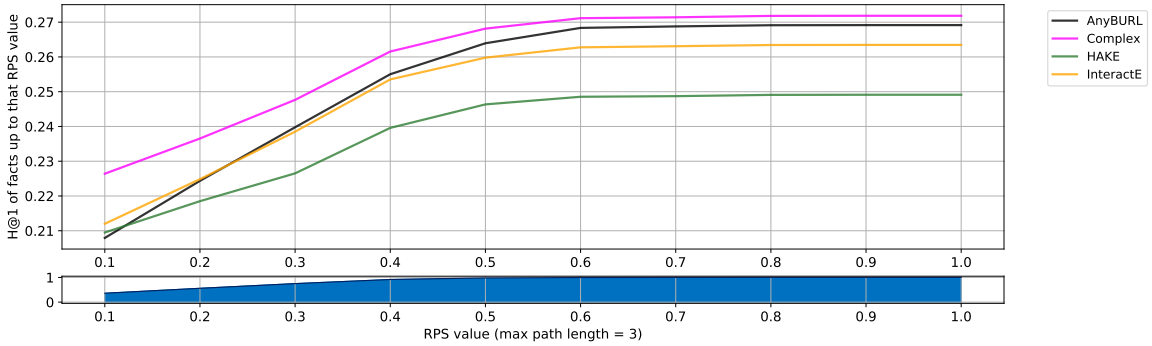
(b) WN18RR



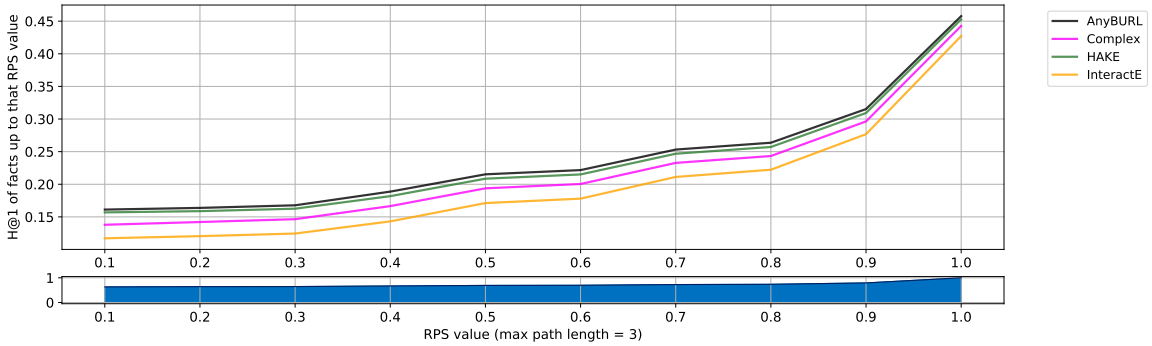
(c) YAGO3-10

Fig. 8. Cumulative H@1 results for each LP model on FB15k-237, WN18RR and Yago3-10 datasets, and corresponding cumulative distribution of test facts, varying the number of source peers (left) and target peers (right). X axis is in logscale.

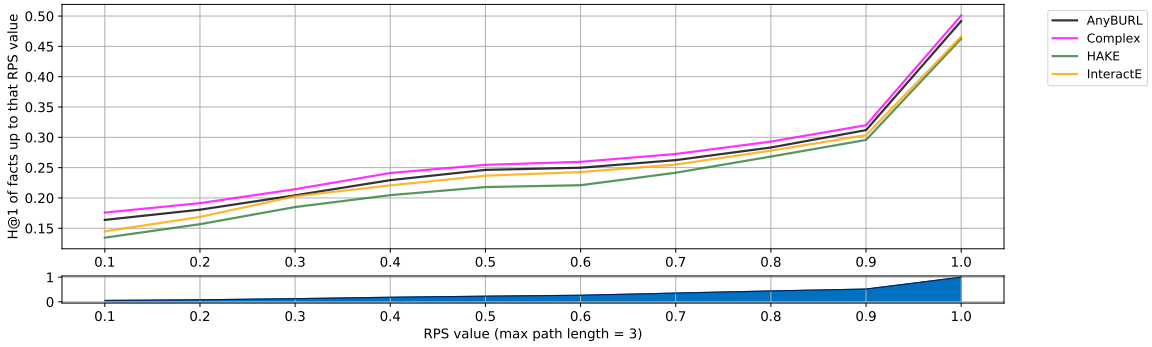
5.3.3 Relation Properties. Our goal in this experiment is to analyze how models perform when handling relations with different properties. We take into account Reflexivity, Symmetry, Transitivity, Irreflexivity and Anti-symmetry, as already described in Section 4.3.3. We divide test facts into buckets based on the properties of their relations; when a relation possesses multiple properties, the corresponding test facts are put in all the corresponding buckets. We report



(a) FB15k-237



(b) WN18RR



(c) YAGO3-10

Fig. 9. H@1 results for each LP model on FB15k-237, WN18 and YAGO3-10 datasets, varying the RPS of the test facts, and corresponding cumulative distribution of test facts.

in Figures 10a,10b, 10c our results, using H@1 for measuring effectiveness. As usual, for the sake of readability we only report in this section results for FB15k-237, WN18RR and YAGO3-10; results for FB15k and WN18 can be found in Appendix E. In all tables we include an initial bucket named *any* with the global H@1 of each model across all test

facts, as a basis for comparison for the performance on the other buckets. Analogously to the distribution graphs in the previous experiments, for each bucket in each dataset we also report the percentage of test facts it contains.

In FreeBase datasets there is an impressive majority of irreflexive and anti-symmetric relations, while just a few facts involve reflexive, symmetric or transitive ones. In WordNet datasets the percentage of facts with symmetric relations is quite higher, but no reflexive and transitive relations are found at all. In YAGO3-10 all test facts feature irreflexive relations. These relations are often anti-symmetric too, whereas they are hardly ever symmetric or transitive.

In FB15K all embedding-based models perform quite well on reflexive relations, while AnyBURL [42] shows the opposite behaviour, possibly due to its rule-based approach. We also observe that translational models such as TransE [6], CrossE [78] and STransE [47] struggle to handle symmetric and transitive relations, with very poor results. This problem seems alleviated by the introduction of rotational operations in TorusE [12], RotatE [59] and HAKE [80].

In FB15K-237, all models display a globally worse performance. Most of them manage to keep good results on reflexive relations (the exceptions being ANALOGY [38], SimplE [30], ConvE [10], RSN [20] and once again AnyBURL [42]), but they all show terrible results in symmetric relations. This can be explained by the sampling policy, that involves removing training facts connecting two entities when they are already linked in the test set: given any test fact $\langle h, r, t \rangle$, even when r is symmetric models can never see in training $\langle t, r, h \rangle$.

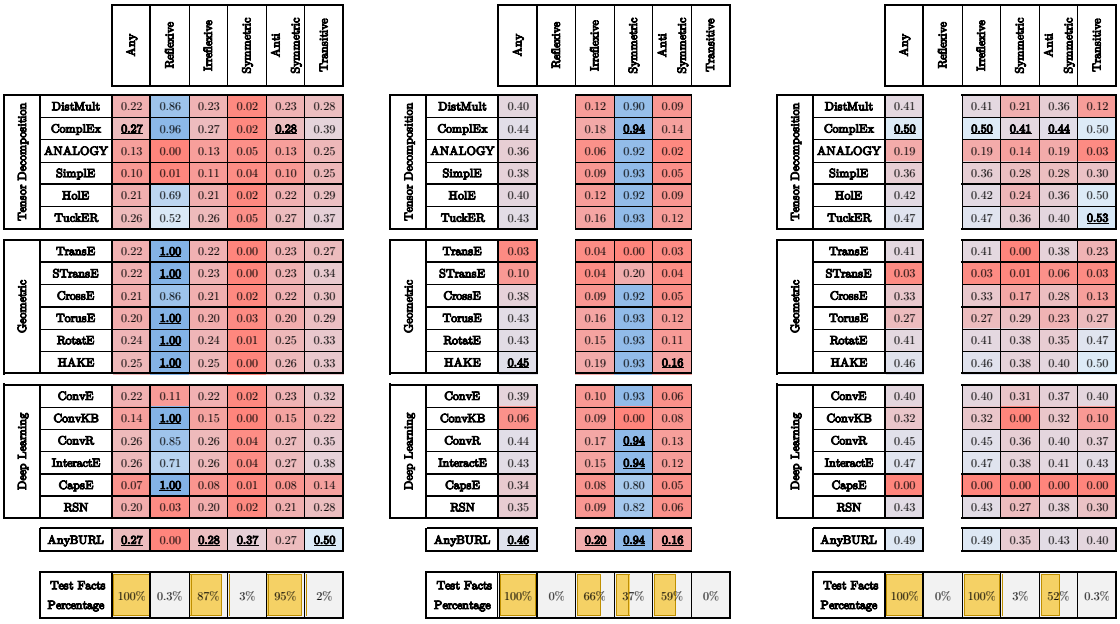
In WN18 and WN18RR we observe a rather different scenario, with most models easily handling symmetric relations (with the exceptions of TransE [6] and ConvKB [48]). Contrary to FreeBase-derived datasets, in the training sets of WordNet-derived datasets symmetric relations are very common and display exceptionally high degrees of symmetry (that is, they very frequently connect two entities in both directions). This can make them easier to learn and predict, thus explaining such a good performance. While in WN18 models tend to achieve very good performance on anti-symmetric and irreflexive relations too, on WN18RR their results on such relation types are always poor. We believe the reason for this difference lies in how these datasets have been built. Among the 18 relations featured in WN18, 4 are symmetric while the remaining 14 form 7 direct-inverse couples. This means that in WN18 each non-symmetric relation has an inverse, resulting in a huge boost to all models. WN18RR was built by filtering away all the inverse relations from WN18, while keeping the symmetric ones untouched. This may seem a fitting solution for the test leakage issue, but the resulting 11 relations appear hardly logically related to one another. The only relations that models can easily handle now are the 4 symmetric ones, because symmetric relations are their own inverse; the remaining 7 relations, on the contrary, often appear almost unpredictable.

In YAGO3-10 all models display H@1 values on anti-symmetric relations comparable to their global ones, albeit generally a little worse. When it comes to symmetric relations their performance usually undergoes further degradation; this phenomenon is particularly severe in TransE [6], ConvKB [48], CrossE [78] and DistMult [77].

Summary: When dealing with different relation types, even fully expressive models can show very different predictive performance. In addition to that, different models may also achieve very different performance on the same relation type, independently from their global metrics.

5.4 Reified Relation Degree

Our goal in these experiments is to analyze how, in FreeBase-derived datasets, the S2C conversion of CVTs into cliques affects predictive performance. S2C replaces CVTs with rich, redundant structures, so we hypothesize that high-degree CVTs result in easier facts. We thus split test facts into disjoint buckets based on the degree of their original CVT, identified as reported in Section 4. We include an additional bucket marked with degree value 1 for the test facts not



(a) FB15k-237 (b) WN18RR (c) YAGO3-10

Fig. 10. H@1 results for each LP model on FB15k-237, WN18RR and YAGO3-10 datasets and corresponding percentages of test facts, for various relation properties. The best results for each column are in bold and underlined.

originated CVTs. We evaluate each model on each bucket, reporting H@1 results in Figures 11a and 11b, and H@10 results in Appendix F; we also show, for each bucket, the percentage of test facts it contains.

In both datasets, higher CVT degrees generally correspond to better results. With the strict H@1 metric the trend is not fully visible in FB15k-237, where most of the redundant S2C-generated facts have been filtered away. In FB15k the pattern is slightly more evident, with the exception of translational models TransE [6], CrossE [78] and STransE [47].

With the more permissive H@10 metric the trend becomes more visible in both datasets. Most notably, in FB15k it now applies to translational models too. To explain this, we point out that S2C often result in many facts featuring symmetric relations. Translational models are naturally inclined to embed symmetric relations into vectors with very low norm: as a consequence, a set of entities connected through symmetric edges will be placed very close to each other in the embedding space. The result would be a crowded subspace in which the the correct target is often outranked when it comes to H@1, but manages to make it to the top K answers for larger values of K.

Summary: The higher the degree of the original reified nodes, the richer the corresponding cliques in the dataset, and thus the better the performance. This trend is particularly visible when using permissive versions of H@K.

5.5 Sensitivity to tie-breaking policy

When a model is unhealthily prone to giving identical scores to multiple entities in the same prediction, different tie-breaking policies produce diverging, incomparable results. In an extreme scenario, a model could systematically give the same score to all entities, obtaining *MRR* equal to 1.0 under *min* policy and tending to 0.0 under *avg* policy.

		Degree of original reified relation					
		1	2-4	5-8	9-16	17-32	> 32
Tensor Decomposition	DistMult	0.64	0.76	0.82	0.82	0.78	0.84
	ComplEx	**0.76**	0.84	**0.88**	**0.89**	**0.88**	**0.90**
	ANALOGY	0.59	0.68	0.73	0.69	0.66	0.70
	SimplE	0.55	0.67	0.76	0.76	0.77	0.78
	HolE	0.68	0.78	0.84	0.82	0.78	0.83
	TuckER	0.70	0.72	0.74	0.77	0.73	0.78
Geometric	TransE	0.45	0.53	0.58	0.50	0.38	0.49
	STransE	0.38	0.40	0.43	0.40	0.30	0.42
	CrossE	0.57	0.64	0.68	0.62	0.54	0.52
	TorusE	0.62	0.68	0.76	0.76	0.74	0.81
	RotatE	0.67	0.76	0.82	0.84	0.79	0.78
	HAKE	0.68	0.76	0.82	0.82	0.79	0.81
Deep Learning	ConvE	0.57	0.56	0.62	0.61	0.56	0.69
	ConvKB	0.10	0.11	0.09	0.08	0.09	0.21
	ConvR	0.66	0.72	0.73	0.74	0.72	0.76
	InteractE	0.70	0.73	0.74	0.78	0.72	0.79
	CapsE	0.02	0.02	0.00	0.00	0.00	0.04
	RSN	0.65	0.78	0.76	0.71	0.68	0.81
	AnyBURL	0.73	**0.86**	0.86	0.88	**0.88**	**0.90**
	Test Facts Percentage	39%	25%	14%	5%	3%	14%

(a) FB15k

		Degree of original reified relation					
		1	2-4	5-8	9-16	17-32	> 32
Tensor Decomposition	DistMult	0.23	0.19	0.20	0.24	0.26	0.24
	ComplEx	**0.28**	**0.24**	0.28	0.24	0.32	0.28
	ANALOGY	0.12	0.13	0.13	0.12	0.11	0.15
	SimplE	0.09	0.12	0.13	0.13	0.20	0.16
	HolE	0.22	0.18	0.19	0.18	0.17	0.22
	TuckER	0.26	0.23	0.25	0.22	0.30	0.26
Geometric	TransE	0.22	0.18	0.20	0.20	0.23	0.22
	STransE	0.23	0.20	0.21	0.19	0.24	0.20
	CrossE	0.22	0.19	0.20	0.16	0.22	0.20
	TorusE	0.20	0.16	0.18	0.16	0.16	0.20
	RotatE	0.24	0.20	0.26	0.22	0.26	0.26
	HAKE	0.26	0.22	0.28	**0.27**	0.24	0.25
Deep Learning	ConvE	0.23	0.19	0.20	0.20	0.26	0.22
	ConvKB	0.15	0.11	0.12	0.12	0.18	0.17
	ConvR	0.26	0.22	0.24	0.22	0.28	0.26
	InteractE	0.27	**0.24**	0.26	0.24	0.26	0.27
	CapsE	0.08	0.06	0.04	0.04	0.09	0.08
	RSN	0.20	0.16	0.19	0.16	0.23	0.21
	AnyBURL	0.26	**0.24**	**0.29**	0.26	**0.35**	**0.43**
	Test Facts Percentage	74%	14%	4%	1%	1%	6%

(b) FB15k-237

Fig. 11. H@1 results for each LP model on the **Freebase** datasets, and corresponding distribution of test facts, varying the degree of the original reified relation in FreeBase. The best results for each column are marked in bold and underlined.

We observe that ConvKB [48], CapsE [49], and to a lesser extent CrossE [78] seem strikingly sensitive to tie-breaking policies; ConvKB and CapsE use *min* policy by default, whereas CrossE uses *ordinal*. When switching to *avg*, ConvKB and CapsE show huge differences across all metrics in FB15k and FB15k-237; on WN18 and WN18RR their behaviour is rather stable. On YAGO3-10 ConvKB is stable too, whereas CapsE shows the most extreme discrepancy, with its MRR going from 1.0 to 0.0. CrossE, when switching to *min* policy, only shows noticeable differences in YAGO3-10.

We find strong experimental evidence that *saturating activation functions* may facilitate these behaviours. Saturating functions yield the same result for inputs beyond or below a threshold, e.g. the ReLU function returns 0 for any input lesser or equal to 0. Intuitively, this can lead models to give identical scores to different entities, causing the observed issues. Both ConvKB and CapsE use ReLUs between their layers; we verify our hypothesis by replacing them with Leaky ReLUs, a non-saturating alternative that keeps a small slope *alpha* for inputs lesser than 0 (we used $\alpha = 0.2$). We include in Table 4 the results for our CapsE "Leaky" variation: the differences between *min* and *avg* policies are now much less prominent, becoming barely observable in FB15k and FB15k-237. CrossE does not rely on inherently saturating functions, but it applies a final sigmoid function to project scores to $[0, 1]$. For large values the slope of the sigmoid is almost nil, so low-level approximations make it *de facto* saturating. We verify that removing the sigmoid in evaluation makes any sensitivity to tie-breaking policies disappear; being the sigmoid monotonous and growing, this does not affect ranks. We include these results too in Table 4.

Summary: Different tie-breaking policies yield incomparable results when models give the same scores to multiple entities in the same predictions. Saturating activation functions appear to be most responsible for this behaviour.

		FB15k				WN18				FB15k-237				WN18RR				YAGO3-10			
		H@1	**H@10**	**MR**	**MRR**	**H@1**	**H@10**	**MR**	**MRR**	**H@1**	**H@10**	**MR**	**MRR**	**H@1**	**H@10**	**MR**	**MRR**	**H@1**	**H@10**	**MR**	**MRR**
ConvKB	**min**	52.86	40.83	324	0.211	52.89	94.89	202	0.709	34.97	52.74	246	0.407	5.63	53.04	1694	0.251	32.16	60.47	1682	0.420
	avg	11.43	40.83	324	0.211	52.89	94.89	202	0.709	13.98	41.46	309	0.230	5.63	52.50	3429	0.249	32.16	60.47	1683	0.420
CapsE	**min**	73.01	73.93	364	0.735	84.55	95.08	233	0.890	49.19	52.87	302	0.526	33.69	55.98	720	0.415	100.0	100.0	1	1.0
	avg	1.93	21.78	610	0.087	84.55	95.08	233	0.890	7.34	35.59	405	0.160	33.69	55.98	720	0.415	0.00	0.00	60676	0.000
CapsE "leaky"	**min**	18.53	50.31	194	0.290	86.00	95.08	232	0.898	12.52	35.75	360	0.201	31.22	55.61	738	0.401	26.82	26.82	1568	0.269
	avg	18.43	50.41	194	0.289	86.00	95.08	232	0.898	12.23	35.61	360	0.199	31.22	55.61	738	0.401	0.00	0.00	32972	0.000
CrossE with sigmoid	**min**	60.37	86.37	33	0.704	73.28	95.11	74	0.834	21.21	47.05	214	0.298	38.07	44.99	4165	0.405	38.65	65.95	363	0.482
	avg	60.08	86.23	136	0.702	73.28	95.03	441	0.834	21.21	47.05	227	0.298	38.07	44.99	5212	0.405	33.09	65.45	3839	0.446
CrossE without sigmoid	**min**	60.16	86.23	50	0.703	73.28	95.05	320	0.834	21.21	47.05	223	0.298	38.07	44.99	4927	0.405	33.40	65.63	1316	0.448
	avg	60.16	86.23	50	0.703	73.28	95.05	320	0.834	21.21	47.05	223	0.298	38.07	44.99	4927	0.405	33.40	65.45	1316	0.446

Table 4. Results obtained with *average* or *ordinal* tie-breaking policy (**avg**) against results obtained with *min* tie-breaking policy (**min**). The table features all the models for which these results show discrepancies (ConvKB; CapsE; CrossE), and the corresponding experiments (CapsE "Leaky"; CrossE without sigmoid).

6 KEY TAKEAWAYS AND RESEARCH DIRECTIONS

In this section we summarize the key takeaways from our comparative analysis. We believe these lessons can inform and inspire future research on LP models based on KG embeddings.

6.1 General considerations and guidelines

In this section we provide some guidelines for choosing which LP model to use when dealing with a new KG.

Unless working on KGs with very peculiar properties, ComplEx [66] in its recent implementation with N3 regularization [33] is always an excellent choice. This system is easily the best-performing of all embedding-based models across most datasets and metrics, so it is probably the first model to try in any situation.

If one has severe limitations in terms of training time, and/or predictive performance is not particularly important (e.g. the embeddings are just used to perform downstream tasks), models that require very little time to train are maybe preferable, such as Analogy [38] or TorusE [12]. We also point out that tensor factorization models and geometric models tend to be overall lighter than models based on deep neural networks when it comes to required storage.

When dealing with very large KGs, the model that seems to scale best against increasing dataset size is HAKE [80]; its performance is very good too, despite still remaining quite behind ComplEx-N3 [33] in most datasets.

We stress that if one is only interested in predictions and does not require the use of KG embeddings, the rule-based model AnyBURL [42] - especially in its recent reinforced implementation [43] - is a very competitive choice that should definitely be taken into account. AnyBURL is the only model in our analysis which manages to approach ComplEx-N3 [33] in most scenarios; its training times are very fast, and its prediction times are quite good too unless

full prediction rankings are required. Furthermore, being a rule-based system it is inherently explainable, whereas embedding-based models can be quite obscure in this regard.

6.2 Effect of the design choices

We report here our findings and trends investing entire families, showing how the architectural design of LP models can affect performance and robustness across datasets and metrics.

Tensor Decomposition models, among the families included in our analysis, show the most solid results across datasets. Including the amazing results displayed by ComplEx-N3 and already discussed in 6.2, most of these systems display uniform performance across all evaluation metrics and datasets (with the potential exceptions of ANALOGY [38] and SimplE [30], seemingly more fluctuating).

Geometric models, on the other hand, show slightly less stable results. In the past years, researchers have devoted considerable effort into translational systems, trying to overcome the limitations of TransE [6] by the introduction of additional embeddings. These models show interesting results, but still display irregularities across metrics and datasets, e.g. TransE [6], STransE [47] and to a lesser extent even CrossE [78] seem to struggle on the WN18RR dataset. All in all, models relying solely on translations seem to have been outclassed by recent roto-translational ones; in this regard, RotatE [59] and HAKE [80] show remarkably consistent performance across all datasets.

Deep Learning models, finally, are the most diverse family, with wildly different results depending on the architectural choices of the models and on their implementations. InteractE [68], ConvR [28] and RSN [20] display the best results in this family, achieving similar performance on FB15k, WN18 and YAGO3-10. In FB15k-237 and WN18RR, whose filtering processes have cut away the most relevant paths, RSN seems to have a harder time, probably due to its formulation that explicitly leverages paths. Models such as ConvKB [48] and CapsE [49] achieve promising results in terms of H@10 and MR but seem to struggle with H@1 and MRR, potentially hindered by their issues with tie-breaking policies described in Section 5.5.

6.3 The importance of the graph structure

We report here our main considerations on the influence of graph structural features on what models manage to learn and predict; our findings are supported by the extensive experimental evidence reported in Section 5.

Source Peers and Target Peers produce evidently antagonistic effects on prediction performance across all models and datasets. As already mentioned, source peers work as examples that allow models to better characterize the relation and the target to predict, whereas target peers lead models to confusion, as they try to optimize embeddings to fit too many different answers for the same question.

Relational Paths play a central role in providing relational patterns that all models leverage, even those that only learn individual facts in training. Interestingly, the support provided by relational paths seem more beneficial to performance in WN18, WN18RR and YAGO3-10, where the relevance of peers is less prominent. On the other hand, in FreeBase-derived datasets, where the effect of peers seems more evident, RPS seems less impactful (but beneficial nonetheless). When it comes to relational paths, we also observe that in FB15k and WN18, due to the presence of inverse and equivalent relations, performance correlates best with the support of short paths with length 1 or 2. In other words, in presence of very simple patterns providing overwhelming evidence, models are prone to just focusing on them, and to disregard more complex reasoning. This can be seen as an unhealthy consequence of the test leakage problem. In FB15k-237, WN18RR and YAGO3-10, on the other hand, models significantly rely on paths of length 3 too, meaning that they are more inclined to investigate to a certain extent longer dependencies.

Relation Types can lead to very different performance, even in fully expressive models. For instance, in WN18RR most models seem capable at handling almost perfectly all symmetric relations, while their performance is always very poor on antisymmetric and irreflexive ones. Different models also display different behaviour with specific relation types: despite their good results in terms of H@K and MRR, models such as RSN [20], ConvE [10] and even AnyBURL [42] have problems with reflexive relations in FreeBase-derived datasets, while the other models - even when achieving much worse global metrics - manage to predict the almost perfectly.

The original degree of relations in datasets generated applying S2C explosion has an observable effect in performance too. This proves that the redundant edges added to the graph, together with the loss of complexity resulting from the subsequent deduplication, can artificially boost the predictions of LP models. We believe that, in order to fully assess this condition, it would be fruitful to extract novel versions of FB15k and FB15k-237 in their original reified structure without applying S2C.

We also acknowledge that handling n-ary relations in LP systems is still an open problem in research. As noted by Fatemi et al. [16] and Wen et al. [75], 61% of the FreeBase relations are n-ary, and the corresponding facts involve more than 1/3rd of the FreeBase entities. Since some KGs are implemented as hyper-graphs, a few models have been designed to explicitly learn hyper-edges to perform the LP task. Notable examples are m-TransH [75] (a TransH [74] extension), and the recent HypE [16], based on a novel convolutional architecture. Their authors have also undergone significant efforts to produce original benchmarking datasets featuring hyper-edges rather than binary links, such as the JF17K graphs, FB-Auto and m-FB15K. Since graphs that only feature binary relations can be seen as a special case of hyper-graphs, these models can technically be used to perform LP on them too. Nonetheless, the unique features of such models emerge when dealing with relations beyond binary. As a consequence we do not include them in our comparative analysis, since it focuses on LP on KGs with binary relations only.

6.4 The importance of tie-breaking policies

We report that differences in the policies used to handle score ties can lead to huge differences in predictive performance in evaluation. As a matter of fact, such policies are today regarded as almost negligible implementation details, and they are hardly ever even reported when presenting novel LP models. Nevertheless, we show that results computed relying on different policies risk to be not directly comparable to one another, and might not even reflect the actual effectiveness of models. Therefore we strongly advise researchers to use the same policy and clearly report it in their works in the future; in our opinion, the *average* policy seems the most reasonable choice. We also find strong experimental evidence that saturating activation functions, such as ReLU, play a key role in leading models to assign the same scores to multiple entities in the same prediction; approximations may also lead non-saturating functions, such as sigmoid, behave as saturating in regions where their slope is particularly close to 0.

6.5 Considerations on the rule-based baseline

As already mentioned, the rule-based baseline AnyBURL proves to be remarkably easy to train and well-performing; apart from some issues with symmetric relations in FreeBase-derived datasets, it consistently surpasses most latent models, with the notable exception of ComplEx-N3 [33].

This may lead one to argue that researching embedding-based systems is irrelevant, since rule-based approaches prove to be equivalent or better than most of them. We do not subscribe to this point of view. KG embeddings are a relatively young technology that in a very short amount of time has achieved state of the art performance in LP; in some cases, their rise has even been the drive to develop better rule-based models, as it happened with AnyBURL[42].

Under these premises, and with a sparkling community producing novel advancements every year, they constitute an exceptionally promising research topic, and a formidable resource for tackling KG incompleteness.

Furthermore, KG embeddings learned with the goal of LP can be successfully employed in a wider variety of downstream tasks, ranging from KG alignment [64] to fact checking [27]; this makes them versatile tools, in the same way in which word embeddings are currently used in numerous NLP activities.

At the same time, we do not see rule-based models becoming obsolete due to the rise of KG embeddings anytime soon. The different paradigm of these systems gives them features that make them preferable, or in some cases even mandatory, in several scenarios both in research and in industrial settings. For instance, rule-based models are inherently transparent, whereas embedding-based models are still rather obscure in their predictions; furthermore, rule-based models may always remain less demanding than embedding-based ones in terms of training times and model size.

7 RELATED WORK

Works related to ours can be roughly divided into two main categories: *analyses* and *surveys*. Analyses deal with studies trying to convey deeper understandings on LP models and are backed by further experiments, whereas surveys usually attempt to organize them into comprehensive taxonomies based on their known features and capabilities.

Analyses. Wang *et al.* [72] provide a critique on the current benchmarking practices. They observe that current evaluation practices only compute the rankings for test facts; therefore, we are only verifying that, when a question is "meaningful" and has answers, our models prioritize the correct ones over the wrong ones. This amounts to performing question answering rather than KG completion, because we are not making sure that questions with no answers (and therefore not in the dataset) result in low scores. Therefore, they propose a novel evaluation pipeline, called Entity-Pair Ranking (PR) including all possible combinations in $\mathcal{E} \times \mathcal{R} \times \mathcal{E}$. We wholly agree with their observations; unfortunately, we have found that for our experiments, where the full ranking for all predictions is required for all models in all datasets, PR evaluation is way too time-consuming and thus unfeasible.

Kadlec *et al.* [29] demonstrate that a carefully tuned implementation of DistMult [77] can achieve state-of-the-art performances, surpassing most of its own successors, raising questions on whether we are developing better LP models or just tuning better hyperparameters.

Tran *et al.* [62] interpret 4 models based on matrix factorization as special cases of the same multi-embedding interaction mechanism. In their formulation, each KG element k is expressed as a set of vectors $\{\boldsymbol{k}^{(1)}, \boldsymbol{k}^{(2)}, ..., \boldsymbol{k}^{(n)}\}$; the scoring functions combine such vectors using trilinear products. The authors also include empirical analyses and comparisons among said models, and introduce a new multi-embedding one based on quaternion algebra.

Rossi and Matinata [56] observe that the most popular LP datasets display wildly skewed distributions in regard to entity degrees. They show how this can affect predictive performances, and question the relevance, fairness, and capability to highlight overfitting of the current LP benchmarking pipelines.

A promising sub-category of analyses focuses on the geometrical properties of the scoring functions and of the learned embeddings in the latent space, with the goal of unveiling the inner mechanisms of LP models.

In this regard, Gutiérrez-Basulto *et al.* [22] investigate how different scoring functions affect which logical patterns can be learned by models. The authors interpret logical dependencies as sets of existential rules embodying the ontology knowledge, and adopt a geometric interpretation of relations as regions in the latent space. They show that, despite many of them being fully expressive, all bilinear models suffer from severe restrictions in the types of rules they can learn. Furthermore, they propose a framework in which relations are represented as arbitrary convex regions, and

demonstrate that this makes it possible to successfully capture quasi-chained existential rules. They finally identify the further generalizations required to faithfully represent arbitrary existential rules.

Chandrahas *et al.* [8] study how geometric features of the learned embeddings reflect the model family and training settings, and relate to the overall predictive performances. They introduce a novel property, called *conicity*, to measure the sparsity of the embeddings, and use it in combination to *Alignment To Mean* (ATM), *Average Vector Length* (AVL) and *Vector Spred* (VS). They take into account additive and multiplicative models, and they find that the former tend to learn significantly sparser embeddings than the latter. Variations in the number of negative samples and embedding size do not affect conicity in additive models, whereas, in multiplicative ones, they can lead to significantly denser or sparser vectors, or even produce opposite effects on entity and relation embeddings. All in all, the authors observe that vector sparsity does not seem to correlate with performances in additive models, while a low conicity in entity embeddings correlates with in better results in multiplicative ones.

We finally report that some works have started to investigate the interpretability of KG embeddings. Despite not constituting actual analyses on LP models, these approaches share their same goal, aiming at explaining the learned vectors and the resulting predictions; we strongly support this research direction, as it may prove invaluable to gain further insights on what our systems are actually learning, and on how they can be improved in this regard.

Zhang et al. [78] provide explanations to any fact $\langle h, r, t \rangle$ in the form of paths connecting h to t. Their approach leverages the graph topology to extract meaningful paths, based on a support metric that estimates how often similar structures are observed elsewhere in the graph. In order to reduce computational complexity, this method only takes into account paths starting with relations similar to r, and entities similar to t.

Following a different approach, Pezeshkpour et al. [55] define explanations as the training facts that have been most influential for the prediction to interpret. Their CRIAGE approach is based on influence functions, proposing an approximation based on the Taylor series to make them computationally feasible; unfortunately such a formulation only applies to models in which the scoring function can be expressed in the form $f(\boldsymbol{h}, \boldsymbol{r})\boldsymbol{st}$.

All the above mentioned analyses have a very different scope from ours and, thus, complement it. Their goal is generally to address specific issues or investigate vertical hypotheses; on the other hand, our objective is to run an extensive comparison of models belonging to vastly different families, investigating the effects of distinct design choices, discussing the effects of different benchmarking practices and underlining the importance of the graph structure.

Surveys. Nickel *et al.* [51] provide an overview for the most popular techniques in the whole field of Statistic Relational Learning, to which LP belongs. The authors include both traditional approaches based on observable graph features and more recent ones based on latent features. Since the paper has been published, however, a great deal of further progress has been made in KG Embeddings.

Cai *et al.* [7] provide a survey for the whole Graph Embedding field. Their scope is not limited to KGs: on the contrary, they overview models handling a wide variety of graphs (Homogeneous, Heterogeneous, with Auxiliary Information, Constructed from Non-Relational Data) with an even wider variety of techniques. Some KG embedding models are briefly discussed in a section dedicated to models that minimize margin-based ranking loss.

The surveys by Wang *et al.* [71] and by Nguyen [46] are the most relevant to our work, as they specifically focus on KG Embedding methods. In the work by Wang *et al.* [71], models are first coarsely grouped based on the input data they rely on (facts only; relation paths; textual contents; etc); the resulting groups undergo further finer-grained selection, taking into account for instance the nature of their scoring functions (e.g. distance-based or semantic-matching-based). What's more, they offer detailed descriptions for each of the models they encompass, explicitly stating its architectural

peculiarities as well as its space and time complexities. Finally, they take into account a large variety of applications that the Knowledge Graph Embedding models can support. The work by Nguyen [46] is similar, albeit more concise, and also includes current state-of-the-art methods such as RotatE [59].

Our work is fundamentally different from these surveys: while they only report results available in the original papers, we design and conduct experiments to extensively investigate the empirical behaviours of models. As discussed in Section 1, results reported in the original papers are generally obtained in very different settings and they are generally global metrics on the whole test sets; as a consequence, it is difficult to interpret and compare them.

8 CONCLUSIONS

In this work we present the first extensive comparative analysis on LP models based on KG embeddings. We survey 18 well known LP models representative of diverse techniques and architectures. We analyze their efficiency and effectiveness on the 5 most popular datasets in literature, and adopt an additional rule-based model as a baseline.

Our study is mainly devoted to analyze the deeper behaviours of LP models, investigating which circumstances enable models to perform satisfactorily, and which ones on the other hand hinder their capabilities of relational reasoning.

We thoroughly investigate the effects of graph structural features on LP models, and find that they play paramount effects on their effectiveness. We demonstrate that source and target peers have strong antagonistic influences on performances, and we show how all LP models - even those that learn training facts individually - largely benefit from the logical patterns provided by relational paths. We further study how even fully expressive models can display very different performances when dealing with specific relation types, and show that the policies used when extracting the current LP benchmarking dataset can result in subtle, unforeseen consequences.

We finally verify how different low-level policies can yield incomparable and misleading results, and we identify which components can make models most sensitive to such policies.

All in all, we identify intriguing behavioural differences that the traditional benchmarking practices, focused on global metrics, would not reveal. The takeaways from our analysis offer a clear picture of which problems have been solved, and which areas still have room for improvement.

We are confident that our findings can inspire and inform future research to overcome the limitations of current LP models and benchmarks.

ACKNOWLEDGMENTS

We thank Simone Scardapane, Matteo Cannaviccio and Alessandro Temperoni for their insightful discussions. We heartfully thank the authors of AnyBURL, ComplEx-N3, ConvR, CrossE, HAKE, InteractE and RSN for their for their amazing support and guidance on their models.

REFERENCES

[1] N. Ahmadi, V. Huynh, V. V. Meduri, S. Ortona, and P. Papotti. Mining Expressive Rules in Knowledge Graphs. *J. Data and Information Quality*, pages 8:1–8:27, 2020.
[2] B. An, B. Chen, X. Han, and L. Sun. Accurate text-enhanced knowledge graph representation learning. In *NAACL-HLT*, 2018.
[3] 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.
[4] I. Balazevic, C. Allen, and T. M. Hospedales. TuckER: Tensor Factorization for Knowledge Graph Completion. In *EMNLP - IJCNLP*, 2019.
[5] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, 2008.
[6] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.
[7] H. Cai, V. W. Zheng, and K. Chang. A comprehensive survey of graph embedding: problems, techniques and applications. *TKDE*, 2018.

[8] Chandrahas, A. Sharma, and P. P. Talukdar. Towards Understanding the Geometry of Knowledge Graph Embeddings. In *ACL*, 2018.

[9] L. Costabello, S. Pai, C. L. Van, R. McGrath, N. McCarthy, and P. Tabacof. AmpliGraph: a Library for Representation Learning on Knowledge Graphs, 2019. URL https://doi.org/10.5281/zenodo.2595043. [Online; accessed 10-October-2019].

[10] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, 2018.

[11] X. L. 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 *SIGKDD*, 2014.

[12] T. Ebisu and R. Ichise. TorusE: Knowledge Graph Embedding on a Lie Group. In *AAAI*, 2018.

[13] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[14] Facebook Research. Knowledge base completion (kbc). https://github.com/facebookresearch/kbc. [Online; accessed 10-October-2019].

[15] B. Fatemi. A faster SimplE implementation. https://github.com/baharefatemi/SimplE. [Online; accessed 10-October-2019].

[16] B. Fatemi, P. Taslakian, D. Vázquez, and D. Poole. Knowledge Hypergraphs: Extending Knowledge Graphs Beyond Binary Relations. *CoRR*, abs/1906.00137, 2019.

[17] L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*, 2013.

[18] L. A. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek. Fast rule mining in ontological knowledge bases with amie+. *VLDB Journal*, 24(6), 2015.

[19] G. A. Gesese, R. Biswas, and H. Sack. A Comprehensive Survey of Knowledge Graph Embeddings with Literals: Techniques and Applications. In *DL4KG*, 2019.

[20] L. Guo, Z. Sun, and W. Hu. Learning to Exploit Long-term Relational Dependencies in Knowledge Graphs. In *ICML*, 2019.

[21] S. Guo, Q. Wang, L. Wang, B. Wang, and L. Guo. Knowledge graph embedding with iterative guidance from soft rules. In *AAAI*, 2018.

[22] V. Gutiérrez-Basulto and S. Schockaert. From Knowledge Graph Embedding to Ontology Embedding? An Analysis of the Compatibility between Vector Space Representations and Rules. In *KR*, 2018.

[23] K. Hayashi and M. Shimbo. On the Equivalence of Holographic and Complex Embeddings for Link Prediction. In *ACL*, 2017.

[24] F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 1927.

[25] J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 1982.

[26] E. Hovy, R. Navigli, and S. P. Ponzetto. Collaboratively Built Semi-structured Content and Artificial Intelligence: The Story So Far. *Artif. Intell.*, 194: 2–27, 2013.

[27] V. Huynh and P. Papotti. A Benchmark for Fact Checking Algorithms Built on Knowledge Bases. In *CIKM*, 2019.

[28] X. Jiang, Q. Wang, and B. Wang. Adaptive Convolution for Multi-Relational Learning. In *NAACL-HLT*, 2019.

[29] R. Kadlec, O. Bajgar, and J. Kleindienst. Knowledge Base Completion: Baselines Strike Back. In *Rep4NLP@ACL*, 2017.

[30] S. M. Kazemi and D. Poole. SimplE Embedding for Link Prediction in Knowledge Graphs. In *NIPS*, 2018.

[31] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM review*, 2009.

[32] S. Kostadinov. *Recurrent Neural Networks with Python Quick Start Guide: Sequential learning and language modeling with TensorFlow*. Packt Publishing Ltd, 2018.

[33] T. Lacroix, N. Usunier, and G. Obozinski. Canonical Tensor Decomposition for Knowledge Base Completion. In *ICML*, 2018.

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

[35] N. Lao, T. Mitchell, and W. W. Cohen. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, 2011.

[36] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[37] Y. Lin, Z. Liu, H. Luan, M. Sun, S. Rao, and S. Liu. Modeling Relation Paths for Representation Learning of Knowledge Bases. In *EMNLP*, 2015.

[38] H. Liu, Y. Wu, and Y. Yang. Analogical Inference for Multi-relational Embeddings. In *ICML*, 2017.

[39] F. Mahdisoltani, J. Biega, and F. M. Suchanek. YAGO3: A knowledge base from multilingual wikipedias. In *CIDR*, 2015.

[40] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[41] C. Meilicke, M. Fink, Y. Wang, D. Ruffinelli, R. Gemulla, and H. Stuckenschmidt. Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion. In *ISWC*, 2018.

[42] C. Meilicke, M. W. Chekol, D. Ruffinelli, and H. Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *IJCAI*, 2019.

[43] C. Meilicke, M. W. Chekol, M. Fink, and H. Stuckenschmidt. Reinforced Anytime Bottom Up Rule Learning for Knowledge Graph Completion. *arXiv preprint arXiv:2004.04412*, 2020.

[44] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. In *ICLR*, 2013.

[45] S. K. Mohamed, V. Novácek, P. Vandenbussche, and E. Muñoz. Loss Functions in Knowledge Graph Embedding Models. In *DL4KG*, 2019.

[46] D. Q. Nguyen. An overview of embedding models of entities and relationships for knowledge base completion. *CoRR*, abs/1703.08098, 2017.

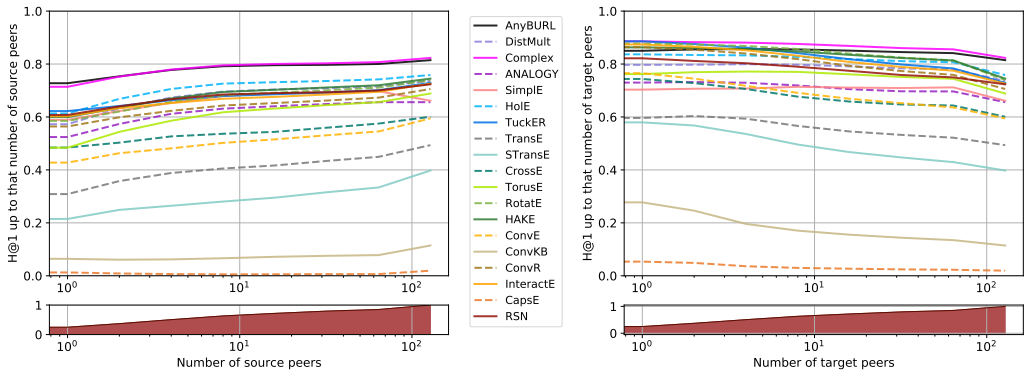
[47] D. Q. Nguyen, K. Sirts, L. Qu, and M. Johnson. STransE: a novel embedding model of entities and relationships in knowledge bases. In *NAACL-HLT*, 2016.

[48] D. Q. Nguyen, T. D. Nguyen, D. Q. Nguyen, and D. Q. Phung. A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In *NAACL-HLT*, 2018.

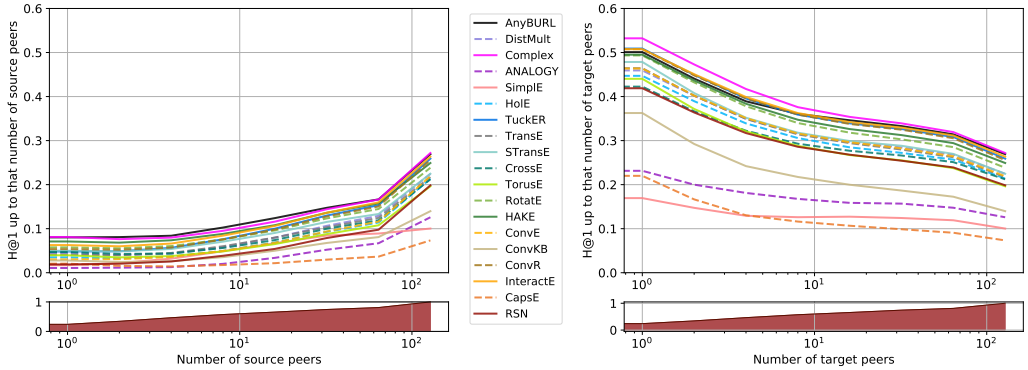
[49] D. Q. Nguyen, T. Vu, T. D. Nguyen, D. Q. Nguyen, and D. Q. Phung. A Capsule Network-based Embedding Model for Knowledge Graph Completion and Search Personalization. In *NAACL-HLT*, 2019.

[50] M. Nickel, V. Tresp, and H.-P. Kriegel. A Three-Way Model for Collective Learning on Multi-Relational Data. In *ICML*, 2011.
[51] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 2015.
[52] M. Nickel, L. Rosasco, and T. A. Poggio. Holographic Embeddings of Knowledge Graphs. In *AAAI*, 2016.
[53] H. Paulheim. Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods. *Semantic web*, 2017.
[54] P. Pezeshkpour, L. Chen, and S. Singh. Embedding Multimodal Relational Data for Knowledge Base Completion. In *EMNLP*, 2018.
[55] P. Pezeshkpour, Y. Tian, and S. Singh. Investigating Robustness and Interpretability of Link Prediction via Adversarial Modifications. In *NAACL-HLT*, 2019.
[56] A. Rossi and A. Matinata. Knowledge Graph Embeddings: Are Relation-Learning Models Learning Relations? In *PIE*, 2020.
[57] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *NIPS*, 2017.
[58] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*. ACM, 2007.
[59] Z. Sun, Z. Deng, J. Nie, and J. Tang. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *ICLR*, 2019.
[60] K. Toutanova and D. Chen. Observed versus latent features for knowledge base and text inference. In *CVSC*, 2015.
[61] K. Toutanova, D. Chen, P. Pantel, H. Poon, P. Choudhury, and M. Gamon. Representing text for joint embedding of text and knowledge bases. In *EMNLP*, 2015.
[62] H. N. Tran and A. Takasu. Analyzing Knowledge Graph Embedding Methods from a Multi-Embedding Interaction Perspective. In *EDBT/ICDT*, 2019.
[63] B. D. Trisedya, J. Qi, and R. Zhang. Entity alignment between knowledge graphs using attribute embeddings. In *AAAI*, 2019.
[64] R. Trivedi, B. Sisman, X. L. Dong, C. Faloutsos, J. Ma, and H. Zha. LinkNBed: Multi-Graph Representation Learning with Entity Linkage. In *ACL*, 2018.
[65] T. Trouillon and M. Nickel. Complex and Holographic Embeddings of Knowledge Graphs: A Comparison. *CoRR*, abs/1707.01475, 2017.
[66] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex Embeddings for Simple Link Prediction. In *ICML*, 2016.
[67] University Mannheim, Data and Web Science Group. AnyBURL. http://web.informatik.uni-mannheim.de/AnyBURL/. [Online; accessed 10-October-2019].
[68] S. Vashishth, S. Sanyal, V. Nitin, N. Agrawal, and P. P. Talukdar. InteractE: Improving Convolution-based Knowledge Graph Embeddings by Increasing Feature Interactions. In *AAAI*, 2020.
[69] E. M. Voorhees. The trec-8 question answering track report. In *TREC*, 1999.
[70] D. Vrandecic and M. Krötzsch. Wikidata: a free collaborative knowledge base. *Commun. ACM*, 57(10):78–85, 2014.
[71] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *TKDE*, 2017.
[72] Y. Wang, D. Ruffinelli, R. Gemulla, S. Broscheit, and C. Meilicke. On Evaluating Embedding Models for Knowledge Base Completion. In *RepL4NLP@ACL*, 2019.
[73] Z. Wang and J.-Z. Li. Text-Enhanced Representation Learning for Knowledge Graph. In *IJCAI*, 2016.
[74] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge Graph Embedding by Translating on Hyperplanes. In *AAAI*, 2014.
[75] J. Wen, J. Li, Y. Mao, S. Chen, and R. Zhang. On the Representation and Embedding of Knowledge Bases beyond Binary Relations. In *IJCAI*, 2016.
[76] R. West, E. Gabrilovich, K. Murphy, S. Sun, R. Gupta, and D. Lin. Knowledge base completion via search-based question answering. In *WWW*, 2014.
[77] B. Yang, W. Yih, X. He, J. Gao, and L. Deng. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *ICLR*, 2015.
[78] W. Zhang, B. Paudel, W. Zhang, A. Bernstein, and H. Chen. Interaction Embeddings for Prediction and Explanation in Knowledge Graphs. In *WSDM*, 2019.
[79] Y. Zhang, Q. Yao, W. Dai, and L. Chen. Autosf: Searching scoring functions for knowledge graph embedding. In *ICDE*. IEEE, 2020.
[80] Z. Zhang, J. Cai, Y. Zhang, and J. Wang. Learning Hierarchy-Aware Knowledge Graph Embeddings for Link Prediction. In *AAAI*, 2020.

A SOURCE PEERS AND TARGET PEERS

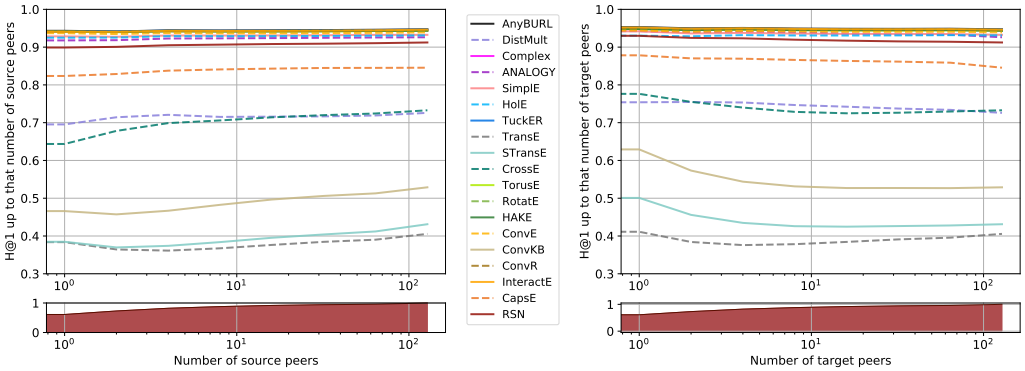


(a) FB15k

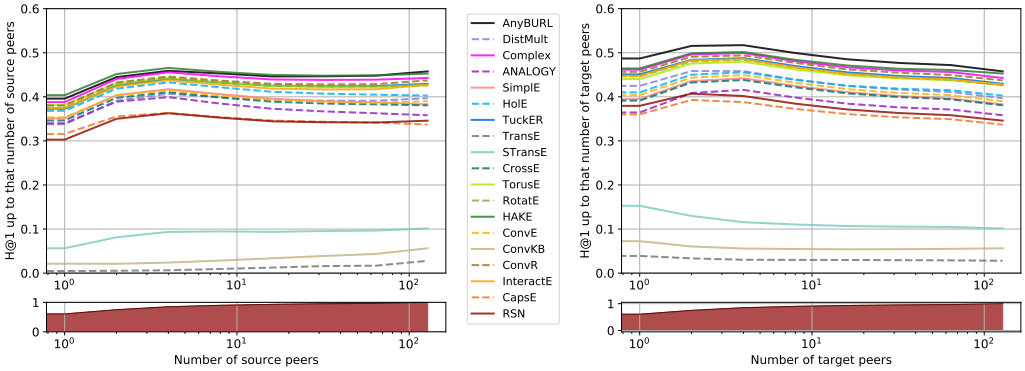


(b) FB15k-237

Fig. 12. Cumulative H@1 results for each LP model on **FreeBase** datasets, and corresponding cumulative distribution of test facts, varying the number of source peers (left) and target peers (right). X axis is in logscale.

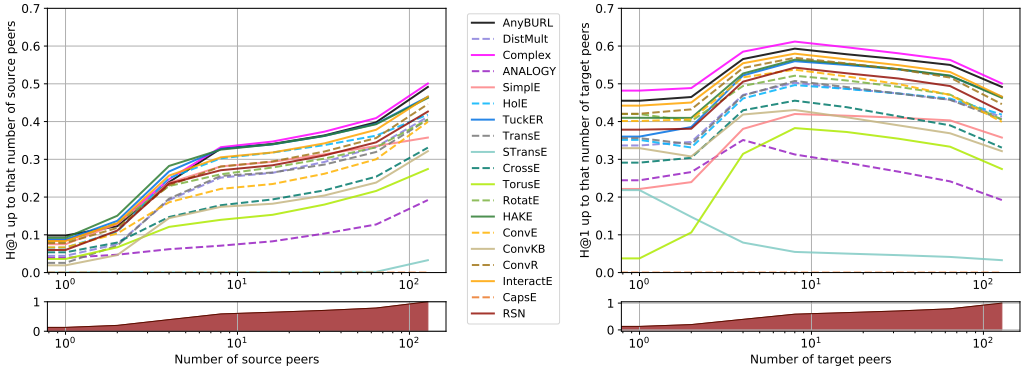


(a) WN18



(b) WN18RR

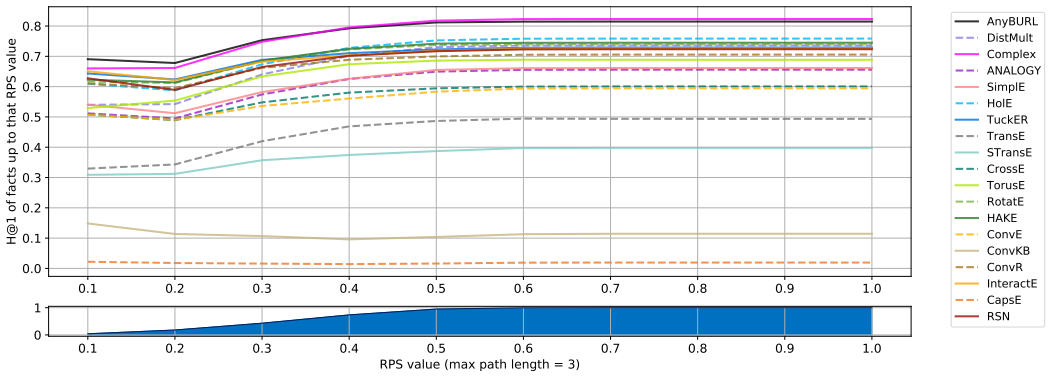
Fig. 13. Cumulative H@1 results for each LP model on WordNet datasets, and corresponding cumulative distribution of test facts, varying the number of source peers (left) and target peers (right). X axis is in logscale.



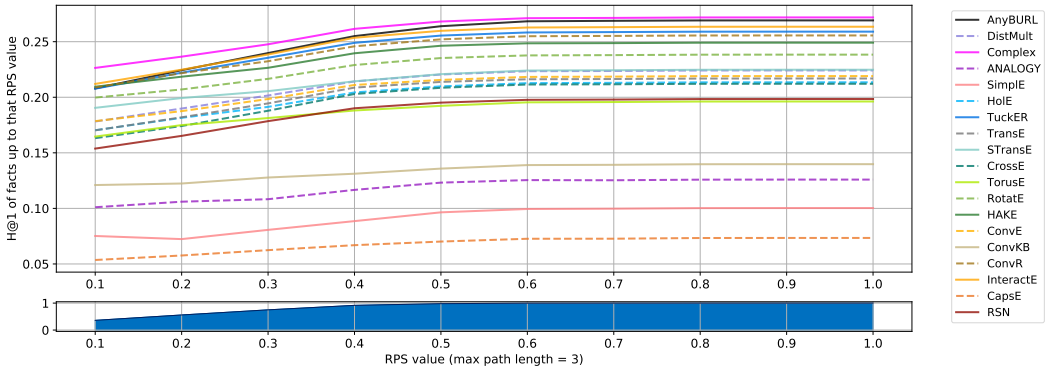
(a) YAGO3-10

Fig. 14. Cumulative H@1 results for each LP model on Yago datasets, and corresponding cumulative distribution of test facts, varying the number of source peers (left) and target peers (right). X axis is in logscale.

B RELATIONAL PATH SUPPORT

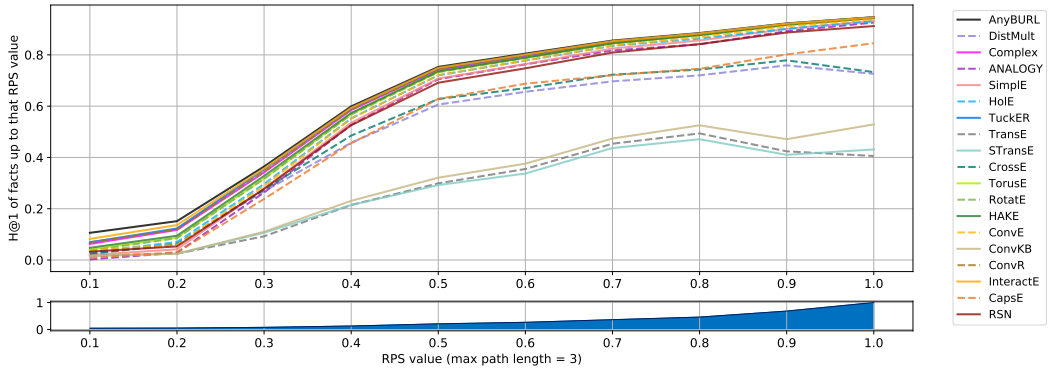


(a) FB15k

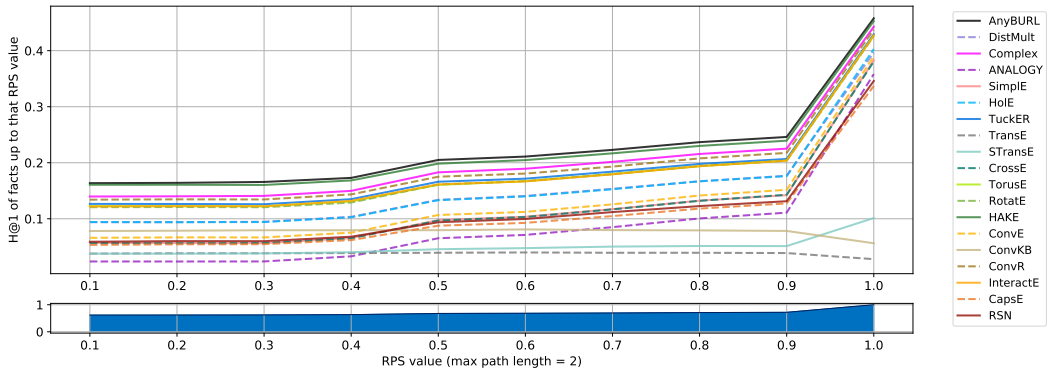


(b) FB15k-237

Fig. 15. Cumulative H@1 results for each LP model on **FreeBase** datasets varying the RPS of the test facts, computing RPS with paths up to length 1 and up to length 2.

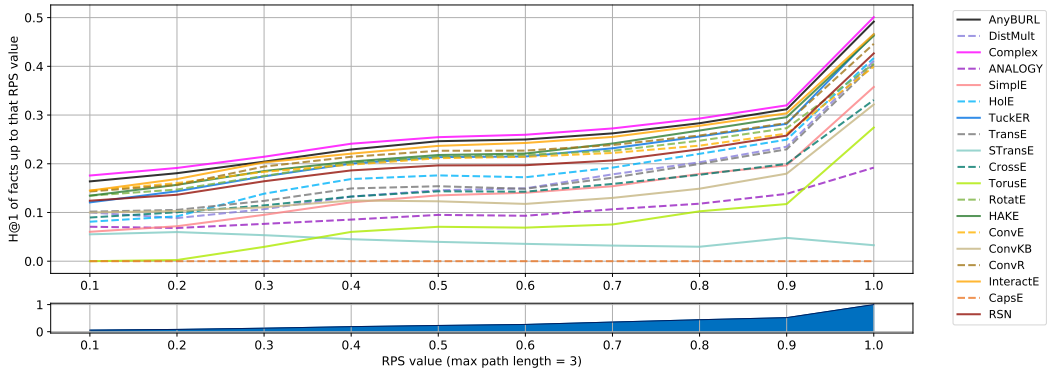


(a) WN18



(b) WN18RR

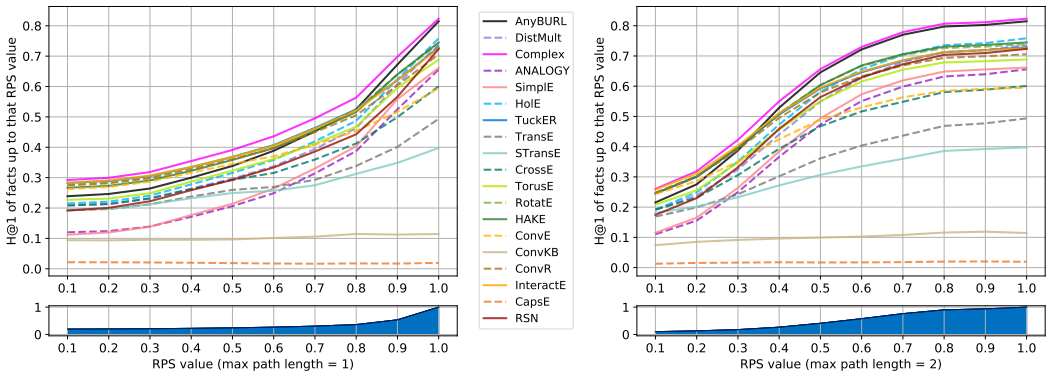
Fig. 16. Cumulative H@1 results for each LP model on **WordNet** datasets varying the RPS of the test facts, computing RPS with paths up to length 1 and up to length 2.



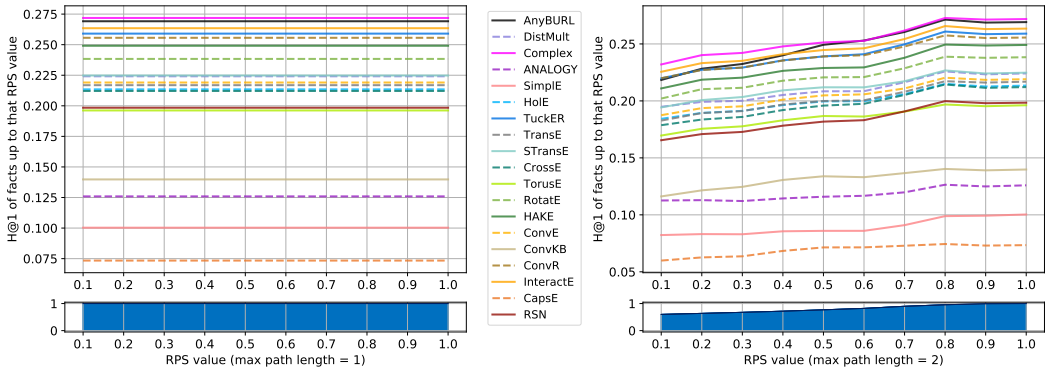
(a) YAGO3-10

Fig. 17. Cumulative H@1 results for each LP model on **Yago** datasets varying the RPS of the test facts, computing RPS with paths up to length 1 and up to length 2.

C RELATIONAL PATH SUPPORT WITH PATHS OF MAXIMUM LENGTHS 1 AND 2

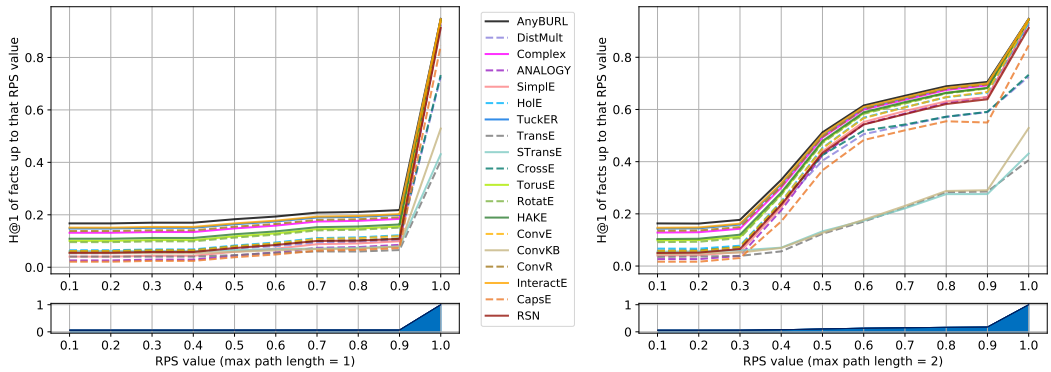


(a) FB15k

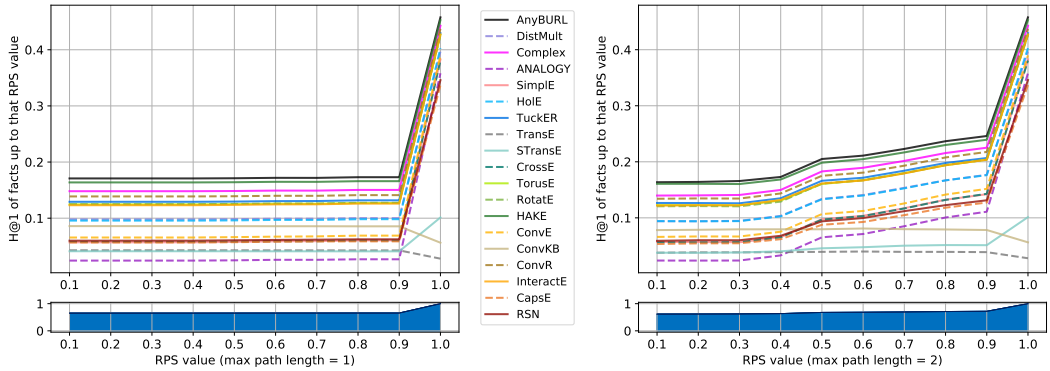


(b) FB15k-237

Fig. 18. Cumulative H@1 results for each LP model on **FreeBase** datasets varying the RPS of the test facts, computing RPS with paths up to length 1 and up to length 2.

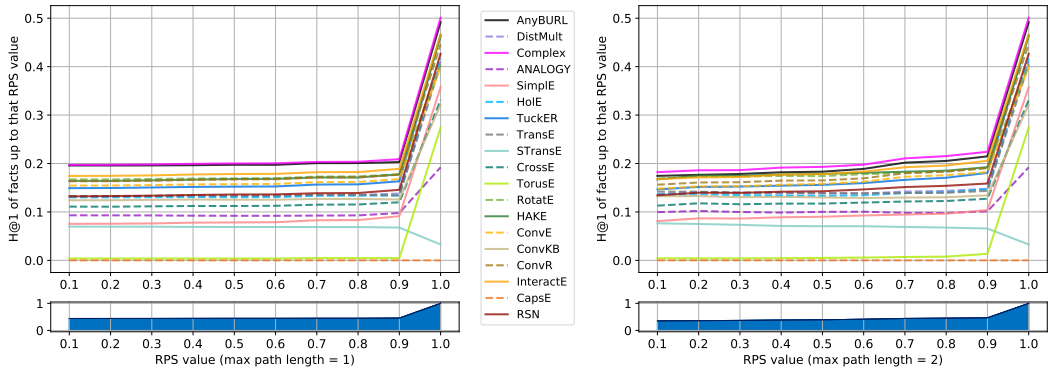


(a) WN18



(b) WN18RR

Fig. 19. Cumulative H@1 results for each LP model on **WordNet** datasets varying the RPS of the test facts, computing RPS with paths up to length 1 and up to length 2.



(a) YAGO3-10

Fig. 20. Cumulative H@1 results for each LP model on **Yago** datasets varying the RPS of the test facts, computing RPS with paths up to length 1 and up to length 2.

D RELATIONAL PATH SUPPORT AS A STANDALONE MODEL

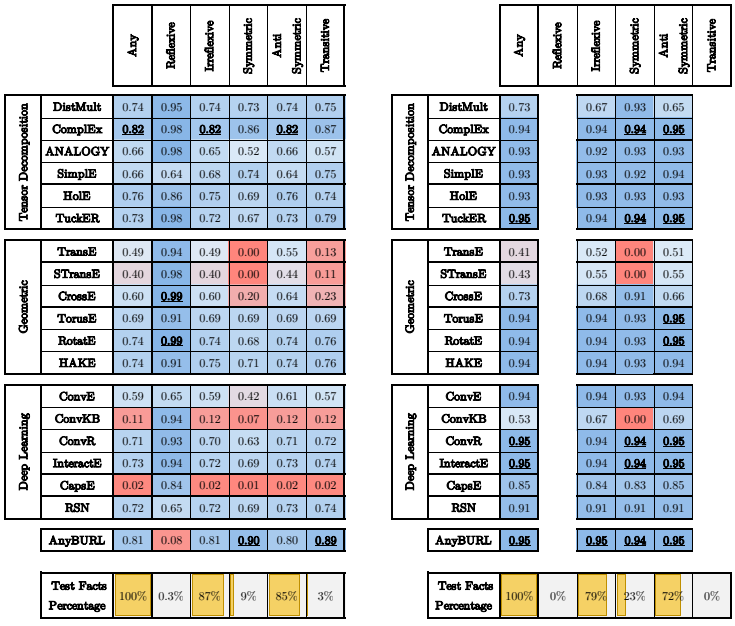
We report here the results for our experiment using RPS as the scoring function of a standalone LP model based on observable features.

Due to computational constraints we limit ourselves to paths up to 2 steps long; we work on random samples of 512 test facts on each dataset; and we do not run this experiment on YAGO3-10, on which the very high number of entities would make the ranking loop unfeasible.

	FB15k				WN18				FB15k-237				WN18RR			
	H@1	H@10	MR	MRR	H@1	H@10	MR	MRR	H@1	H@10	MR	MRR	H@1	H@10	MR	MRR
RPS Model	50.49	67.38	329	0.559	93.55	94.04	1123	0.937	10.45	25.00	2157	0.153	36.33	41.11	11493	0.380

Table 5. Performance of an LP model based on observable features, using as a scoring function the RPS measure with relational paths up to 3 steps long.

E RELATION PROPERTIES



(a) FB15k (b) WN18

Fig. 21. H@1 results for each LP model on FB15k and WN18 datasets datasets and corresponding percentages of test facts, for various relation properties. The best results for each column are in bold and underlined.

F REIFIED RELATION DEGREE

		Degree of original reified relation					
		1	2-4	5-8	9-16	17-32	> 32
Tensor Decomposition	DistMult	0.80	0.84	0.92	0.93	0.94	0.96
	ComplEx	**0.86**	**0.92**	**0.94**	**0.95**	**0.96**	**0.98**
	ANALOGY	0.76	0.83	0.90	0.92	0.93	0.96
	SimplE	0.75	0.84	0.91	0.91	0.92	0.96
	HolE	0.82	0.85	0.92	0.93	0.94	0.96
	TuckER	0.85	0.88	0.92	0.93	0.94	0.97
Geometric	TransE	0.82	0.82	0.89	0.86	0.85	0.94
	STransE	0.76	0.76	0.82	0.82	0.84	0.92
	CrossE	0.80	0.86	0.91	0.91	0.94	0.96
	TorusE	0.79	0.81	0.90	0.90	0.92	0.94
	RotatE	0.84	0.86	0.93	0.94	0.95	0.96
	HAKE	0.84	0.86	0.93	0.94	0.95	0.96
Deep Learning	ConvE	0.81	0.82	0.88	0.89	0.90	0.96
	ConvKB	0.36	0.37	0.38	0.40	0.40	0.64
	ConvR	0.84	0.88	0.92	0.93	0.94	0.96
	InteractE	0.84	0.88	0.92	0.93	0.95	0.98
	CapsE	0.22	0.22	0.16	0.13	0.12	0.32
	RSN	0.80	0.90	0.91	0.90	0.92	0.96
	AnyBURL	0.82	0.90	0.91	0.94	0.95	0.96
	Test Facts Percentage	39%	25%	14%	5%	3%	14%

(a) FB15k

		Degree of original reified relation					
		1	2-4	5-8	9-16	17-32	> 32
Tensor Decomposition	DistMult	0.47	0.52	0.52	0.55	0.60	0.65
	ComplEx	**0.54**	**0.60**	**0.60**	**0.64**	0.69	0.69
	ANALOGY	0.34	0.39	0.38	0.40	0.38	0.50
	SimplE	0.32	0.40	0.38	0.40	0.46	0.54
	HolE	0.46	0.50	0.50	0.50	0.55	0.60
	TuckER	0.52	0.57	0.56	0.58	0.64	0.68
Geometric	TransE	0.47	0.54	0.54	0.55	0.61	0.64
	STransE	0.48	0.52	0.51	0.54	0.63	0.64
	CrossE	0.45	0.52	0.49	0.50	0.51	0.62
	TorusE	0.44	0.46	0.48	0.48	0.50	0.55
	RotatE	0.51	0.57	0.58	0.60	0.65	0.68
	HAKE	0.52	0.57	0.59	0.60	0.66	0.67
Deep Learning	ConvE	0.46	0.50	0.50	0.55	0.55	0.63
	ConvKB	0.40	0.43	0.44	0.44	0.50	0.55
	ConvR	0.50	0.56	0.54	0.59	0.62	0.68
	InteractE	0.51	0.58	0.57	0.55	0.65	**0.70**
	CapsE	0.35	0.34	0.36	0.38	0.40	0.48
	RSN	0.42	0.47	0.48	0.48	0.51	0.60
	AnyBURL	0.49	0.55	**0.60**	0.58	**0.70**	0.69
	Test Facts Percentage	74%	14%	4%	1%	1%	6%

(b) FB15k-237

Fig. 22. H@10 results for each LP model on the **Freebase** datasets, and corresponding distribution of test facts, varying the degree of the original reified relation in FreeBase. The best results for each column are marked in bold and underlined.

G HYPERPARAMETERS

We report here the hyperparameter setting used for each model in our experiments. We highlight in yellow the settings we have found manually, and report in the *Space* column the size of the corresponding space of combinations.

	FB15k	WN18	FB15k-237	WN18RR	YAGO3-10	Space
DistMult	BC:50; Ep:4000; d:200; γ:1; LR:5e-4; N:20; Loss: self_adv;	BC:50; Ep:4000; d:200; γ:1; LR: 5e-4; N:20; Loss:NLL	BC:50; Ep:4000; d:300; LR:5e-5; Reg:L3(λ:1e-4); N:50; Loss: multiclassNLL;	BC:100; Ep:4000; d:350; LR:1e-4; Reg:L2(λ:1e-4); N:30; Loss:multiclassNLL;	BC:100; Ep:4000; d:350; LR:5e-05; Reg:L3(λ:1e-4); N:30; Loss:multiclassNLL;	
ComplEx	B:100; Ep:200; d:2000; LR:1e-2; Reg:N3; Opt:Adagrad; Loss:multiclassNLL(full);	B:1000; Ep:20; d:2000; LR:0.1; Reg:N3; Opt:Adagrad; Loss:multiclassNLL(full);	B:100; Ep:100; d:2000; LR:0.1; Reg:N3; Opt:Adagrad; Loss:multiclassNLL(full);	B:100; Ep:100; d:2000; LR:0.1; Reg:N3; Opt:Adagrad; Loss:multiclassNLL(full);	B:1000; Ep: 100; d:2000; LR:0.1; Reg:N3; Opt:Adagrad; Loss:multiclassNLL(full);	
ANALOGY	B:1; Ep:500; d:200; LR:0.1; Decay:1e-3; Opt:Adagrad; N:6; Loss:NLL(variant);	B:1; Ep:500; d:200; LR:0.1; Decay:1e-2; Opt:Adagrad; N:3; Loss:NLL(variant);	B:1; Ep:500; d:150; LR:1e-2; Decay:1e-2; Opt:Adagrad; N:6; Loss:NLL(variant);	B:1; Ep:500; d:200; LR:0.1; Decay:1e-3; Opt:Adagrad; N:6; Loss:NLL(variant);	B:1; Ep:500; d:100; LR:0.1; Decay:1e-3; Opt:Adagrad; N:6; Loss:NLL(variant);	36
SimplE	B:4832; Ep:1000; d:200; Reg:L2(λ:0.1); LR:5e-2; N:10; Loss:NLL;	B:1415; Ep:1000; d:200; Reg:L2(λ:3e-2); LR:0.2; N:1; Loss:NLL;	B:4832; Ep:500; d:200; Reg:L2(λ:0.1); LR:0.1; N:3; Loss:NLL;	B:1415; Ep:1000; d:150; Reg:L2(λ:3e-2); LR:0.1; N:10; Loss:NLL;	B:2048; Ep:1000; d:200; Reg:L2(λ:3e-2); LR:0.2; N:10; Loss:NLL;	1176
HolE	BC:50; Ep:4000; d:200; γ:1; LR:5e-4; N:20; Loss:self_adv;	BC:50; Ep:4000; d:200; γ:1; LR:5e-4; N:20; Loss:self_adv;	BC:64; Ep:4000; d:350; LR:1e-4; Reg:L2(λ:1e-4); N:30; Loss:multiclassNLL;	BC:50; Ep:4000; d:200; γ:1; LR:5e-4; N:20; Loss:self_adv;	BC:100; Ep:4000; d:350; γ:0.5; LR:5e-4; N:30; Loss:self_adv;	
TuckER	B:128; Ep:500; d:200; DecayRate:0.99; Drop:{in:0.2; h_1:0.2; h_2:0.3}; LR:3e-3; ε:0; Loss:BCE;	B:128; Ep:500; d:200; DecayRate:0.995; Drop:{in:0.3; h_1:0.4; h_2:0.5}; LR:5e-4; ε:0.1; Loss:BCE;	B:128; Ep:500; d_e:200; d_r:30; DecayRate:1; Drop:{in:0.2; h_1:0.1; h_2:0.2}; LR:5e-3; ε:0.1; Loss:BCE;	B:128; Ep:500; d_e:200; d_r:30; DecayRate:1; Drop:{in:0.2; h_1:0.2; h_2:0.3}; LR:1e-2; ε:0.1; Loss:BCE;	B:128; Ep:500; d_e:200; d_r:30; DecayRate:1; Drop:{in:0.2; h_1:0.2; h_2:0.2}; LR:1e-2; ε:0; Loss:BCE;	3750
TransE	BC:100; Ep:4000; d:150; Norm:L1; LR: 5e-05; Reg:L3(λ:1e-4); N:10; Loss:multiclassNLL;	BC:100; Ep:4000; d:150; Norm: L1; LR: 5e-05; Reg:L3(λ:1e-4); N:10; Loss:multiclassNLL(sampled;	BC:64; Ep:4000; d:400; Norm: L1; LR:1e-4; Reg:L2(λ:1e-4); N:30; Loss:multiclassNLL;	BC:150; Ep:4000; d:350; Norm:L1; LR:1e-4; Reg:L2(λ:1e-4); N:30; Loss:multiclassNLL;	BC:100; Ep:4000; d:350; Norm:L1; LR:1e-4; Reg:L2(λ:1e-4); N:30; Loss:multiclassNLL;	
STransE	B:1; Ep:2000; d:100; LR:1e-4; Opt:SGD; Norm:L1; γ:1; N:1; Loss:Margin-based;	B:1; Ep:2000; d:50; LR:5e-4; Opt:SGD; Norm:L1; γ:5; N:1; Loss:Margin-based;	B:1; Ep:2000; d:100;LR:1e-3; Opt:SGD; Norm:L1; γ:5; N:1; Loss:Margin-based;	B:1; Ep:2000; d:50; LR:5e-4; Opt:SGD; Norm:L1; γ:5; N:1; Loss:Margin-based;	B:1; Ep:500; d:350; LR:1e-2; Opt:SGD; Norm:L2; γ:5; N:1; Loss:Margin-based;	60
CrossE	B:4000; Ep:500; d:300; LR:1e-2; Reg:L1(λ:1e-6); Loss:BCE;	B:2048; Ep:500; d:100; LR:1e-2; Reg:L1(λ:1e-4); Loss:BCE;	B:4000; Ep:500; d:100; LR:1e-2; Reg:L1(λ:1e-4); Loss:BCE;	B:4000; Ep:120; d:100; LR:1e-2; Reg:L1(λ:1e-4); Loss:BCE;	B:1024*; Ep:150; d:300; LR:1e-2; Reg:L1(λ:1e-6); Loss:BCE;	12
TorusE	BC:100; Ep:1000; d:10000; LR:5e-4; Norm:EL2; Opt:SGD; γ:500; N:1; Loss:Margin-based;	BC:100; Ep:1000; d:10000; LR:5e-4; Norm:L1; Opt:SGD; γ:2000; N:1; Loss:Margin-based;	BC:100; Ep:1000; d:10000; LR:1e-3; Norm:L1; Opt:SGD; γ:2000; N:1; Loss:Margin-based;	BC:100; Ep:1000; d:10000; LR:2e-4; Norm:L1; Opt:SGD; γ:3000; N:1; Loss:Margin-based;	BC:100; Ep:250; d:10000; LR:2e-4; Norm:EL2; Opt:SGD; γ:100; N:1; Loss:Margin-based;	75
RotatE	B:1024; Steps:150000; d:1000; LR:1e-4; N:256; γ:24; Loss:self_adv; α:1.0;	B:512; Steps:80000; d:500; LR:1e-4; N:1024; γ:12; Loss:self_adv; α:0.5;	B:1024; Steps:100000; d:1000; LR:5e-5; N:256; γ:9; Loss:self_adv; α:1.0;	B:512; Steps:80000; d:500; LR:5e-5; N:1024; γ:6; Loss:self_adv; α:0.5;	B:1024; Steps:100000; d:500; LR:2e-4; N:400; γ:24; Loss:self_adv; α:1.0;	
HAKE	B:1024; Steps:80000; d:1000; LR:5e-5; N:256; γ:24.0; Loss:self_adv; α:1.0; λ_m:3.5; λ_p:1.0;	B:512; Steps:80000; d:500; LR:1e-4; N:1024; γ:12.0; Loss:self_adv; α:0.5; λ_m:0.5; λ_p:0.5;	B:1024; Steps:100000; d:1000; LR:5e-5; N:256; γ:9.0; Loss:self_adv; α:1.0; λ_m:3.5; λ_p:1.0;	B:512; Steps:80000; d:500; LR:5e-5; N:1024; γ:6.0; Loss:self_adv; α:0.5; λ_m:0.5; λ_p:0.5;	B:1024; Steps:180000; d:500; LR:2e-4; N:256; γ:24.0; Loss:self_adv; α:1.0; λ_m:1.0; λ_p:0.5;	
ConvE	B:128; Ep:1000; d:200; LR:3e-3; Decay Rate:0.995; ω:3x3; T:256; ε:0.1; Drop:{in:0.2;h:0.3;feat:0.2}; Loss:BCE;	B:128; Ep:1000; d:200; LR:3e-3; Decay Rate:0.995; ε:0.1; ω:3x3; T:256; Drop:{in:0.2;h:0.3;feat:0.2}; Loss:BCE;	B:128; Ep:1000; d:200; LR:3e-3; Decay Rate:0.995; ε:0.1; ω:3x3; T:256; Drop:{in:0.2;h:0.3;feat:0.2}; Loss:BCE;	B:128; Ep:1000; d:200; LR:3e-3; Decay Rate:0.995; ε:0.1; ω:3x3; T:256; Drop:{in:0.2;h:0.3;feat:0.2}; Loss:BCE;	B:128; Ep:1000; d:200; LR:3e-3; Decay Rate:0.995; ε:0.1; ω:3x3; T:256; Drop:{in:0.2;h:0.3;feat:0.2}; Loss:BCE;	
ConvKB	B:128; Ep:200; d:100; LR:5e-4; T:200; Reg:L2(λ:1e-3); Loss:NLL; ConstantInit:Y;	B:256; Ep:200; d:50; LR:5e-5; T:500; Reg:L2(λ:1e-3); Loss:NLL; ConstantInit:N;	B:256; Ep:200; d:100; LR:5e-6; T:50; Reg:L2(λ:1e-3); Loss:NLL; ConstantInit:Y;	B:256; Ep:200; d:50; LR:1e-4; T:500; Reg:L2(λ:1e-3); Loss:NLL; ConstantInit:N;	B:256; Ep:200; d:350; LR:1e-05; T:500; Reg:L2(λ:1e-3); Loss:NLL; ConstantInit:N;	50
ConvR	B:128; Ep:1000; d_e:200; ω_r:3x3; T:100; ε:0.1; Drop:{in:0.1;h:0.2;feat:0.4}; LR:1e-3; Loss:BCE;	B:128; Ep:1000; d_e:200; ω_r:3x3; T:100; ε:0.1; Drop:{in:0.4;h:0.3;feat:0.3}; LR:1e-3; Loss:BCE;	B:128; Ep:1000; d_e:100; ω_r:5x5; T:100; ε:0.1; Drop:{in:0.3;h:0.2;feat:0.3}; LR:1e-3; Loss:BCE;	B:128; Ep:1000; d_e:200; ω_r:3x3; T:200; ε:0.1; Drop:{in:0.2;h:0.2;feat:0.5}; LR:1e-3; Loss:BCE;	B:128; Ep:1000; d_e:200; ω_r:3x3; T:100; ε:0.1; Drop:{in:0.1;h:0.2;feat:0.2}; LR:1e-3; Loss:BCE;	3000
InteractE	B:128; Ep:500; d:200; p:2; ω:7x7; T:64; ε:0.1; Drop:{in:0.2;h:0.3;feat:0.2}; LR:1e-4; Loss:BCE;	B:128; Ep:300; d:200; p:1; ω:9x9; T:96; ε:0.1; Drop:{in:0.2;h:0.3;feat:0.2}; LR:1e-4; Loss:BCE;	B:128; Ep:300; d:200; p:1; ω:9x9; T:96; ε:0.1; Drop:{in:0.2;h:0.5;feat:0.5}; LR:1e-4; Loss:NLL; N:1000;	B:256; Ep:300; d:200; p:4; ω:11x11; T:96; ε:0.1; Drop:{in:0.2;h:0.3;feat:0.2}; LR:1e-3; Loss:BCE;	B:128; Ep:500; d:200; p:2; ω:7x7; T:64; ε:0.1; Drop:{in:0.2;h:0.3;feat:0.2}; LR:1e-4; Loss:BCE;	
CapsE	B:128; Ep:60; d:100; T:500; LR:5e-4; Loss:NLL; ConstantInit:Y;	B:128; Ep:60; d:50; T:50; LR:5e-5; Loss:NLL; ConstantInit:N;	B:128; Ep:30; d:100; T:50; LR:5e-4; Loss:NLL; ConstantInit:Y;	B:128; Ep:60; d:100; T:400; LR:5e-4; Loss:NLL; ConstantInit:N;	B:128; Ep:30; d:350; T:200; LR:5e-4; Loss:NLL; ConstantInit:Y;	12
RSN	B: 2048; Ep: 200; d:256; LR:1e-4; Drop:{h:0.5};	B: 2048; Ep: 200; d:256; LR:1e-5; Drop:{h:0.5};	B: 2048; Ep: 200; d:256; LR:1e-4; Drop:{h:0.5};	B: 2048; Ep: 200; d:256; LR:5e-5; Drop:{h:0.5};	B: 2048; Ep: 200; d:256; LR:1e-4; Drop:{h:0.5};	2
AnyBURL	Training time:100s; Max cyclic paths length:3;	Training time:100s; Max cyclic paths length:5;	Training time:1000s: Max cyclic paths length:3;	Training time:1000s; Max cyclic paths length:5;	Training time:1000s; Max cyclic paths length:3;	

Table 6. Hyperparameters used to train all the models in our work. B: batch size; alternatively BC: batch count. Ep: training epochs; alternatively $Steps$: training steps. D: embedding dimension; alternatively, D_e and D_r: entity and relation embedding dimension. LR: learning rate. γ: regularization margin. Reg: regularization method; λ: lambda for $Reg \in \{L1, L2, L3\}$. ϵ: label smoothing. Opt: optimizer (default: $ADAM$). K: convolutional filters. $Kernel$: convolutional kernel. $Drop$: dropout rate (in: in input; h_i: in i-th hidden layer; $feat$: in features). N: negative samples per training fact (default: 1). $AdvTemp$: temperature in adversarial negative sampling. $ConstantInit$: initialize filters as [0.1, 0.1, -0.1] if Y, otherwise from a truncated normal distribution.

H CONFIDENCE INTERVALS

We report here the confidence intervals we have obtained on the main LP evaluation metrics. We have focused on the best performing models of each family, that is, ComplEx [33] for the tensor decomposition family, HAKE [80] for the geometric family, and InteractE [68] for the deep learning family; we have included the baseline AnyBURL[42] as well. We have run 5 training processes for each model on each of the datasets FB15k, FB15k-237, WN18, and WN18RR, and we have extracted the corresponding evaluation metrics. We report in Table 7, for each metric, the mean value across the various re-training processes, as well as the lower and upper bounds computed with the Bootstrap method [13] (note that the mean values may differ from the values reported Table 3, that refer to one training only). We have not included YAGO3-10 in these experiments because, due to its larger size, it requires significantly longer training and evaluation times than the other datasets.

Across all datasets and models, all metrics tend to be very stable. MRR is by far the most consistent, remaining almost identical in all re-trainings, with lower and upper bounds differing from the mean value for less than its 0.3% in all cases. MR, on the contrary, is the most fluctuating metric, with the difference between the upper/lower bounds and the mean value generally between 1% and 3% of the mean value.

	FB15k				**WN18**				**FB15k-237**				**WN18RR**			
	H@1	**H@10**	**MR**	**MRR**	**H@1**	**H@10**	**MR**	**MRR**	**H@1**	**H@10**	**MR**	**MRR**	**H@1**	**H@10**	**MR**	**MRR**
ComplEx	82.38 L:82.34 U:82.42	91.00 L:90.98 U:91.02	34 L:33 U:34	0.855 L:0.855 U:0.855	94.42 L:94.40 U:94.44	96.12 L:96.10 U:96.17	187 L:172 U:201	0.951 L:0.950 U:0.951	27.03 L:26.95 U:27.09	55.75 L:55.71 U:55.79	144 L:144 U:145	0.366 L:0.365 U:0.366	44.18 L:44.08 U:44.29	58.02 L:57.96 U:58.07	2863 L:2852 U:2880	0.488 L:0.487 U:0.489
HAKE	74.68 L:74.59 U:74.78	88.38 L:88.34 U:88.42	44 L:44 U:44	0.797 L:0.796 U:0.797	94.39 L:94.36 U:94.41	96.16 L:96.14 U:96.19	230 L:230 U:231	0.950 L:0.950 U:0.950	24.79 L:24.69 U:24.91	54.19 L:54.12 U:54.26	184 L:183 U:185	0.345 L:0.344 U:0.346	45.16 L:45.08 U:45.26	58.23 L:58.16 U:58.33	3316 L:3300 U:3329	0.497 L:0.496 U:0.497
InteractE	72.15 L:71.90 U:72.36	88.56 L:88.36 U:88.76	64 L:62 U:65	0.783 L:0.781 U:0.785	26.41 L:26.36 U:26.45	53.96 L:53.82 U:54.10	183 L:182 U:184	0.356 L:0.355 U:0.356	26.41 L:26.36 U:26.45	53.96 L:53.82 U:54.10	183 L:182 U:184	0.356 L:0.355 U:0.356	42.80 L:42.71 U:42.90	51.70 L:51.47 U:51.93	5133 L:4992 U:5215	0.458 L:0.458 U:0.459
AnyBURL	81.22 L:81.01 U:81.43	87.80 L:87.57 U:88.03	202 L:177 U:233	0.835 L:0.833 U:0.837	94.73 L:94.70 U:94.75	96.18 L:96.16 U:96.22	361 L:354 U:365	0.953 L:0.953 U:0.953	26.88 L:26.83 U:26.93	51.75 L:51.61 U:51.91	275 L:272 U:279	0.353 L:0.352 U:0.354	45.75 L:45.68 U:45.80	57.56 L:57.45 U:57.72	4060 L:4035 U:4077	0.497 L:0.496 U:0.498

Table 7. Confidence intervals on MR, MRR, H@1 and H@10 metrics on FB15k, FB15k-237, WN18 and WN18RR dataset on ComplEx, HAKE, InteractE and AnyBURL. In each cell, the first reported value is the mean obtained across 5 separate trainings; the values denoted by L and U are respectively the lower and upper bounds obtained applying the Boostrap method.