Knowledge Graph Embedding for Link Prediction: A Comparative Analysis

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

arXiv:2002.00819v2 [cs.LG] 6 Mar 2020

Knowledge Graphs (KGs) have found many applications in industry and 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 state-of-the-art KGs suffer from incompleteness. Link Prediction (LP), the task of predicting missing facts among entities already a KG, is a promising and widely studied task aimed at addressing KG incompleteness. Among the recent LP techniques, those based on *KG embeddings* have achieved very promising performances in some benchmarks. Despite the fast growing literature in the subject, insufficient attention has been paid to the effect of the various 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 over-represented; this allows LP methods to exhibit good performance 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 effectiveness and efficiency of 16 state-of-the-art methods, consider a rule-based baseline, and report detailed analysis over the most popular benchmarks in the literature.

Additional Key Words and Phrases: A, B, C, D

ACM Reference format:
Andrea Rossi, Donatella Firmani, Antonio Matinata, Paolo Merialdo, and Denilson Barbosa. 2016. Knowledge Graph Embedding for Link Prediction: A Comparative Analysis. 1, 1, Article 1 (January 2016), 43 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

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, complex data in a machine-readable way, KGs are nowadays widely employed in various domains, ranging from question answering to information retrieval and content-based recommendation systems, and they are vital to any semantic web project [23]. Examples of notable KGs are FreeBase [7], WikiData [62], DBPedia [4], Yago [54] and – in industry – Google KG [52], Microsoft Satori [48] and Facebook Graph Search [53]. These massive KGs can contain millions of entities and billions of facts.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2016 ACM. Manuscript submitted to ACM

Despite such efforts, it is well known that even state-of-the-art KGs suffer from *incompleteness*. For instance, it has been observed that over 70% of person entities have no known place of birth, and over 99% have no known ethnicity [12, 69] in FreeBase, one of the largest and most widely used KGs for research purposes. This has led researchers to propose various techniques for correcting errors as well as adding missing facts to KGs [47], commonly known as the task of *Knowledge Graph Completion* or *Knowledge Graph Augmentation*. Growing an existing KG can be done by extracting new facts from external sources, such as Web corpora, or by inferring missing facts 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, which has more recently benefited from the explosion of machine learning and deep learning techniques. The vast majority of LP models nowadays use original KG elements to learn low-dimensional representations dubbed *Knowledge Graph Embeddings*, and then employ them to infer new facts. Inspired by a few seminal works such as RESCAL [44] and TransE [8], in the short span of just a few years researchers have developed dozens of novel 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. As a result, LP methods can exhibit good performance on these benchmarks by attending only to such entities while ignoring the others. Moreover, the limitations of the current best-practice can make it difficult for one to understand how the papers in this literature fit together and to picture what research directions are worth pursuing. In addition to that, the strengths, weaknesses and limitations of the current techniques are still unknown, that is, 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 the issues mentioned above, we carry out an extensive comparative analysis of a representative set of LP models based on KG 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 16 models belonging to diverse machine learning and deep learning architectures; we also adopt as a baseline an additional state-of-the-art LP model based on rule mining. We provide a detailed description of the approaches considered for experimental comparison and a summary of related literature, together with an educational taxonomy for Knowledge Graph Embedding techniques.
- We take into account 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.

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 follow. Section 2 provides background on KG embedding 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.

[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.

2 THE LINK PREDICTION PROBLEM

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

We define a KG as a labeled, directed multi-graph $KG = (\mathcal{E}, \mathcal{R}, \mathcal{G})$:

- $\mathcal{E}$: 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* of the fact.

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 $\langle h, r, ? \rangle$ (tail prediction) or $\langle ?, r, t \rangle$ (head prediction). For the sake of simplicity, when talking about head and tail prediction globally, we call *source* entity the known entity in the prediction, and *target* entity the one to predict.

In time, numerous approaches have been proposed to tackle the LP task. Some methods are based on observable features and employ techniques such as Rule Mining [17][16][37][24] or the Path Ranking Algorithm [31][32] to identify missing triples in the graph. Recently, with the rise of novel Machine Learning techniques, researchers have been experimenting on capturing latent features of the graph with vectorized representations, or embeddings, of its components. In general, *embeddings* are vectors of numerical values that can be used to represent any kind of elements (e.g., depending on the domain: words, people, products...). Embeddings 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 in a vocabulary, and they are usually learned using textual corpora as input data. When it comes to KGs, embeddings are typically used to represent entities and relationships using the graph structure; the resulting vectors, dubbed **KG Embeddings**, embody the semantics of the original graph, and can be used to identify new links inside it, thus tackling the LP task.

In the following we use $italic$ letters to identify KG elements (entities or relations), and $\boldsymbol{bold}$ letters to identify the corresponding embeddings. Given for instance a generic entity, we may use e when referring to its element in the graph, and $\boldsymbol{e}$ when referring to its embedding.

Datasets employed in LP research are typically obtained subsampling 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 three disjoint subsets: a training set $\mathcal{G}_{train}$, a validation set $\mathcal{G}_{valid}$ and a test set $\mathcal{G}_{test}$.

Most of LP models based on embeddings define a scoring function ϕ to estimate the plausibility of any fact $\langle h,\ r,\ t \rangle$ using their embeddings:

$$\phi(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})$$

In this paper, unless differently specified, we are going to assume that the higher the score of ϕ, the more plausible the fact.

During training, embeddings are usually initialized randomly and subsequently improved with optimization algorithms such as back-propagation with gradient descent. The positive samples in $\mathcal{G}_{train}$ are often randomly corrupted in order to generate negative samples. The optimization process aims at maximizing the plausibility of positive facts as well as minimizing the plausibility of negative facts; this often amounts to employing a **triplet loss** function. Over time, more effective ways to generate negative triples have been proposed, such as sampling from a Bernouilli distribution [66]

or generating them with adversarial algorithms [55]. In addition to the embeddings of KG elements, models may also use the same optimization algorithms to learn additional parameters (e.g. the weights of neural layers). Such parameters, if present, are employed in the scoring function ϕ to process the actual embeddings of entities and relations. Since they are not specific to any KG element, they are often dubbed *shared parameters*.

In 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:

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

Head prediction is performed analogously.

Evaluation is carried out by performing both head and tail prediction on all test triples in $\mathcal{G}_{test}$, and computing for each prediction how the target entity ranks against all the other ones. Ideally, the target entity should yield the highest plausibility.

Ranks can be computed in two largely different settings, called raw and filtered scenarios. As a matter of fact, a prediction may have multiple valid answers: 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 the predicted fact is contained in $\mathcal{G}$ (that is, either in $\mathcal{G}_{train}$, or in $\mathcal{G}_{valid}$ or in $\mathcal{G}_{test}$), the answer is valid. Depending on whether valid answers should be considered acceptable or not, two separate settings have been devised:

- *Raw Scenario*: in this scenario, valid entities outscoring the target one are considered as mistakes. Therefore 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:

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

  The raw rank in head prediction can be computed analogously.
- *Filtered Scenario*: in this scenario, valid entities outscoring the target one are not considered mistakes. Therefore they are skipped when computing the rank. 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:

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

  The filtered rank in head prediction can be computed analogously.

In order to compute the rank it is also necessary to define the policy to apply when the target entity obtains the same score as other ones. This event is called a *tie* and it can be handled with different policies:

- *min*: the target is given the lowest rank among the entities in tie. This is the most permissive policy, and it may result in artificially boosting performances: as an extreme example, a model systematically setting the same score to all entities would obtain perfect results under this policy.
- *average*: the target is given the average rank among the entities in tie.
- *random*: the target is given a random rank among the entities in tie. On large test sets, this policy should globally amount to the average policy.
- *ordinal*: the entities in tie are given ranks based on the order in which they have been passed to the model. This usually depends on the internal identifiers of entities, which are independent from their scores: therefore this policy should globally correspond to the random policy.
- *max*: the target is given the highest (worst) rank among the entities in tie. This is the most strict policy.

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

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

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

It is always between 1 and $|\mathcal{E}|$, and the lower it is, the better the model results. It is very sensitive to outliers, therefore researchers lately have started avoiding it, resorting to Mean Reciprocal Rank instead.

Mean Reciprocal Rank (MRR). It is the average of the inverse of the obtained ranks:

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

It is always between 0 and 1, and the higher it is, the better the model results.

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

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

Common values for K are 1, 3, 5, 10. The higher the H@K, the better the model results. In particular, when $K = 1$, it measures the ratio of the test facts in which the target was predicted correctly on the first try. H@1 and MRR are often closely related, because these predictions also correspond to the most relevant addends to the MRR formula.

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

3 OVERVIEW OF LINK PREDICTION TECHNIQUES

In this section we survey and discuss the main LP approaches for KGs based on latent features. As already described in Section 2, LP models can exploit a large variety of approaches and architectures, depending on how they model the optimization problem and on the techniques they implement to tackle it.

In order to overview their highly diverse characteristics we propose a novel taxonomy illustrated in Figure 1. We define three main families of models, and further divide each of them into smaller groups, identified by unique colours. For each group, we include the most valid representative models, prioritizing the ones reaching state-of-the-art performance and, whenever possible, those with publicly available implementations. The result is a set of 16 models based on extremely diverse architectures; these are the models we subsequently employ in the experimental sections of our comparative analysis. For each model we also report the year of publication as well as 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, such as their loss function 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. We refer the reader to works discussing how to leverage additional sources of information, such as textual captions [58],[65],[3], images [70] or pre-computed rules [20]; see [18] for a survey exclusive to these models.

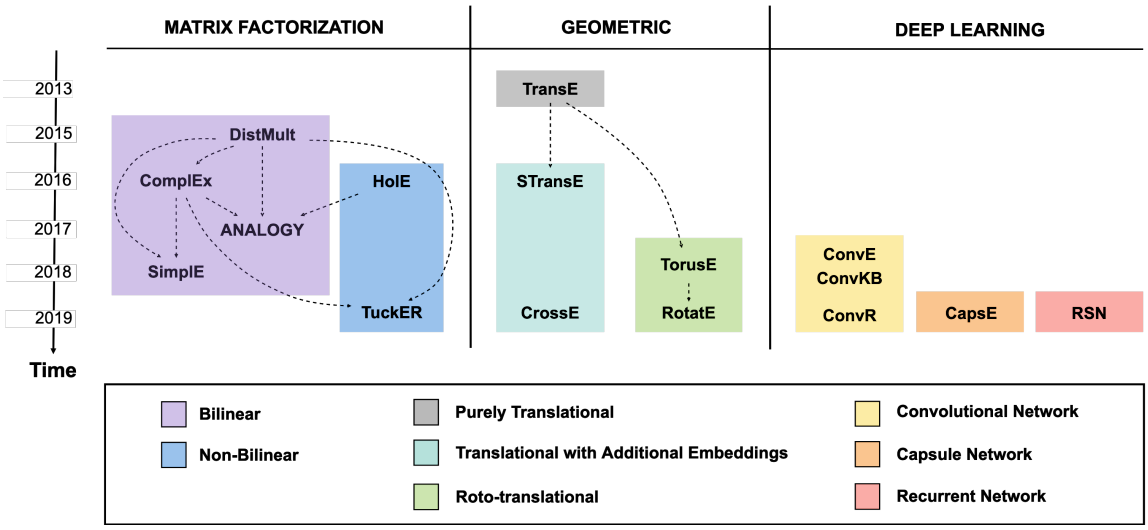


Fig. 1. Taxonomy for the LP models included in our analysis. Dotted arrows indicate that the target method builds on the source method by either generalizing or specializing the definition of its scoring function. The included models are: DistMult [71]; ComplEx [61]; ANALOGY [35]; SimplE [27]; HolE [46]; TuckER [6]; TransE [8]; STransE [41]; CrossE [72]; TorusE [13]; RotatE [55]; ConvE [11]; ConvKB [42]; ConvR [25]; CapsE [43]; RSN [19].

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

3.1 Tensor Decomposition Models

Models in this family interpret LP as a task of tensor decomposition [28]. These models implicitly consider the KG as a 3D adjacency matrix (that is, a 3-way tensor), that is only only partially observable due to the KG incompleteness. The tensor is decomposed into a combination (e.g. a multi-linear product) of low-dimensional vectors: such vectors are used as embeddings for entities and relations. The core idea is that, provided that the model does not overfit on the training set, the learned embeddings should be able to generalize, and associate high values to unseen true facts in the graph adjacency matrix. In practice, the score of each fact is computed operating that combination on the specific embeddings involved in that fact; the embeddings are learned as usual by optimizing the scoring function for all training facts. 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 embedding $\boldsymbol{h} \in \mathbb{R}^d$ and the tail embedding $\boldsymbol{t} \in \mathbb{R}^d$, these models represent the relation embedding as a bidimensional matrix $\boldsymbol{r} \in \mathbb{R}^{d \times d}$. The scoring function is then computed as a bilinear product:

$$\phi(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \boldsymbol{h} \times \boldsymbol{r} \times \boldsymbol{t}$$

where symbol $\times$ denotes matrix product. These models usually differ from one another by introducing specific additional constraints on the embeddings they learn. For this group, in our comparative analysis, we include the following representative models:

DistMult [71] forces all relation embeddings to be diagonal matrices, which consistently reduces the space of parameters to be learned, resulting in a much easier model to train. On the other hand, this makes the scoring function commutative, with $\phi(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \phi(\boldsymbol{t}, \boldsymbol{r}, \boldsymbol{s})$, which amounts to treating all relations as symmetric. Despite this flaw, it has been demonstrated that, when carefully tuned, DistMult can still reach state-of-the-art performance [26].

ComplEx [61], similarly to DistMult, forces each relation embedding to be a diagonal matrix, but extends such formulation in the complex space: $\boldsymbol{h} \in \mathbb{C}^d$, $\boldsymbol{t} \in \mathbb{C}^d$, $\boldsymbol{r} \in \mathbb{C}^{d \times d}$. In the complex space, the bilinear product becomes a Hermitian product, where in lieu of the traditional $\boldsymbol{t}$, its conjugate-transpose $\bar{\boldsymbol{t}}$ is employed. This disables the commutativeness above mentioned for the scoring function, allowing ComplEx to successfully model asymmetric relations as well.

Analogy [35] aims at modeling analogical reasoning, which is key for any kind of knowledge induction. It employs the general bilinear scoring function but adds two main constraints inspired by analogical structures: $\boldsymbol{r}$ must be a normal matrix: $\boldsymbol{r}\boldsymbol{r}^T = \boldsymbol{r}^T\boldsymbol{r}$; for each pair of relations r_1, r_2, their composition must be commutative: $\boldsymbol{r_1} \circ \boldsymbol{r_2} = \boldsymbol{r_2} \circ \boldsymbol{r_1}$. The authors demonstrate that normal matrices can be successfully employed for modelling asymmetric relations.

SimplE [27] forces relation embeddings to be diagonal matrices, similarly to DistMult, but extends it by (i) associating with each entity e two separate embeddings, $\boldsymbol{e_h}$ and $\boldsymbol{e_t}$, depending on whether e is used as head or tail; (ii) associating with each relation r two separate diagonal matrices, $\boldsymbol{r}$ and $\boldsymbol{r_{-1}}$, expressing the relation in its regular and inverse direction. The score of a fact is computed averaging the bilinear scores of the regular fact and its inverse version. It has been demonstrated that SimplE is fully expressive, and therefore, unlike DistMult, it can model also asymmetric relations.

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

HolE [46], instead of using bilinear products, computes circular correlation (denoted by $\star$ in Table 1) between the embeddings of head and tail entities; then, it performs matrix multiplication with the relation embedding. Note that in this model the relation embeddings have the same shape as the entity embedding. The authors point out that circular correlation can be seen as a compression of the full matrix product: this makes HolE less expensive than an unconstrained bilinear model in terms of both time and space complexity.

TuckER [6] relies on the Tucker decomposition [21], which factorizes a tensor into a set of vectors and a smaller shared core $\boldsymbol{W}$. The TuckER model learns $\boldsymbol{W}$ jointly with the KG embeddings. As a matter of fact, learning globally shared parameters is rather uncommon in Matrix Factorization Models; the authors explain that $\boldsymbol{W}$ can be seen as a shared pool of prototype relation matrices, that get combined in a different way for each relation depending in its embedding. In TuckER the dimensions of entity and relation embeddings are independent from each other, with entity embeddings $\boldsymbol{e} \in \mathbb{R}^{d_e}$ and relation embeddings $\boldsymbol{r} \in \mathbb{R}^{d_r}$. The shape of $\boldsymbol{W}$ depends on the dimensions of entities and relations, with $\boldsymbol{W} \in \mathbb{R}^{d_e \times d_r \times d_e}$. In Table 1, we denote with $\times_i$ the tensor product along mode i used by TuckER.

3.2 Geometric Models

Geometric Models interpret relations as geometric transformations in the latent space. Given a fact, the head embedding undergoes a spatial transformation τ that uses the values of the relation embedding as parameters. The fact score is the distance between the resulting vector and the tail vector; such an offset is computed using a distance function δ (e.g. L1

of L2 norm).

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

Depending on the analytical form of τ, Geometric models may share similarities with Tensor Decomposition models, but in these cases geometric models usually need to enforce additional constraints in order to make their τ implement a valid spatial transformation. For instance, the rotation operated by model RotatE can be formulated as a matrix product, but the rotation matrix would need to be diagonal and to have elements with modulus 1.

Much like with Matrix Factorization Models, these systems usually 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 one-dimensional vectors of same length.

TransE [8] was the first LP model to propose a geometric interpretation of the latent space, largely inspired by the capability observed in Word2vec vectors [39] to capture relations between words in the form of translations between their embeddings. TransE enforces this explicitly, requiring that the tail embedding lies close to the sum of the head and relation embeddings, according to the chosen distance function. Due to the nature of translation, TransE is not able to correctly handle one-to-many and many-to-one relations, as well as symmetric and transitive relations.

3.2.2 Translational models with Additional Embeddings. These models may associate more than one embedding to each KG element. This often amounts to using specialized embeddings, such as relation-specific embeddings for each entity or, vice-versa, entity-specific embeddings for each relation. As a consequence, these models overcome the limitations of purely translational models at the cost of learning a larger number of parameters.

STransE [41], in addition to the d-sized embeddings seen in TransE, associates to each relation r 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 operating 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 use relation-specific embeddings for the head and tail, alleviating the issues suffered by TransE on 1-to-many, many-to-one and many-to-many relations.

CrossE [72] is one of the most recent and also most effective models in this group. For each relation it learns an additional relation-specific embedding $\boldsymbol{c_r}$. Given any fact $\langle h, r, t \rangle$, CrossE uses element-wise products (denoted by $\odot$ in Table 1) to combine $\boldsymbol{h}$ and $\boldsymbol{r}$ with $\boldsymbol{c_r}$. This results in triple-specific embeddings, dubbed interaction embeddings, that are then used in the translation. Interestingly, despite not relying on neural layers, this model adopts the common deep learning practice to interpose operations with non-linear activation functions, such as *hyperbolic tangent* and *sigmoid* denoted (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 perform rotation-like transformations either in combination or in alternative to translations.

TorusE [13] was motivated by the observation that the regularization used in TransE forces entity embeddings to lie on a hypersphere, thus limiting their capability to satisfy the translational constraint. To solve this problem, TorusE projects each point $\boldsymbol{x}$ of the traditional open manifold $\mathbb{R}^d$ into a $\boldsymbol{[x]}$ point on a torus $\mathbb{T}^d$. The authors define torus

distance functions d_{L1}, d_{L2} and d_{eL2}, corresponding to L1, L2 and squared L2 norm respectively (we report in Table 1 the scoring function with the extended form of d_{L1}).

RotatE [55] represents 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 complex component conveys the rotation along that axis, whereas the real component is always equal to 1. The rotation $\boldsymbol{r}$ is applied to $\boldsymbol{h}$ by operating an element-wise product (once again noted with $\odot$ in 1). L1 norm is used for measuring the distance from $\boldsymbol{t}$. The authors demonstrate that rotation allows to model correctly numerous relational patterns, such as symmetry/anti-symmetry, inversion and composition.

3.3 Deep Learning Models

Deep Learning Models use deep neural networks to perform the LP task. Neural Networks learn parameters such as weights and biases, that they combine with the input data in order to recognize significant patterns. Deep neural networks usually organize parameters into separate layers, generally interspersed with non-linear activation functions.

In time, numerous types of layers have been developed, applying very different operations to the input data. 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 heavier, harder 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 use one or multiple convolutional layers [33]: each of these layers performs 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 [11] represents entities and relations as one-dimensional d-sized embeddings. When computing the score of a fact, it concatenates and reshapes the head and relation embeddings $\boldsymbol{h}$ and $\boldsymbol{r}$ into a unique input $[\boldsymbol{h}; \boldsymbol{r}]$; we dub the resulting dimensions $d_m \times d_n$. This input is let through a convolutional layer with a set $\boldsymbol{\omega}$ of $m \times n$ filters, and then through a dense layer with d neurons and a set of weights W. The output is finally combined with the tail embedding $\boldsymbol{t}$ using dot product, resulting in the fact score. When using the entire matrix of entity embeddings instead of the embedding of just the one target entity $\boldsymbol{t}$, this architecture can be seen as a classifier with $|\mathcal{E}|$ classes.

ConvKB [42] models entities and relations as same-sized one-dimensional embeddings. Differently from ConvE, given any fact $\langle h, r, t \rangle$, it concatenates all their embeddings $\boldsymbol{h}$, $\boldsymbol{r}$ and $\boldsymbol{t}$ into a $d \times 3$ input matrix $[\boldsymbol{h}; \boldsymbol{r}; \boldsymbol{t}]$. This input is passed to a convolutional layer with a set $\boldsymbol{\omega}$ of T filters of shape 1×3, resulting in a $T \times 3$ feature map. The feature map is let through a dense layer with only one neuron and weights $\boldsymbol{W}$, resulting in the fact score. This architecture can be seen as a binary classifier, yielding the probability that the input fact is valid.

ConvR [25] represents entity and relation embeddings as one-dimensional vectors of different dimensions d_e and d_r. For any fact $\langle h, r, t \rangle$, $\boldsymbol{h}$ is first reshaped into a matrix of shape d_{e_m}, d_{e_n}, where $d_{e_m} \times d_{e_n} = d_e$. $\boldsymbol{r}$ is then reshaped

and split into a set $\boldsymbol{\omega_r}$ of T convolutional filters, each of which has size $m \times n$. These filters are then employed to run convolution on $\boldsymbol{h}$; this amounts to performing an adaptive convolution with relation-specific filters. The resulting feature maps are passed to a dense layer with weights W, As in ConvE, the fact score is obtained combining the neural output with the tail embedding $\boldsymbol{t}$ using dot product.

3.3.2 Capsule Neural Networks. Capsule networks (CapsNets) are composed of groups of neurons, called **capsules**, that encode specific features of the input, such as the presence of a specific object in an image [49]. CapsNets are designed to recognize such features without losing spatial information the way that convolutional networks do. Each capsule sends its output to higher order ones, with connections decided by a dynamic routing process. The probability of a capsule detecting the feature is given by the length of its output vector.

CapsE [43] embeds entities and relations into d-sized one-dimensional vectors, under the basic assumption that different embeddings encode homologous aspects in the same positions. Similarly to ConvKB, it concatenates $\boldsymbol{h}$, $\boldsymbol{r}$ and $\boldsymbol{t}$ into one $d \times 3$ input matrix. This is let through a convolutional layer with E 1×3 filters. The result is a $d \times E$ matrix in which the i-th value of any row uniquely depends on $\boldsymbol{h}[i]$, $\boldsymbol{r}[i]$ and $\boldsymbol{t}[i]$. The matrix is let through a capsule layer; a separate capsule handles each column, thus receiving 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 the capsule layers with *capsnet*.

3.3.3 Recurrent Neural Networks (RNNs). These models employ one or multiple recurrent layers [22] to analyze entire paths (sequences of facts) extracted from the training set, instead of just processing individual facts separately.

RSN [19] is based on the observation that basic RNNs may be unsuitable for LP, because they do not explicitly handle the path alternation of entities and relations, and when predicting a fact tail, in the current time step they are only passed its relation, and not the head (seen in the previous step). To overcome these issues, they propose Recurrent Skipping Networks (RSNs): in any time step, if the input is a relation, the hidden state is updated re-using the fact head too. The fact score is computed performing the dot product between the output vector and the target embedding. In training, the model learns relation paths built from the train facts using biased random walk sampling. It employs a specially optimized loss function resorting to a type-based noise contrastive estimation. In Table 1 we denote the RSN operation with *rsn*; the number of layers stacked in a RSN cell as L; the number of weight matrices as k; the number of neurons in each RSN layer as n.

4 METHODOLOGY

In this section we describe the implementations and training protocols of the models discussed before, as well as the datasets and procedures we use to study their efficiency and effectiveness.

4.1 Datasets

Datasets for benchmarking LP are usually obtained by sampling real-world KGs, and then splitting the obtained facts into a training, a validation and a test set. We conduct our analysis using the 5 best-established datasets in the LP field; we report some of their most important properties in Table 2.

FB15k is probably the most commonly used benchmark so far. Its creators [8] selected all the FreeBase entities with more than 100 mentions and also featured in the Wikilinks database;[2] they extracted all facts involving them (thus also

[2]https://code.google.com/archive/p/wiki-links/

Family and Group		Model	Loss	Constraints	Space Complexity
Tensor Decomposition Models	Bilinear	**DistMult**	$\boldsymbol{h} \times \boldsymbol{r} \times \boldsymbol{t}$	$\forall r \in \mathcal{R} : \boldsymbol{r}$ is diagonal;	$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
		ComplEx	$\boldsymbol{h} \times \boldsymbol{r} \times \bar{\boldsymbol{t}}$	$\boldsymbol{h} \in \mathbb{C}^d;\ \boldsymbol{t} \in \mathbb{C}^d;\ \boldsymbol{r} \in \mathbb{C}^{d \times d};$ $\forall r \in \mathcal{R} : \boldsymbol{r}$ is diagonal;	$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
		Analogy	$\boldsymbol{h} \times \boldsymbol{r} \times \boldsymbol{t}$	$\forall r \in \mathcal{R} : \boldsymbol{r} \times \boldsymbol{r}^T = \boldsymbol{r}^T \times \boldsymbol{r};$ $\forall (r_1, r_2) \in \mathcal{R} \times \mathcal{R} : \boldsymbol{r_1} \times \boldsymbol{r_2} = \boldsymbol{r_2} \times \boldsymbol{r_1};$	$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
		SimplE	$\frac{1}{2}(\boldsymbol{h_h} \times r \times \boldsymbol{t_t}) + \frac{1}{2}(\boldsymbol{h_t} \times \boldsymbol{r_{-1}} \times \boldsymbol{t_h})$	$\forall r \in \mathcal{R} : \boldsymbol{r}, \boldsymbol{r_{-1}}$ are diagonal;	$\mathcal{O}(2\lvert\mathcal{E}\rvert d + 2\lvert\mathcal{R}\rvert d)$
	Non-bilinear	**HolE**	$(\boldsymbol{h} \bigstar \boldsymbol{t}) \times \boldsymbol{r}$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
		TuckER	$\boldsymbol{W} \times_1 \boldsymbol{h} \times_2 \boldsymbol{r} \times_3 \boldsymbol{t}$		$\mathcal{O}(\lvert\mathcal{E}\rvert d_e + \lvert\mathcal{R}\rvert d_r + d_e d_r d_e)$
Geometric Models	Pure Translation	**TransE**	$\lVert \boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t} \rVert$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
	Additional Embeddings	**STransE**	$\lVert \boldsymbol{W_r^h} \times \boldsymbol{h} + \boldsymbol{r} - \boldsymbol{W_r^t} \times \boldsymbol{t} \rVert$		$\mathcal{O}\big(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert (d + 2d^2)\big)$
		CrossE	$\sigma(tanh(\boldsymbol{h} \odot \boldsymbol{c_r} + \boldsymbol{r} \odot \boldsymbol{h} \odot \boldsymbol{c_r}) \times \boldsymbol{t}^T)$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + 2\lvert\mathcal{R}\rvert d)$
	Roto-translation	**TorusE**	$min_{(\boldsymbol{x},\boldsymbol{y}) \in ([\boldsymbol{h}]+[\boldsymbol{r}]) \times [\boldsymbol{t}]} \lVert \boldsymbol{x} - \boldsymbol{y} \rVert_i$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
		RotatE	$-\lVert \boldsymbol{h} \odot \boldsymbol{r} - \boldsymbol{t} \rVert$	$\boldsymbol{h} \in \mathbb{C}^d;\ \boldsymbol{r} \in \mathbb{C}^d;\ \boldsymbol{t} \in \mathbb{C}^d;$ $\forall \boldsymbol{r_i} \in \boldsymbol{r} : \lvert \boldsymbol{r_i} \rvert = 1;$	$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
Deep Learning Models	Convolution	**ConvE**	$g(\boldsymbol{W} \times g([\boldsymbol{h}\,;\boldsymbol{r}] \circledast \boldsymbol{\omega}) + \mathrm{b}) \times \boldsymbol{t}$		$\mathcal{O}\big(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d + Tmn + Td(2d_m - m + 1)(d_n - n + 1)\big)$
		ConvKB	$g(\boldsymbol{W} \times g([\boldsymbol{h}\,;\boldsymbol{r}\,;\boldsymbol{t}] \circledast \boldsymbol{\omega}) + \mathrm{b})$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d + 4T)$
		ConvR	$g(\boldsymbol{W} \times g([\boldsymbol{h}] \circledast \boldsymbol{\omega_r}) + \mathrm{b}) \times \boldsymbol{t}$		$\mathcal{O}\Big(\lvert\mathcal{E}\rvert d_e + \lvert\mathcal{R}\rvert d_r + Tmn + Td_e\big(2d_{e_m} - m + 1\big)\big(d_{e_n} - n + 1\big)\Big)$
	Capsule	**CapsE**	$\lVert capsnet(g([\boldsymbol{h}\,;\boldsymbol{r}\,;\boldsymbol{t}] \circledast \boldsymbol{\omega})) \rVert$		$\mathcal{O}(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d + 3T + Td)$
	Recurrent	**RSN**	$\sigma(rsn(\boldsymbol{h}, \boldsymbol{r}) \times \boldsymbol{t})$		$\mathcal{O}(2\lvert\mathcal{E}\rvert d + 2\lvert\mathcal{R}\rvert d + Lknd)$

Table 1. Loss Function, constraints and space complexity for the models included in our analysis.

including their lower-degree neighbors), except the ones with literals, e.g. dates, proper nouns, etc. They also converted *n*-ary relations represented with reification 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 the authors of TransE [8], was extracted from WordNet[3], a linguistig 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.

FB15k-237 is a subset of FB15k built by Toutanova and Chen [57], inspired by the observation that FB15k suffers from *test leakage*, consisting in test data being seen by models at training time. In FB15k this issue is due to the presence of relations that are near-identical or the inverse of one another. In order to assess the severity of this problem, Toutanova and Chen have shown that a simple model based on observable features can easily reach state-of-the-art performance on FB15k. FB15k-237 was built to be a more challenging dataset: the authors first selected facts from FB15k involving

[3]https://wordnet.princeton.edu/

	Entities	Relations	Triples			Reified	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.

the 401 largest relations and removed all equivalent or inverse relations. 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 subset of WN18 built by Dettmers et al. [11], also after observing test leakage in WN18. They demonstrate the severity of said leakage by showing that a simple rule-based model based on inverse relation detection, dubbed Inverse Model, achieves state-of-the-art results in both WN18 and FB15k. To resolve that, they build the far more challenging WN18RR dataset by applying a pipeline similar to the one employed for FB15k-237 [57]. It has been recently acknowledged by the authors [56] that the test set includes 212 entities that do not appear in the training set, making it impossible to reasonably predict about 6.7% test facts.

YAGO3-10, sampled from the YAGO3 KG [36], was also proposed by Dettmers et al. [11]. It was obtained selecting entities with at least 10 different relations and gathering all facts involving them, thus also including their neighbors. Moreover, unlike FB15k and FB15k-237, YAGO3-10 also keeps the facts about textual attributes found in the KG. As a consequence, as stated by the authors, the majority of its triples deals with descriptive properties of people, such as citizenship or gender. That the poor performances of the Inverse Model [11] in YAGO3-10 suggest that this benchmark should not suffer from the same test leakage issues as FB15k and WN18.

4.2 Efficiency Analysis

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

- *Training Time*: the time required to learn the optimal embeddings for all entities and relations.
- *Prediction Time*: the time required to generate the 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 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, for the Training Time, the number of training triples to process. 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); (*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, for the Training Time, the number of training triples to process.

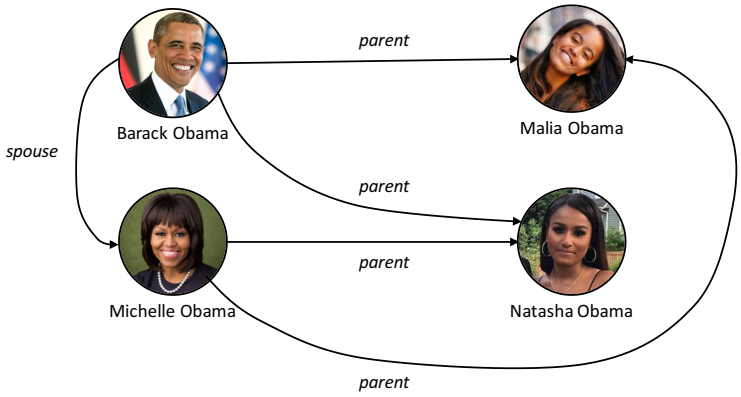


Fig. 2. Example of head peers and tail peers in a small portion of a KG.

4.3 Effectiveness Analysis

We analyze the effectiveness of LP models based on the structure of the training graph. Therefore, 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;
- *Degree of the original reified relation*, for datasets generated from KGs using reification.

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

4.3.1 Number of Peers.

- *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 for h seen during training, conditioned to having relation r and tail t. Analogously, tail peers are the alternatives for t when the head is h and the relation is r. Consistently to the notation introduced in Section 2, we identify the peers for the source and the target entity of a prediction as *source peers* and *target peers* respectively.

We illustrate an example in Figure 2: considering the fact ⟨*Barack Obama*, *parent*, *Malia Obama*⟩, the entity *Michelle Obama* would be a peer for *Barack Obama*, because entity *Michelle Obama* is *parent* to *Malia Obama* too. Analogously, entity *Natasha Obama* is a peer for *Malia Obama*. In head prediction, when *Malia Obama* is the source entity and *Barack Obama* is the target entity, *Michelle Obama* is a target peer and *Natasha Obama* is a source peer. In tail prediction peers are just reversed: since now *Malia Obama* is target entity and *Barack Obama* is source entity, *Michelle Obama* is a source peer whereas *Natasha Obama* is a target peer.

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

On the one hand, the number of source peers can be seen as the number of training samples from which models can directly learn how to predict the current target entity given the current relation. For instance, when performing tail prediction on fact ⟨*Barack Obama*, *nationality*, *USA*⟩, the source peers are all the other entities with *nationality USA* that the model gets to see in training: they are the examples from which our models can learn what can make a person have American citizenship.

On the other hand, the number of target peers can be seen as the number of answers correctly satisfying this prediction seen by the model during training. For instance, given the same fact as before ⟨*Barack Obama*, *nationality*, *USA*⟩, but performing head prediction this time, the other USA citizens seen in training are now target peers. Since all of them constitute valid alternatives for the target answers, too many 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 any KG a *path* is a sequence of 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 consecutive facts it contains. In what follows, we call the sequence of relation names (ignoring entities) in a path a *relational path*.

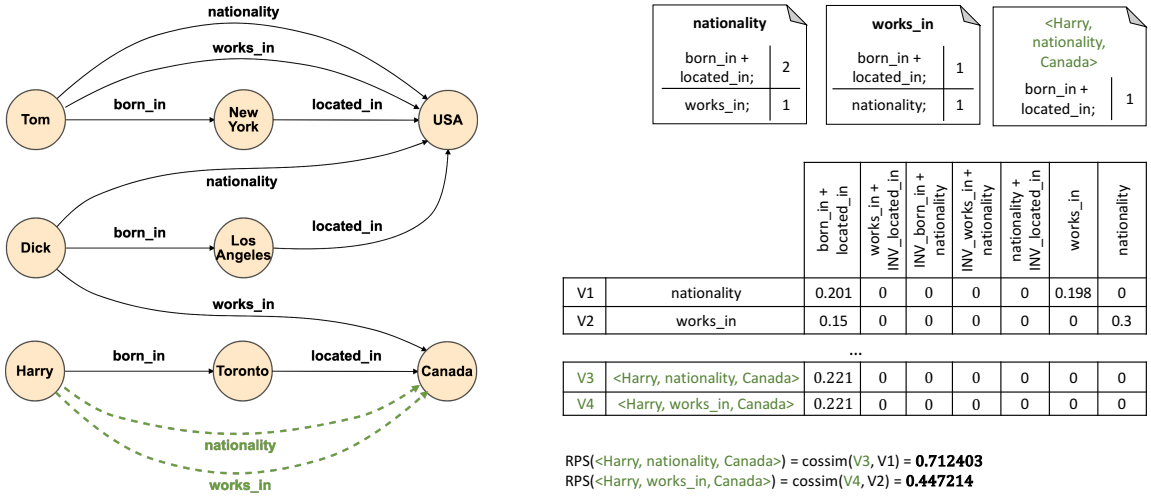
Relational paths allow one to identify patterns corresponding to specific relations. For instance, knowing the facts ⟨*Barack Obama*, *place of birth*, *Honolulu*⟩ and ⟨*Honolulu*, *located in*, *USA*⟩, it should be possible to predict that ⟨*Barack Obama*, *nationality*, *USA*⟩. Paths have been leveraged for a long time by LP techniques based on observable features, such as the Path Ranking Algorithm [31],[32]. The same cannot be said about models based on embeddings, in which the majority of them learn individual facts separately. Just a few models directly rely on paths, e.g. PTransE [34] or, in our analysis, RSN [19]; some models do not employ paths directly in training but use them for additional tasks, as the explanation approach proposed by CrossE [72].

Our intuition is that even models that train on individual facts, as they progressively scan and learn the entire training set, acquire indirect knowledge of its paths as well. As a consequence, in a roundabout way, they may be able to leverage to some extent the patterns observable in paths in order to make better predictions.

Therefore we investigate how the support provided by paths in training can make test predictions easier for embedding-based models. We define a novel measure of *Relational Path Support* (RPS) that estimates for any fact how the paths connecting the head to the tail facilitate their prediction. In greater detail, the RPS value for a fact $\langle h, r, t \rangle$ 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 TF-IDF statistical measure [50] 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 we treat each relation path p as a word and each relation r as 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:



		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

Fig. 3. Example for Relational Path Support

(1) For each training fact $\langle h, r, t \rangle$ we extract from $\mathcal{G}_{train}$ the set of relational paths p leading from h to t. Whenever in a path a step does not have the correct orientation, we reverse it and mark its relation with the prefix "INV". Our vocabulary V is the set of resulting relational paths. Due to computational constraints, we limit ourselves to relational paths with length equal or lesser than 3.
(2) We aggregate the extracted sets by the relation of the training fact. We obtain, for each relation r:
   - the number n_r of training facts featuring r;
   - for each relational path $p \in V$, the number n_{r_p} of times that r is supported by p. 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 connecting its head to its tail analogously to 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: $RPS(q) = cossim(TF - IDF_q, TF - IDF_{r_q})$.

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 measure 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.

Example 4.1. Figure 3 shows a graph where black solid edges represent training facts and green dashed edges represent test facts. The collection of documents is $C = \{nationality, works\_in, born\_in, located\_in\}$, and test facts $\langle$ *Harry*, *nationality*, *Canada* $\rangle$ and $\langle$ *Harry*, *works_in*, *Canada* $\rangle$ correspond to two queries. We compute words and

frequencies for each document and query. Note that the two test facts in our example connect the same head to the same tail, so the corresponding queries have the same keywords (the relational path *born_in + located_in*).

We obtain TF-IDF values for each word in each document as described above. For instance, for document *nationality* and word *born_in + located_in*:

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

Other values can be computed analogously; for instance, $TFIDF(\ born\_in + located\_in,\ works\_in\ ) = 0.15$; $TFIDF(\ works\_in,\ nationality\ ) = 0.198$.

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.

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

The RPS for ⟨ *Harry*, *nationality*, *Canada* ⟩ is the cosine-similarity between its vector 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 RPS value is higher than the latter: the relational paths connecting *Harry* with *Canada* are more similar to the those usually observed with *nationality* than those usually observed with *works_in*. In other words, in our small example the relation path *born_in + located_in* co-occurs with *nationality* more than with *works_in*.

While the number of peers only depends on the local neighborhood of the source and target entity, RPS relies on paths that typically have length 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.

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

4.3.3 Relation Properties. Depending on their semantics, relations can be characterized by several properties heavily affecting the ways in which they appear in the facts. Such properties have been well known in the LP literature for a long time, because they may lead a relation to form very specific structures and patterns in the graph; this, depending on the model, can make their facts easier or harder to learn and predict.

As a matter of fact, depending on their scoring function, some models may be even incapable of learning certain types of relations correctly. For instance, TransE [8] and some of its successors are inherently unable to learn symmetric and transitive relations due to the nature of translation itself. Analogously, DistMult [71] 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.

This has led some works to formally introduce the concept of **full expressiveness** [27]: a model is fully expressive if, given any valid graph, there exists at least one combination of embedding values for the model that correctly separates all correct triples from incorrect ones. A fully expressive model has the theoretical potential to learn correctly any valid graph, without being hindered by intrinsic limitations. Examples of models that have been demonstrated to be fully expressive are SimplE [27], TuckER [6], ComplEx [61] and HolE [60].

Being capable of learning certain relations, however, does not necessarily imply reaching good performance on them. Even for fully expressive models, certain properties may be inherently harder to handle than others. For instance,

Meilicke *et al.* [37] have analyzed how their implementations of HolE [46], RESCAL [44] and TransE [8] perform on symmetric relations in various datasets; they report surprisingly bad results for HolE on symmetric relations in FB15K, despite HolE being fully expressive).

At this regard, we lead a systematical analysis: we define a comprehensive set of relation properties and verify how they affect performance for all our models.

We take into account the following properties:

- *Reflexivity*: in the original definition, a reflexive relation connects each element with itself. This is not suitable for KGs, where different entities may only be involved with some relations, based on their type. As a consequence, in our analysis we use the following definition: $r \in \mathcal{R}$ is reflexive if $\forall \langle h, r, t \rangle \in \mathcal{G}_{train}$, $\langle h, r, h \rangle \in \mathcal{G}_{train}$ too.
- *Irreflexivity*: $r \in \mathcal{R}$ is irreflexive if $\forall e \in \mathcal{E} \quad \langle e, r, r \rangle \notin \mathcal{G}_{train}$.
- *Symmetry*: $r \in \mathcal{R}$ is symmetric if $\forall \langle h, r, t \rangle \in \mathcal{G}$, $\langle t, r, h \rangle \in \mathcal{G}$ too.
- *Anti-symmetry*: $r \in \mathcal{R}$ is anti-symmetric if $\forall \langle h, r, t \rangle \in \mathcal{G}$, $\langle t, r, h \rangle \notin \mathcal{G}$.
- *Transitivity*: $r \in \mathcal{R}$ is transitive if $\forall$ pair of facts $\langle h, r, x \rangle \in \mathcal{G}$ and $\langle x, r, t \rangle \in \mathcal{G}$, $\langle h, r, t \rangle \in \mathcal{G}$ as well.

We do not consider other properties, such as Equivalence and Order (partial or complete), because we experimentally observed that in all datasets included in our analysis only a negligible number of facts would be included in the resulting buckets.

On each dataset we use the following approach. First, for each relation in the dataset we extract the corresponding training facts and use them to identify its properties. Due to the inherent incompleteness of the training set, we employ a tolerance threshold: a property is verified if the ratio of training facts showing the corresponding behaviour exceeds the threshold. In all our experiments, we set tolerance to 0.5. Then, we group the test facts based on the properties of their relations. If a relation possesses multiple properties, its test facts will belong to multiple groups. Finally, we compute predictive performance scored by each model on each group of test facts.

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, connecting more than two entities at a time. In relations, cardinality is closely related to semantics, and some relations inherently make more sense when modeled in this way. For example, an actor winning an award for her performance in a movie can be modeled with a unique relation connecting the actor, the award and the movie. KGs that support relations with cardinality greater than 2 often handle them in one of the following ways:

- 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 a set of triples.
- using **reification**: if a relation needs to connect multiple entities, it is modeled with an intermediate node linked to those entities by binary relations. The relation cardinality thus becomes the degree of the reified node. Reification allows relations with cardinality greater than 2 to be indirectly modeled; the graph is thus still representable as a set of triples.

The popular KG FreeBase, that has been used to generate important LP datasets such as FB15k and FB15k-237, employs reified relations, with intermediate nodes of type *Compound Value Type* (CVT). By extension, we refer to such intermediate nodes as CVTs.

In the process of extracting FB15k from FreeBase [8], CVTs were removed and converted into cliques in which the entities previously connected to the CVT are now connected to one another; the labels of the new edges are obtained

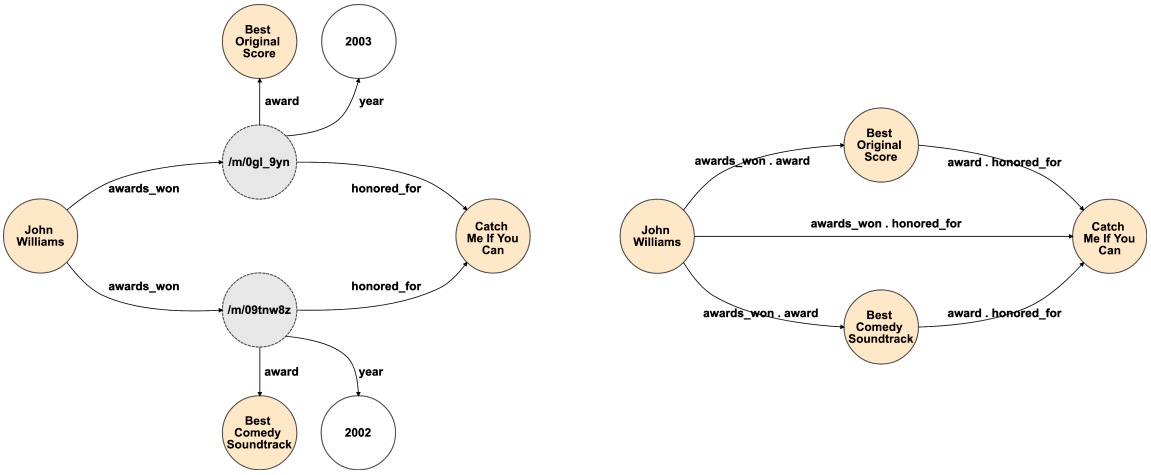


Fig. 4. Example of how the Star2Clique process operates on a small portion of a KG.

concatenating the corresponding old ones. This also applies to FB15k-237, that was obtained by just sampling FB15k further [57]. It has been pointed out that this conversion, dubbed "Star-to-Clique" (S2C), is irreversible [68]. In our study we have observed further consequences to the S2C policy:

- From a structural standpoint, S2C transforms a CVT with degree n into a clique with at most at most $n!$ edges. Therefore, some parts of the graph become locally much denser than before. The generated edges are often redundant, and in the filtering operated to create FB15k-237, many of them are removed.
- From a semantic standpoint, the original meaning of the relations is vastly altered. After exploding CVTs into cliques, deduplication is performed: if the same two entities were linked multiple times by the same types of relation using multiple CVTs – e.g. an artist winning multiple awards for the same work – this information is lost, as shown in Figure 4. In other words, in the new semantics, each fact has happened *at least once*.

We hypothesize that the very dense, redundant and locally-consistent areas generated by S2C may have consequences on predictive behaviour. Therefore, for FB15k and FB15k-237, we have tried to extract for each test fact generated by S2C the degree of the original CVT, in order to correlate it with the predictive performance of our models.

For each test fact generated by S2C we have tried to recover the corresponding CVT from the latest FreeBase dump available.[4] As already pointed out, S2C is not reversible due to its inherent deduplication; therefore, this process often yields multiple CVTs for the same fact. In this case, we have taken into account the CVT with highest degree. We also report that, quite strangely, for a few test facts built with S2C we could not find any CVTs in the FreeBase dump. For these facts, we have set the original reified relation degree to the minimum possible value, that is 2.

In this way, we were able to map each test fact to the degree of the corresponding CVT with highest degree; we have then proceeded to investigate the correlations between such degrees and the predictive performance of our models.

Our results on the analysis of reified relations are reported in Section 5.4.

[4]https://developers.google.com/freebase

5 EXPERIMENTAL RESULTS

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

5.1 Experimental set-up

In this section we provide a brief overview of the environment we have used in our work and of the procedures followed to train and evaluate all our LP models. We also provide a description for the baseline model we use in all the experiments of our analysis.

5.1.1 Environment. All of our experiments, as well as the training and evaluation of each model, have been 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 have trained and evaluated from scratch all the models introduced in Section 3. In order to make our results directly reproducible, we have employed, whenever possible, publicly available implementations. As a matter of fact we only include one model for which the implementation is not available online, that is ConvR [25]; we thank the authors for kindly sharing their code with us.

When we have found multiple implementations for the same model, we have always chosen the best best performing one, with the goal of analyzing each model at its best. This has resulted in the following choices:

- For TransE [8], DistMult [71] and HolE [46] we have used the implementation provided by project Ampligraph [10] and available in their repository [1];
- For ComplEx [61] we have used Timothée Lacroix's version with N3 regularization [30], available in Facebook Research repository [14];
- For SimplE [27] we have used the fast implementation by Bahare Fatemi [5], as suggested by the creators of the model themselves.

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

As shown by Kadlec et al. [26], LP models tend to be extremely sensitive to hyperparameter tuning, and the hyperparameters for any model often need to be tuned separately for each dataset. The authors of a model usually define the space of acceptable values of each hyperparameter, and then run grid or random search to find the best performing combination.

In our trainings, we have relied on the hyperparameter settings reported by the authors for all datasets on which they have run experiments. Not all authors have evaluated their models on all of the datasets we include in our analysis, therefore in several cases we have not found official guidance on the hyperparameters to use. In these cases, we have explored ourselves the spaces of hyperparameters defined by the authors in their papers.

Considering the sheer size of such spaces (often containing thousands of combinations), as well as the duration of each training (usually taking several hours), running a grid search or even a random search was generally unfeasible. We have thus resorted to hand tuning to the best of our possibilities, using what is familiarly called a *panda* approach [29] (in contrast to a *caviar* approach where large batches of training processes are launched). We report in Appendix A the best hyperparameter combination we have found for each model, in Table 6.

The filtered H@1, H@10, MR and MRR results obtained for each model in each dataset are displayed in Table 3. As mentioned in Section 5.1.3, for models relying on *min* policy in their original implementation we report their results obtained with *average* policy instead, as we observed that *min* policy can lead to results not directly comparable to

		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
Tensor 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	**81.56**	**90.53**	**34**	**0.848**	94.53	95.50	3623	0.949	25.72	52.97	202	0.349	42.55	52.12	4907	0.458	**50.48**	**70.35**	1112	**0.576**
	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	1827	0.498
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
	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	60676	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	81.09	87.86	288	0.835	94.63	95.96	233	**0.951**	24.03	48.93	480	0.324	**44.93**	55.97	2530	**0.485**	45.83	66.07	**815**	0.528

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.

those of the other models. We investigate this phenomenon in Section 5.4.1 We note that in AnyBURL [38], for datasets FB15k-237 and YAGO3-10 we used training time 1,000 secs whereas the original papers report slightly better results with a training time of 10000s; this is because, due to our already described necessity for full rankings, when using models trained for 10,000 secs prediction times got prohibitively long. Therefore, under suggestion of the authors themselves, we resorted to using the second best training time 1,000 secs for these datasets. We also note that STransE [41], ConvKB [42] and CapsE [43] use transfer learning and require embeddings pre-trained on TransE [8]. For FB15k, FB15k-237, WN18 and WN18RR we have found and used TransE [8] embeddings trained and shared by the authors themselves across their repositories; for YAGO3-10, on which the authors did not work, we used embeddings trained with the TransE implementation that we used in our analysis [1].

5.1.3 Evaluation Metrics. When it comes to evaluate the predictive performance of models, we focus on the **filtered scenario**, and, when reporting global results, we use all the four most popular metrics: H@1, H@10, MR and MRR.

In our finer-grained experiments, we have run all our experiments computing both H@1 and MRR. The use of a H@K measure coupled with a mean-rank-based measure is very popular in LP works. At this regard, we focus on H@1 instead of using larger values of K because, as observed by Kadlec et al. [26], low values of K allow the emerging of more more marked differences among different models. Similarly, we choose MRR because is a very stable metric, while simple MR tends to be highly sensitive to outliers. In this paper we mostly report H@1 results for our experiments, as we have usually observed analogous trends using MRR.

As described in Section 2, we have observed that the implementations of different models may rely on different policies for handling ties. Therefore, we have modified them in order to extract evaluation results with multiple policies for each model. In most cases we have not found significant variations; nonetheless, for a few models, *min* policy yields significantly different results from the other policies. In other words, using different tie policies may make results not directly comparable to one another.

Therefore, unless differently specified, for models that employed *min* policy in their original implementation we report their *average* results instead, as the latter are directly comparable to the results of the other models, whereas the former are not. We have led further experiments on this topic and report interesting findings in Section 5.4.1

5.1.4 Baseline. As a baseline we use AnyBURL [38], 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, with the goal of covering and satisfying 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 containing $n-1$ 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 $n+1$.

As a matter of fact, AnyBURL is a very challenging baseline for LP, as it is shown to outperform most latent-features-based models. It is also computationally fast: depending on the training setting, in order to learn its rules it can take from a couple of minutes (*100s* setting) to about 3 hours (*10000s* setting). When it comes to 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 needed to extract **full ranking** for each prediction, setting $k = |\mathcal{E}|$: this resulted in much longer computation times.

As a side note, we observe that in AnyBURL predictions, even the full ranking may not contain all entities, as it only includes those supported by at least one rule in the model. This means that in very few facts, the target entity may not be included even in the full ranking. In this very unlikely event, we assume that all the entities that have not been predicted have identical score 0, and we apply the *avg* policy for ties.

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

5.2 Efficiency

In this section we report our results regarding the efficiency of LP models in terms of time required for training and for prediction.

In Figure 5 we illustrate, for each model, the time in hours spent for training on each dataset. We observe that training times range from around 1h to about 200-300 hours. Not surprisingly, the largest dataset YAGO3-10 usually requires significantly longer training times. In comparison to the embedding-based models, the baseline project AnyBURL [38] is strikingly fast. AnyBURL treats the training time as a configuration parameter, and reaches state-of-the-art performance after just 100s for FB15k and WN18, and 1000s for FB15k237, WN18RR and YAGO3-10. As

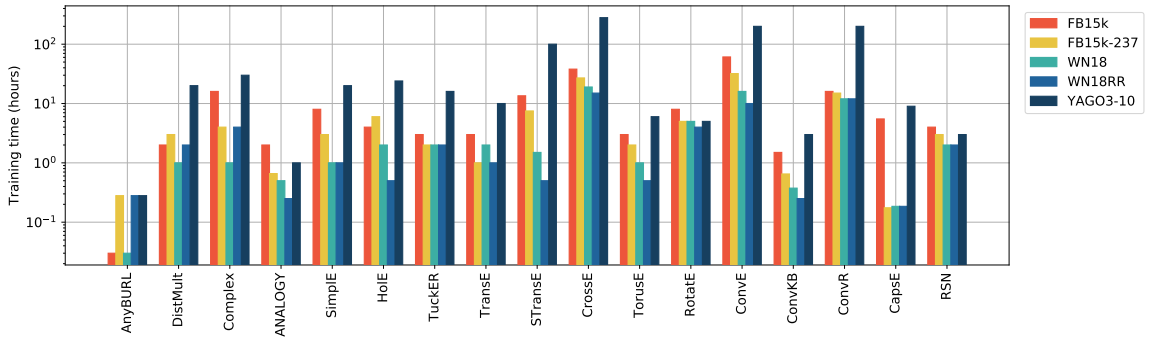


Fig. 5. Training times in hours for each LP model on each dataset. Y axis is in logscale.

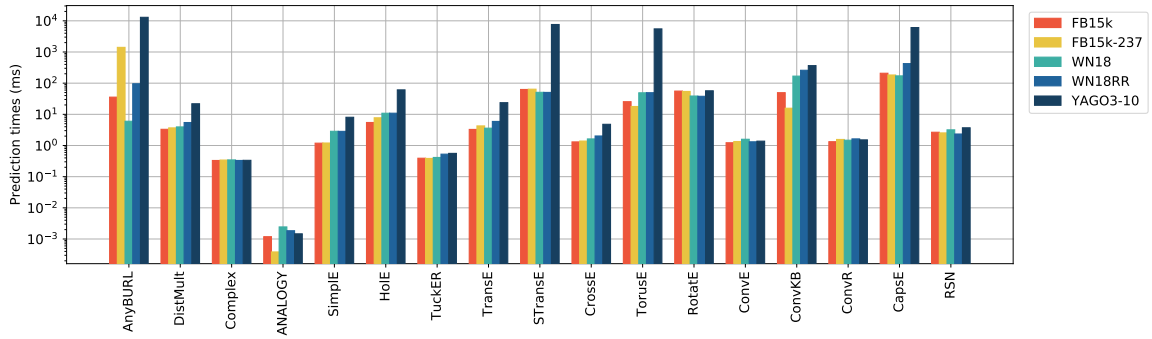


Fig. 6. Prediction times in milliseconds for each LP model on each dataset. Y axis is in logscale.

already mentioned, STransE [41], ConvKB [42] and CapsE [43] require embeddings pre-trained on TransE [8]; we do not include pre-training times in our measurements. In Figure 6 we illustrate, for each model, the prediction time, defined as the time required to generate the full ranking in both head and tail predictions for one fact. These scores are mainly affected by the embedding dimensions and by the evaluation batch size; at this regard we note that ConvKB [42] and CapsE [43] are the only models that require multiple batches for running one prediction, and this may have negatively affected their prediction performance. In our experiments for these models we have used evaluation batch size 2048 (the maximum allowed in our setting). We observe that ANALOGY behaves particularly well in terms of prediction time; this may possibly depend on this model being implemented in C++. For the baseline AnyBURL [38], we have obtained prediction times a posteriori by dividing the we whole rules application times by the numbers of facts in the datasets. The obtained prediction 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, and that using top-k policy lower k values would result in much faster computations.

5.3 Effectiveness

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

5.3.1 Peer Analysis. Our goal in this experiment is to analyze how the predictive performance varies when taking into account test facts with different numbers of source peers, or tail peers, or both.

We report in Figures 7a, 7b, 8a, 8b, 9 our results. These plots show how performances trend when increasing the number of source peers or target peers. We use use H@1 for measuring effectiveness. The plots are incremental, meaning that, for each number of source (target) peers, we report H@1 percentage for all predictions with source (target) peers equal or lesser than that number. In each graph, for each number of source (target) peers we also report the overall percentage of involved test facts: this provides the distribution of facts by peers.

Our observations are intriguing. First, we point out that almost always, predictions with a greater number of source peers show better H@1 results. A way to explain this phenomenon is to consider that the source peers of a prediction are the examples seen in training in which the target entity displays the same role as in the fact to predict. For instance, when performing tail prediction for ⟨ *Barack Obama*, *nationality*, *USA* ⟩, having many source peers means that the model has seen in training numerous examples of people with *nationality USA*. Intuitively, such examples provide the models with meaningful information, allowing them to more easily understand when other entities (such as *Barack Obama)* have *nationality USA* as well.

Second, we observe that very often a greater number of target peers leads to worse H@1 results. For instance, when performing head prediction for ⟨ ? *Michelle Obama*, *born in*, *Chicago* ⟩, target peers are numerous if we have already seen in training many other entities born in Chicago. These entities seem confuse models when they are asked to predict that other entities (such as *Michelle Obama*) are born in *Chicago* as well.

We underline that this decrease in performance is not caused by the target peers just outscoring the target entity: we are taking into account filtered scenario results, therefore target peers, being valid answers to our predictions, do not contribute to the rank computation.

These correlations between numbers of peers and performance are particularly evident in datasets FB15K and FB15K-237. Albeit at 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.

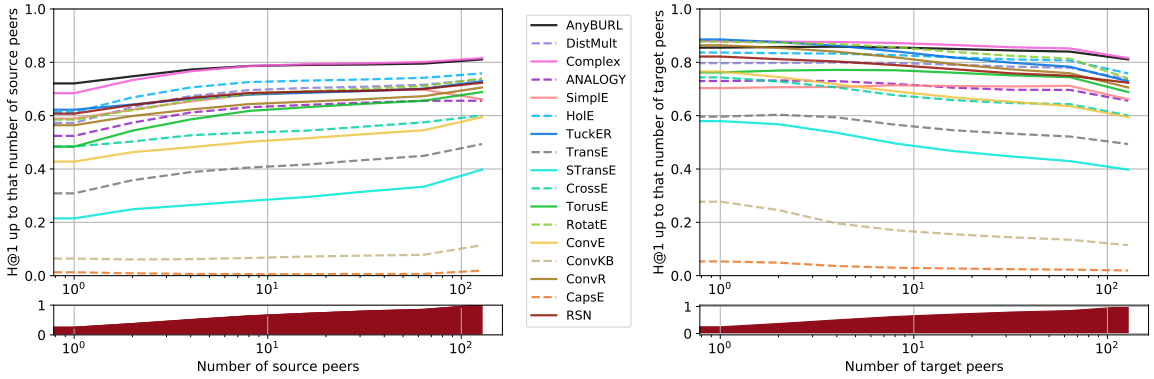
5.3.2 Relational Path Support. Our goal in this experiment is to analyze how the predictive effectiveness of LP models varies when taking into account predictions with different values of Relational Path Support (RPS). RPS is computed using the TF-IDF-based metric introduced in Section 4, using relational paths with length up to 3.

We report in Figures 10a, 10b, 11a, 11b, 12 our results, using H@1 for measuring performance. Similarly to the experiments with source and target peers reported in Section 5.3.2, we use incremental metrics, showing for each value of RPS the percentage and the H@1 of all the facts with support up to that value.

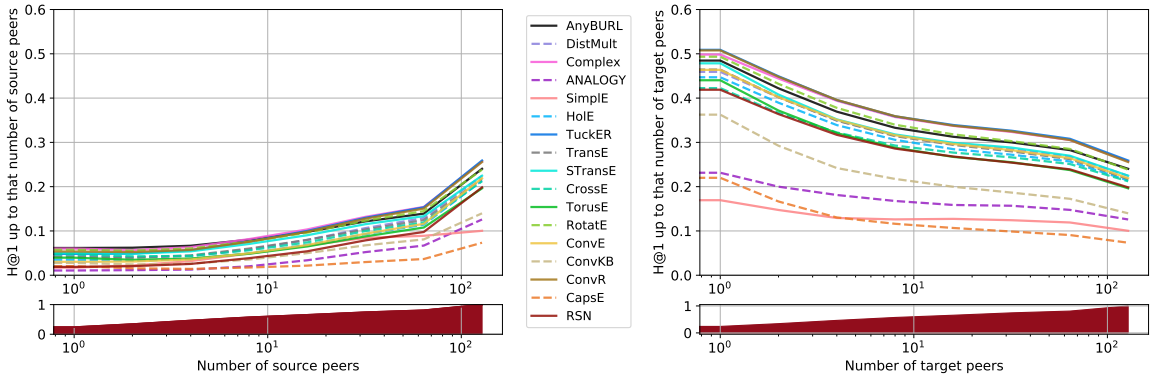
We observe that, for almost all models, greater RPS values lead to better performance. This proves that such models, to a certain extent, are capable of benefiting from longer-range dependencies.

This correlation is visible in all datasets. It is 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 performances, albeit clearly observable, seems weakest; we see this as a consequence of the severe test leakage displayed by FB15k. As a matter of fact, we have



(a) FB15k

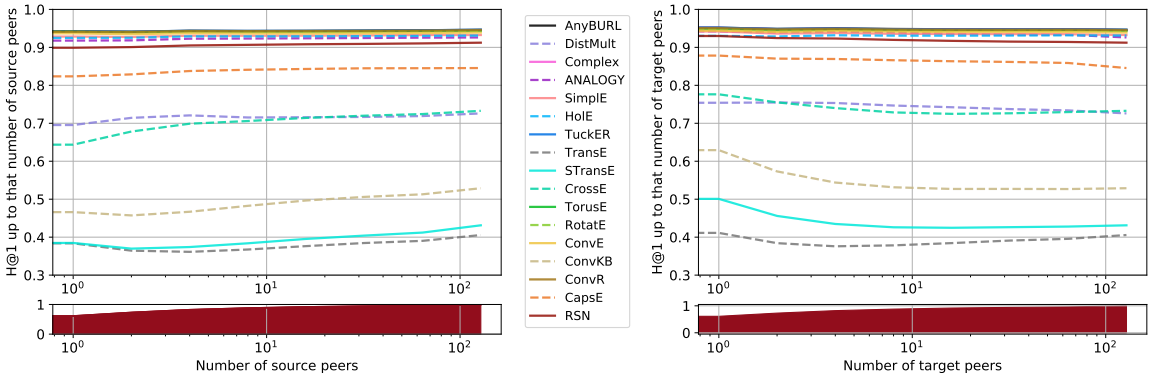


(b) FB15k-237

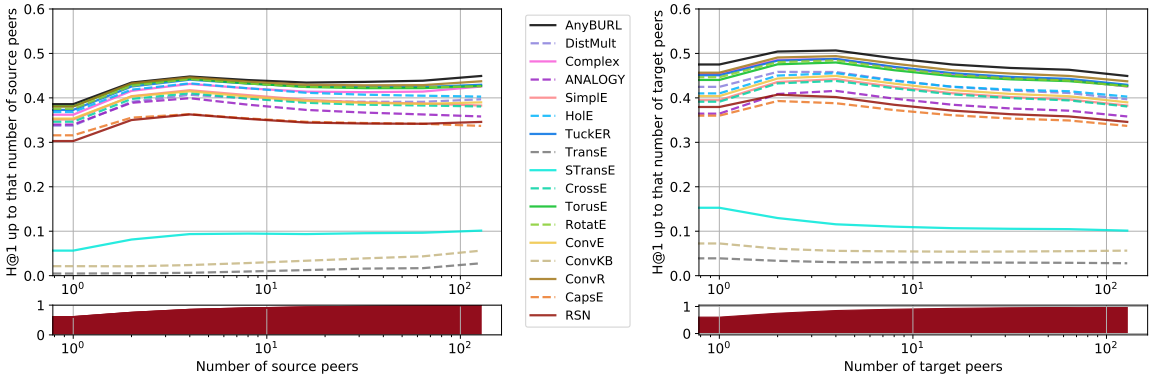
Fig. 7. Cumulative H@1 results for each LP model on the **Freebase** datasets, and corresponding cumulative distribution of test facts, varying the number of source peers (left) and target peers (right). X axis is in logscale.

found evidence suggesting that, in presence of many relations with same or inverse meaning, models tend to focus on shorter dependencies for predictions, ignoring longer relational paths. We show this by replicating our experiment using RPS values computed with relational paths of maximum lengths 1 and 2, instead of 3. We report the FB15k chart Figure 13, and the other charts in Appendix A. In FB15k and WN18, well known for their test leakage, the correlation with performances becomes evidently stronger. In FB15k-237, WN18RR and YAGO3-10, on the contrary, it weakens, meaning that 3-step relational paths are actually associated with correct predictions in these datasets.

Test leakage in FB15k and WN18 is actually so prominent that, on these datasets, we were able to use RPS as the scoring function of a standalone LP model based on observable features, obtaining acceptable results. We report the results of this experiment in Table 4. The evaluation pipeline is the same employed by all LP models and described in Section 2. Due to computational constraints, we use the RPS measure with paths up to 2 steps long, and use on each dataset a sample of 512 test facts instead of the whole test set (a single test fact can take more than 1h to predict). We do not run the experiment on YAGO3-10, on which the very high number of entities would make the ranking loop unfeasible. This experiment can be seen as analogous to the ones run by Toutanova et al. [57] and Dettmers et al. [11],



(a) WN18



(b) WN18RR

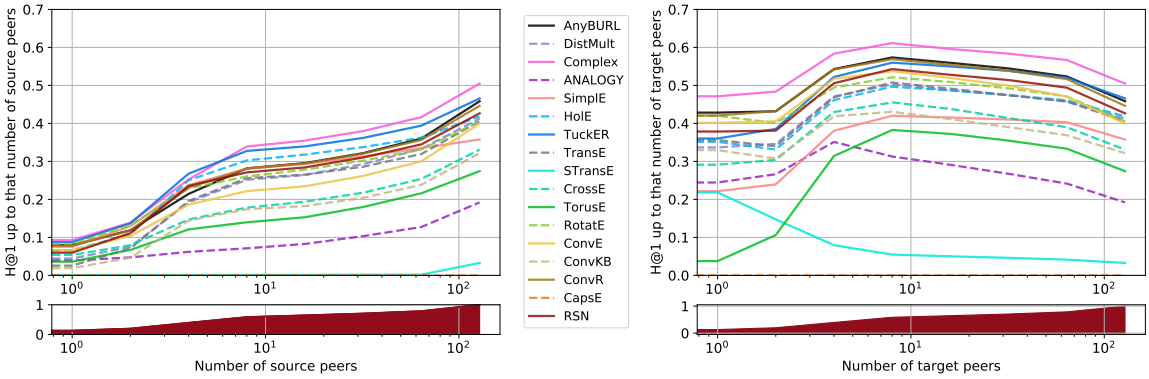
Fig. 8. Cumulative H@1 results for each LP model on the **Wordnet** datasets, and corresponding cumulative distribution of test facts, varying the number of source peers (left) and target peers (right). X axis is in logscale.

	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	328.6	0.559	93.55	94.04	1122.9	0.937	10.45	25.0	2157.0	0.153	36.33	41.11	11492.5	0.38

Table 4. performances of an LP model based on observable features, using as a scoring function the RPS measure with relational paths up to 2 steps long.

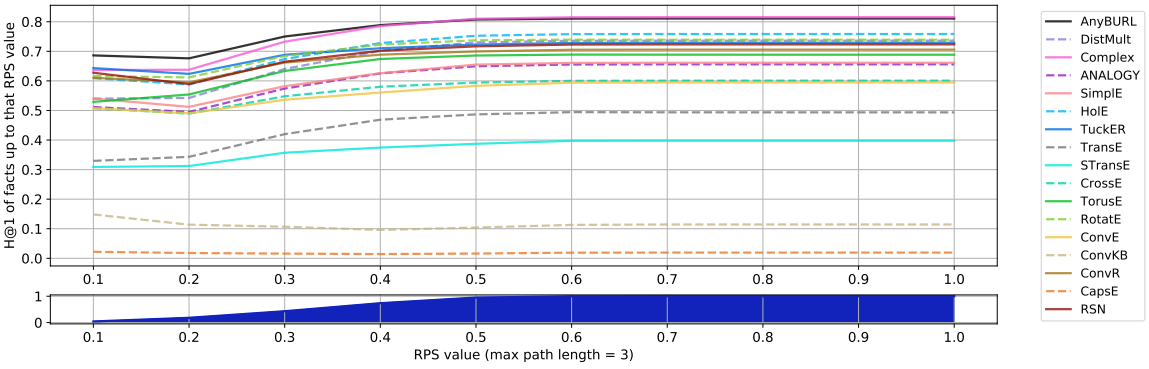
where simple models based on observable features are run on FB15k and WN18 to assess the consequences of their test leakage.

5.3.3 Relation Properties. Our goal in this experiment is to analyze how models perform when the relation in the fact to predict have specific properties. We take into account Reflexivity, Symmetry, Transitivity, Irreflexivity and

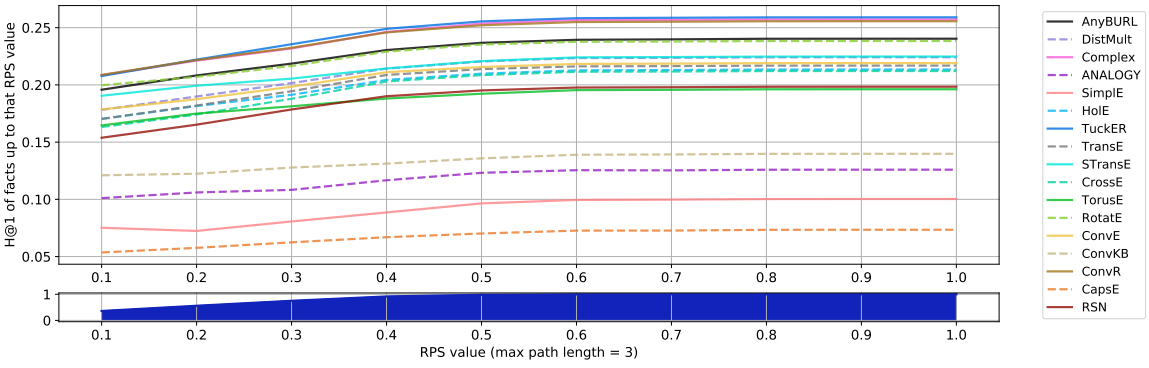


(a) YAGO3-10

Fig. 9. Cumulative H@1 results for each LP model on YAGO3-10, and corresponding cumulative distribution of test facts, varying the number of source peers (left) and target peers (right). X axis is in logscale.

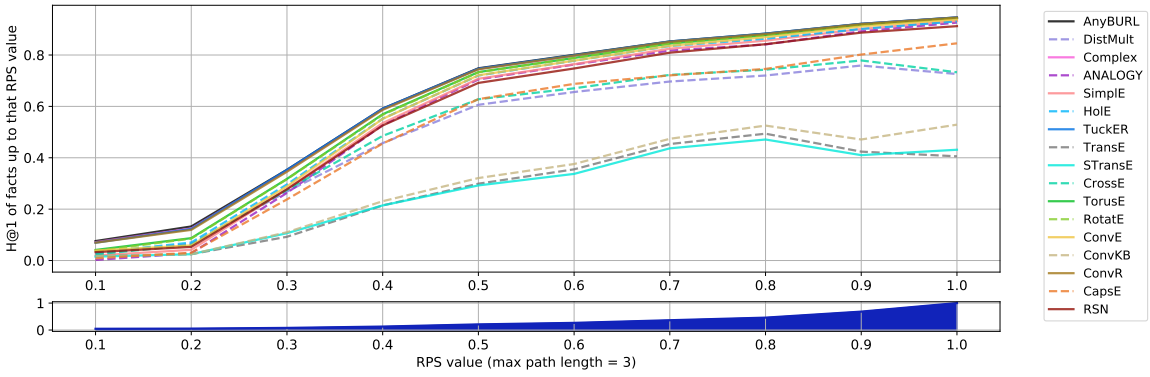


(a) FB15k

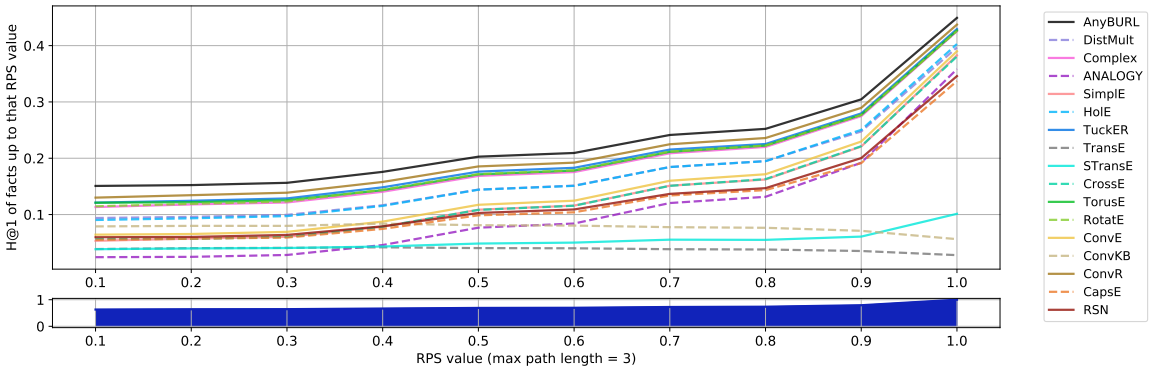


(b) FB15k-237

Fig. 10. H@1 results for each LP model on the **Freebase** datasets varying the RPS of the test facts, and corresponding cumulative distribution of test facts.

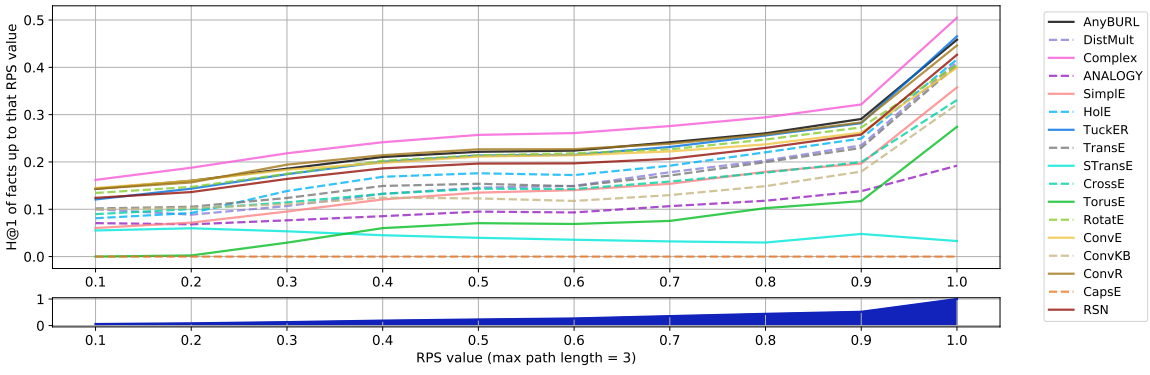


(a) WN18



(b) WN18RR

Fig. 11. H@1 results for each LP model on the **Wordnet** datasets varying the RPS of the test facts, and corresponding cumulative distribution of test facts.



(a) YAGO3-10

Fig. 12. H@1 results for each LP model on YAGO3-10 varying the RPS of the test facts, and corresponding cumulative distribution.

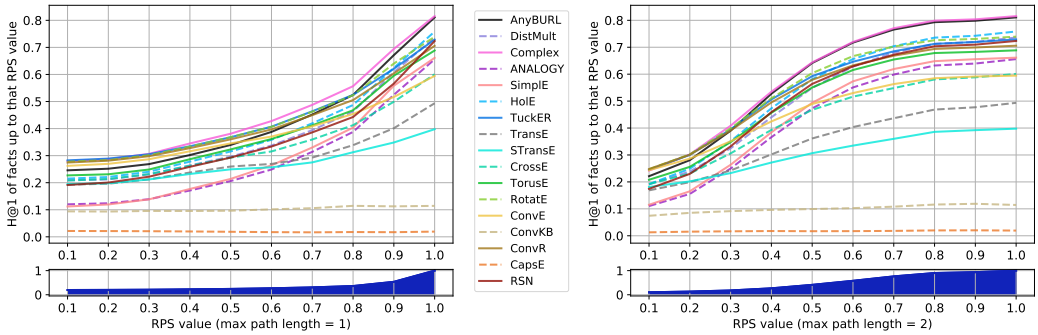


Fig. 13. H@1 results for each LP model on FB15k varying the RPS of the test facts, computing RPS with paths up to length 1 and up to length 2.

Anti-symmetry, as already described in Section 4.3.3. We report in Figures 14a, 14b, 15a, 15b, 16 our results, using H@1 for measuring effectiveness.

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. In all charts we include an initial bucket named *any* containing all the test facts. For each model, this works as a global baseline, as it allows to compare the H@1 of each bucket to the global H@1 of that model. Analogously to the distribution graphs observed in the previous experiments, for each bucket in each dataset we also report the percentage of test facts it contains.

In FB15K and FB15K-237 we observe an impressive majority of irreflexive and anti-symmetric relations. Only a few facts involve reflexive, symmetric or transitive relations. In WN18 and WN18RR 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; there is a high percentage of facts featuring anti-symmetric relations as well, whereas only a few of them involve symmetric or transitive relations.

In FB15K all models based on embeddings seem to perform quite well on reflexive relations; on the other hand, the baseline AnyBURL [38] obtains quite bad results on them possibly due to its rule-based approach We also observe that translational models such as TransE [8], CrossE [72] and STransE [41] struggle to handle symmetric and transitive relations, with very poor results. This problem seems alleviated by the introducton of rotational operations in TorusE [13] and RotatE [55].

In FB15K-237, all models display globally worse performance; nonetheless, interestingly most of them manage to keep good performance on reflexive relations, the exceptions being ANALOGY , SimplE [27], ConvE [11] and RSN [19]. On the contrary they all display terrible performance in symmetric relations. This may depend on 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 situation. This time, symmetric relations are easily handled by most models, with the notable exceptions of TransE [8] and ConvKB [42]. On WN18RR, the good results on symmetric relations balance, for most models, sub-par performance on irreflexive and anti-symmetric relations.

In YAGO3-10 we observe once again TransE [8] and ConvKB [42] having a hard time handling symmetric relations; on these relations, most models actually tend to behave a little worse than their global H@1.

		Any	Reflexive	Irreflexive	Symmetric	Anti Symmetric	Transitive
Tensor Decomposition	DistMult	0.74	0.95	0.74	0.73	0.74	0.75
	ComplEx	**0.82**	0.97	**0.81**	0.88	**0.80**	**0.88**
	ANALOGY	0.66	0.98	0.65	0.52	0.66	0.57
	SimplE	0.66	0.64	0.68	0.74	0.64	0.75
	HolE	0.76	0.86	0.75	0.69	0.76	0.74
	TuckER	0.73	0.98	0.72	0.67	0.73	0.79
Geometric	TransE	0.49	0.94	0.49	0.00	0.55	0.13
	STransE	0.40	0.98	0.40	0.00	0.44	0.11
	CrossE	0.60	**0.99**	0.60	0.20	0.64	0.23
	TorusE	0.69	0.91	0.69	0.69	0.69	0.69
	RotatE	0.74	**0.99**	0.74	0.68	0.74	0.76
Deep Learning	ConvE	0.59	0.65	0.59	0.42	0.61	0.57
	ConvKB	0.11	0.94	0.12	0.07	0.12	0.12
	ConvR	0.71	0.93	0.70	0.63	0.71	0.72
	CapsE	0.02	0.84	0.02	0.01	0.02	0.02
	RSN	0.72	0.65	0.72	0.69	0.73	0.74
	AnyBURL	0.81	0.1	**0.81**	**0.89**	**0.80**	0.86
	Test Facts Percentage	100%	0.3%	87%	9%	85%	3%

(a) FB15k

		Any	Reflexive	Irreflexive	Symmetric	Anti Symmetric	Transitive
Tensor Decomposition	DistMult	0.24	0.00	0.24	**0.07**	0.25	0.36
	ComplEx	0.22	0.86	0.23	0.02	0.23	0.28
	ANALOGY	**0.26**	**1.00**	**0.26**	0.06	**0.27**	**0.38**
	SimplE	0.13	0.00	0.13	0.05	0.13	0.25
	HolE	0.10	0.01	0.11	0.04	0.10	0.25
	TuckER	0.21	0.69	0.21	0.02	0.22	0.29
Geometric	TransE	**0.26**	0.52	**0.26**	0.05	**0.27**	0.37
	STransE	0.22	**1.00**	0.22	0.00	0.23	0.27
	CrossE	0.22	**1.00**	0.23	0.00	0.23	0.34
	TorusE	0.21	0.86	0.21	0.02	0.22	0.30
	RotatE	0.20	**1.00**	0.20	0.03	0.20	0.29
Deep Learning	ConvE	0.24	**1.00**	0.24	0.01	0.25	0.33
	ConvKB	0.22	0.11	0.22	0.02	0.23	0.32
	ConvR	0.14	**1.00**	0.15	0.00	0.15	0.22
	CapsE	**0.26**	0.85	**0.26**	0.04	**0.27**	0.35
	RSN	0.07	**1.00**	0.08	0.01	0.08	0.14
	AnyBURL	0.20	0.03	0.20	0.02	0.21	0.28
	Test Facts Percentage	100%	0.3%	87%	3%	95%	2%

(b) FB15k-237

Fig. 14. H@1 results for each LP model on the **Freebase** datasets and corresponding percentages of test facts, for various relation properties. The best results for each column are in bold and underlined.

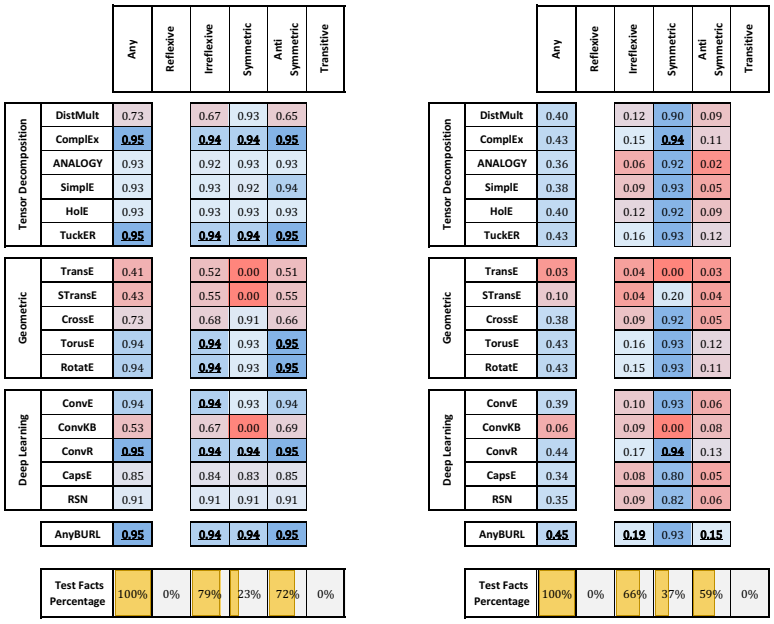
5.4 Reified Relation Degree

Our goal in this experiment is to analyze how, in FreeBase-derived datasets, the degrees of the original reified relations affect predictive performance. Due to the properties of the S2C operations employed to explode reified relations into cliques, a higher degree of the original reified relation corresponds to a locally richer area in the dataset; therefore we expect such higher degrees to correspond to better performance.

We divide test facts into disjoint buckets based on the degree of the original reified relation, extracted as reported in Section 4.

We compute the predictive performance of these buckets separately; we also include a separate bucket with degree value 1, containing the test facts that were not originated from reified relations in FreeBase. We report predictive performances using H@1 in Figures 17a and 17b. We also show, for each bucket, the percentage of test facts it contains with respect to the whole test set.

In FB15K, in most models we observe that a higher degree generally corresponds to better H@1. The main exceptions are TransE [8], CrossE [72] and STransE [41], that show a stable or even worsening pattern. We found that, considering more permissive H@K metrics (e.g. H@10), all models, including these three, improve their performance; we explain this by considering that, due to the very nature of the S2C transformation, original reified relations tend to generate a high number of facts containing symmetric relations. TransE, STransE and CrossE are naturally inclined to represent symmetric relations with very small vectors in the embedding space: as a consequence, when learning facts with symmetric relations, these models tend to place the possible answers very close to each other in the embedding space.



(a) WN18 (b) WN18RR

Fig. 15. H@1 results for each LP model on the **Wordnet** datasets, and corresponding percentages of test facts, for various relation properties. The best results for each column are in bold and underlined.

The result would be a crowded area 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.

In FB15k-237 most of the redundant facts obtained from reified relations have been filtered away, therefore the large majority of test facts belongs to the first bucket.

5.4.1 Sensitivity to tie policy. We have observed that a few models, in their evaluation, are strikingly sensitive to the policy used for handling ties. This happens when models give the same score to multiple different entities in the same prediction: in this case results obtained with different policies diverge, and they are not comparable to one another anymore. In the most extreme case, if a model always gives the same score to all entities in the dataset, using *min* policy it will obtain H@1 = 1.0 (perfect score) whereas using any other policy it would obtain H@1 around 0.0.

In our experiments we have found that CrossE [72] and, to a much greater extent, ConvKB [42] and CapsE [43], seem sensitive to this issue. Note that in their original implementations ConvKB and CapsE use *min* policy by default, whereas CrossE uses *ordinal* policy by default.

In FB15k and FB15k-237 both ConvKB and CapsE display huge discrepancies on all metrics, whereas on WN18 and WN18RR the results are almost identical. On these datasets, no remarkable differences are observable for CrossE, except for MR, that is inherently sensitive to small variations. On YAGO3-10, quite interestingly, ConvKB does not seem to suffer from this issue, while CrossE shows a noticeable difference. CapsE shows the largest problems, with a behaviour akin to the extreme example described above.

		Any	Reflexive	Irreflexive	Symmetric	Anti Symmetric	Transitive
Tensor Decomposition	DistMult	0.41		0.41	0.21	0.36	0.12
	ComplEx	**0.50**		**0.50**	**0.42**	**0.45**	**0.53**
	ANALOGY	0.19		0.19	0.14	0.19	0.03
	SimplE	0.36		0.36	0.28	0.28	0.30
	HolE	0.42		0.42	0.24	0.36	0.50
	TuckER	0.47		0.47	0.36	0.40	**0.53**
Geometric	TransE	0.41		0.41	0.00	0.38	0.23
	STransE	0.03		0.03	0.01	0.06	0.03
	CrossE	0.33		0.33	0.17	0.28	0.13
	TorusE	0.27		0.27	0.29	0.23	0.27
	RotatE	0.41		0.41	0.38	0.35	0.47
Deep Learning	ConvE	0.40		0.40	0.31	0.37	0.40
	ConvKB	0.32		0.32	0.00	0.32	0.10
	ConvR	0.45		0.45	0.36	0.40	0.37
	CapsE	0.00		0.00	0.00	0.00	0.00
	RSN	0.43		0.43	0.27	0.38	0.30
	AnyBURL	0.46		0.46	0.40	0.42	0.17
	Test Facts Percentage	100%	0%	100%	3%	5%	0.3%

(a) YAGO3-10

Fig. 16. H@1 results for each LP model on YAGO3-10, and corresponding percentages of test facts, for various relation properties. The best results for each column are in bold and underlined.

		Degree of the 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	**0.75**	0.82	**0.87**	**0.88**	**0.88**	**0.92**
	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
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
	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	**0.84**	0.86	**0.88**	**0.88**	0.90
	Test Facts Percentage	39%	25.0%	14%	5%	3%	14%

(a) FB15k

		Degree of the 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	**0.26**	**0.24**	**0.28**	0.22	**0.34**	**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
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
	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	0.24	0.21	0.24	0.19	0.26	0.24
	Test Facts Percentage	74%	14.0%	4%	1%	1%	6%

(b) FB15k-237

Fig. 17. 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.

		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	1.000
	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 5. Results obtained with *average* or *ordinal* tie policy (**avg**) against results obtained with *min* tie 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).

We have run experiments on the architecture of these models in order to investigate which components are most responsible for these behaviours. We have found strong experimental evidence that *saturating activation functions* may be the one of the main causes of this issue.

Saturating activation functions yield the same result for inputs beyond (or below) a certain value. For instance, the ReLU function (Rectified Linear Unit), returns 0 for any input lesser or equal to 0. Intuitively, saturating activation functions make it more likely to set identical scores to different entities, thus causing the observed issue.

ConvKB and CapsE both use ReLUs between their layers. In order to verify our hypothesis, we have trained a version of CapsE substituting its ReLUs with Leaky ReLUs. The Leaky ReLU function is a non-saturating alternative to ReLU: it keeps a linear behaviour even for inputs lesser than 0, with slope α between 0 and 1. In our experiment we used $\alpha = 0.2$. We report in Table 5 also the result of this CapsE variation, that we dubbed CapsE "Leaky". As a matter of fact, for CapsE "Leaky" the differences between results obtained with *min* and *average* are much less prominent. In FB15k and FB15k-237, differences in H@1 and MRR either disappear or decrease of 2 or even 3 orders of magnitude, becoming barely observable. In WN18 and WN18RR, much like in the original CapsE, no differences are observable. In YAGO3-10 we still report a significant difference between *min* and *average* results, but it is much smaller than before.

CrossE does not employ explicitly saturating activation functions; nonetheless, after thoroughly investigating the model architecture, we have found that in this case too the issue is rooted in saturation. As shown in the scoring function in table 1, CrossE normalizes its scores by applying a sigmoid function in a final step. The sigmoid function is not saturating, but for values with very large modulus its slope is almost nil: therefore, due to low-level approximations, it behaves just like a saturating function. Therefore, we tested again CrossE just removing the sigmoid function in evaluation; since the sigmoid function is monotonous and growing, removing it does not affect entity ranks. In the obtained results all discrepancies between *min* and *average* policies disappear. As before, we report the results in Table 5.

For all the other models in our analysis we have not found significant differences among their results obtained with different policies.

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 Effect of the design choices

We discuss here comprehensive observations regarding the performances of models as well as their robustness across evaluation datasets and metrics. We report findings regarding trends investing entire families based on specific design choices, as well as unique feats displayed by individual models.

Among those included in our analysis, Tensor Decomposition models show the most solid results across datasets. In the implementations taken into account, most of these systems display uniform performances on all evaluation metrics across the datasets of our analysis (with the potential exceptions of ANALOGY and SimplE, that are seemingly more fluctuating). In particular, ComplEx with its N3 regularization displays amazing results on all metrics across all datasets, being the only embedding-based model consistently comparable to the baseline AnyBURL.

The Geometric family, on the other hand, shows slightly more unstable results. In the past years, research has devoted a considerable effort into translational models, ranging from TransE to its many successors with multi-embedding policies for handling many-to-one, one-to-many and many-to-many relations. These models show interesting results, but still suffer from some irregularities across metrics and datasets. For instance, models such as TransE and STransE seem to particularly struggle on the WN18RR dataset, especially when it comes to H@1 and MRR metrics. All in all, models relying solely on translations seem to have been outclassed by recent roto-translational ones. At this regard, RotatE shows remarkably consistent performances across all datasets, and it particularly shines when taking into account H@10.

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. ConvR and RSN display by far the best results in this family, achieving very similar, state-of-the-art performance in 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. On the other hand, models such as ConvKB and CapsE often achieve promising results on H@10 and MR metrics, whereas they seem to struggle with H@1 and MRR; furthermore, in some datasets they are clearly hindered by their issues with tie policies described in Section 5.4.1.

We stress that in the LP task the rule-based AnyBURL proves to be a remarkably well-performing model, as it consistently ranks among the best models across almost all datasets and metrics.

6.2 The importance of the graph structure

We have shown consistent experimental evidence that graph structural features have a large influence on what models manage to learn and predict.

We observe that in almost all models and datasets, predictions seem to be facilitated by the presence of source peers and hindered by the presence of target peers. As already mentioned, source peers work as examples that allow models

to characterize more effectively 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.

We also observe evidence suggesting that almost all models – even across those that only learn individual facts in training – seem able to leverage to some extent relational paths and patterns.

All in all, the toughest scenarios for LP models seem to take place when there are relatively more target peers than source peers, in conjunction with a low support offered by relational paths. In these cases, models usually tend to show quite unsatisfactory performances. We believe that these are the areas where future research has most room for improvement, and thus the next big challenges to address in the LP research field.

We also point out interesting differences in behaviours and correlations depending on the features of the employed dataset. In FB15k and WN18, which display strong test leakage, model performances show a prominent correlation with the support provided by shorter relational paths, with length 1 or 2. This is likely caused by such short paths including relations with inverse meaning or same meaning as the relation in the facts to predict. On the contrary, in their FB15k-237, WN18RR and YAGO3-10, which do not suffer from test leakage, models appear to rely also on longer relational paths (3 steps), as well as on the numbers of source/target peers.

We believe that this leaves room for intriguing observations. In presence of very short patterns providing overwhelming predictive evidence (e.g., the inverse relations that cause test leakage), models seem very prone to just focusing on them, disregarding other forms of reasoning: this can be seen as an unhealthy consequence of the test leakage problem. In more balanced scenarios, on the contrary, models seem to investigate to a certain extent longer dependencies, as well as to focus more on analogical reasoning supported by examples (such as source peers).

We also observe that applying LP models based on embeddings to infer relations with cardinality greater than 2 is still an open problem. As already mentioned in Section 4.3.4, the FreeBase KG represents hyperedges as reified CVT nodes. Hyperedges constitute the large majority of edges in FreeBase: as noted by Fatemi et al. [15] and Wen et al. [68], 61% of the FreeBase relations are beyond-binary, and the corresponding hyperedges involve more than 1/3rd of the FreeBase entities. The FB15k and FB15k-237 datasets have been built by performing S2C explosion on FreeBase subsamples; this has resulted in greatly altering both the graph structure and its semantics, with overall loss of information. We believe that, in order to assess the effects of this process, it would be fruitful to extract novel versions of FB15k and FB15k-237 in their original reified structure without applying S2C. We also note that models such as m-TransH [68] and the recent HypE [15] have tried to circumvent these issues by developing systems that can explicitly learn hyperedges. Despite them being technically usable on datasets with binary relations, of course their unique features emerge best when dealing with relations beyond binary.

6.3 The importance of tie 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 treated as almost negligible implementation details, and they are hardly ever even reported when presenting novel LP models. Nevertheless, we show that performances computed relying on different policies risk to be not directly comparable to one another, and might not even reflect the actual predictive effectiveness of models. Therefore we strongly advise researchers to use the same policy in the future; in our opinion, the "average" policy seems the most reasonable choice. We have also found 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.

7 RELATED WORKS

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

Analyses. Chandrahas *et al.* [51] study geometrical properties of the obtained embeddings in the latent space. They separate models into additive and multiplicative and measure the *Alignment To Mean* (ATM) and *conicity* of the learned vectors, showing that additive models tend to learn significantly sparser vectors than multiplicative ones. They then check how this reflects on the model peformances. Their observations are intriguing, especially for multiplicative models, where a high conicity (and thus a low vector spread) seems to correlate to better effectiveness.

Wang *et al.* [64] 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 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.

Akrami *et al.*[2] use the same intuition as Toutanova *et al.* [57] to carry out a slightly more structured analysis, as they use a wider variety of models to check the performance gap between FB15k and FB15K-237.

Kadlec *et al.* [26] demonstrate that a carefully tuned implementation of DistMult [71] 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.*[59] 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.

All the above mentioned analyses have a very different scope from ours. 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.* [45] 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.* [9] 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.

To this end, the surveys by Wang *et al.* [63] and by Nguyen [40] are the most relevant to our work, as they specifically focus on KG Embedding methods. In the work by Wang *et al.* [63], 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 [40] is similar, albeit more concise, and also includes current state-of-the-art methods such as RotatE [55].

Our work is fundamentally different from these surveys: while they only report results available in the original papers, we design 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 have presented the first extensive comparative analysis on LP models based on KG embedding.

We have surveyed 16 LP models representative of diverse techniques and architectures, and we have analyzed their efficiency and effectiveness on the 5 most popular datasets in literature.

We have introduced a set of structural properties characterizing the training data, and we have shown strong experimental evidence that they produce paramount effects on prediction performances. In doing so, we have investigated the circumstances that allow models to perform satisfactorily, while identifying the areas where research still has room for improvement.

We have thoroughly discussed the current evaluation practices, verifying that they can rely on different low-level policies producing incomparable and, in some cases, misleading results. We have analyzed the components that make models most sensitive to these policies, providing useful observations for future research.

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 and RSN for their for their amazing support and guidance on their models.

REFERENCES

[1] Accenture. Ampligraph. https://github.com/Accenture/AmpliGraph. [Online; accessed 10-October-2019].

[2] F. Akrami, L. Guo, W. Hu, and C. Li. Re-evaluating embedding-based knowledge graph completion methods. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1779–1782. ACM, 2018.

[3] B. An, B. Chen, X. Han, and L. Sun. Accurate text-enhanced knowledge graph representation learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 745–755, 2018.

[4] 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*, pages 722–735. Springer, 2007.

[5] baharefatemi. Simple. https://github.com/baharefatemi/SimplE. [Online; accessed 10-October-2019].

[6] I. Balazevic, C. Allen, and T. M. Hospedales. Tucker: Tensor factorization for knowledge graph completion. *CoRR*, abs/1901.09590, 2019. URL http://arxiv.org/abs/1901.09590.

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

[8] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795. NIPS, 2013.

[9] H. Cai, V. W. Zheng, and K. Chang. A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

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

[11] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI Conference on Artificial Intelligence*, 2018.

[12] 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 *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 601–610. ACM, 2014. ISBN 978-1-4503-2956-9. doi: 10.1145/2623330.2623623. URL http://doi.acm.org/10.1145/2623330.2623623.

[13] T. Ebisu and R. Ichise. Toruse: Knowledge graph embedding on a lie group. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1819–1826, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16227.

[14] facebookresearch. kbc. https://github.com/facebookresearch/kbc. [Online; accessed 10-October-2019].

[15] B. Fatemi, P. Taslakian, D. Vázquez, and D. Poole. Knowledge hypergraphs: Extending knowledge graphs beyond binary relations. *CoRR*, abs/1906.00137, 2019. URL http://arxiv.org/abs/1906.00137.

[16] L. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek. Fast rule mining in ontological knowledge bases with amie+. *The VLDB JournalfiThe International Journal on Very Large Data Bases*, 24(6):707–730, 2015.

[17] L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*, pages 413–422. ACM, 2013.

[18] G. A. Gesese, R. Biswas, and H. Sack. A comprehensive survey of knowledge graph embeddings with literals: Techniques and applications. In *Proceedings of the Workshop on Deep Learning for Knowledge Graphs (DL4KG2019) Co-located with the 16th Extended Semantic Web Conference 2019 (ESWC 2019), Portoroz, Slovenia, June 2, 2019*, pages 31–40, 2019. URL http://ceur-ws.org/Vol-2377/paper_4.pdf.

[19] L. Guo, Z. Sun, and W. Hu. Learning to exploit long-term relational dependencies in knowledge graphs. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2505–2514. PMLR, 2019.

[20] S. Guo, Q. Wang, L. Wang, B. Wang, and L. Guo. Knowledge graph embedding with iterative guidance from soft rules. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[21] F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

[22] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[23] 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, Jan. 2013. ISSN 0004-3702. doi: 10.1016/j.artint.2012.10.002.

[24] V.-P. HUYNH, V. MEDURI, S. ORTONA, P. PAPOTTI, and N. AHMADI. Mining expressive rules in knowledge graphs. 2019.

[25] X. Jiang, Q. Wang, and B. Wang. Adaptive convolution for multi-relational learning. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 978–987, 2019. URL https://www.aclweb.org/anthology/N19-1103/.

[26] R. Kadlec, O. Bajgar, and J. Kleindienst. Knowledge base completion: Baselines strike back. *arXiv preprint arXiv:1705.10744*, 2017.

[27] S. M. Kazemi and D. Poole. Simple embedding for link prediction in knowledge graphs. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 4289–4300, 2018. URL http://papers.nips.cc/paper/7682-simple-embedding-for-link-prediction-in-knowledge-graphs.

[28] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

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

[30] T. Lacroix, N. Usunier, and G. Obozinski. Canonical tensor decomposition for knowledge base completion. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 2869–2878. PMLR, 2018.

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

[32] N. Lao, T. Mitchell, and W. W. Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 529–539. Association for Computational Linguistics, 2011.

[33] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[34] Y. Lin, Z. Liu, H. Luan, M. Sun, S. Rao, and S. Liu. Modeling relation paths for representation learning of knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 705–714, 2015. URL https://www.aclweb.org/anthology/D15-1082/.

[35] H. Liu, Y. Wu, and Y. Yang. Analogical inference for multi-relational embeddings. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 2168–2178, 2017. URL http://proceedings.mlr.press/v70/liu17d.html.

[36] F. Mahdisoltani, J. Biega, and F. M. Suchanek. Yago3: A knowledge base from multilingual wikipedias. 2013.

[37] 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 *International Semantic Web Conference*, pages 3–20. Springer, 2018.

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

[39] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL http://arxiv.org/abs/1301.3781.

[40] D. Q. Nguyen. An overview of embedding models of entities and relationships for knowledge base completion. *CoRR*, abs/1703.08098, 2017. URL http://arxiv.org/abs/1703.08098.

[41] D. Q. Nguyen, K. Sirts, L. Qu, and M. Johnson. Stranse: a novel embedding model of entities and relationships in knowledge bases. *arXiv preprint arXiv:1606.08140*, 2016.

[42] 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 *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 327–333, 2018. URL https://www.aclweb.org/anthology/N18-2053/.

[43] 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 (1)*, pages 2180–2189. Association for Computational Linguistics, 2019.

[44] M. Nickel, V. Tresp, and H.-P. Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, volume 11, pages 809–816, 2011.

[45] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1): 11–33, 2015.

[46] M. Nickel, L. Rosasco, and T. A. Poggio. Holographic embeddings of knowledge graphs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 1955–1961, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12484.

[47] H. Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2017.

[48] R. Qian. Understand your world with bing. https://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing, 2013. Accessed: 2019-10-30.

[49] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017.

[50] H. Schütze, C. D. Manning, and P. Raghavan. Introduction to information retrieval. In *Proceedings of the international communication of association for computing machinery conference*, page 260, 2008.

[51] A. Sharma, P. Talukdar, et al. Towards understanding the geometry of knowledge graph embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 122–131, 2018.

[52] A. Singhal. Introducing the knowledge graph: things, not strings. http://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html, 2012. Accessed: 2019-10-30.

[53] T. Stocky and L. Rasmussen. Introducing graph search beta, 2014. URL https://newsroom.fb.com/news/2013/01/introducing-graph-search-beta/. Blogpost in Facebook Newsroom.

[54] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

[55] Z. Sun, Z. Deng, J. Nie, and J. Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. URL https://openreview.net/forum?id=HkgEQnRqYQ.

[56] TimDettmers. Conve. https://github.com/TimDettmers/ConvE. [Online; accessed 10-October-2019].

[57] K. Toutanova and D. Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, 2015.

[58] K. Toutanova, D. Chen, P. Pantel, H. Poon, P. Choudhury, and M. Gamon. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, 2015.

[59] H. N. Tran and A. Takasu. Analyzing knowledge graph embedding methods from a multi-embedding interaction perspective. In *Proceedings of the Workshops of the EDBT/ICDT 2019 Joint Conference, EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019*, 2019. URL http://ceur-ws.org/Vol-2322/dsi4-6.pdf.

[60] T. Trouillon and M. Nickel. Complex and holographic embeddings of knowledge graphs: A comparison. *CoRR*, abs/1707.01475, 2017.

[61] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2071–2080, 2016. URL http://proceedings.mlr.press/v48/trouillon16.html.

[62] D. Vrandečić and M. Krötzsch. Wikidata: a free collaborative knowledge base. 2014.

[63] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 29(12):2724–2743, 2017.

[64] Y. Wang, D. Ruffinelli, R. Gemulla, S. Broscheit, and C. Meilicke. On evaluating embedding models for knowledge base completion. In *Proceedings of the 4th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2019, Florence, Italy, August 2, 2019*, pages 104–112, 2019. URL https://www.aclweb.org/anthology/W19-4313/.

[65] Z. Wang and J.-Z. Li. Text-enhanced representation learning for knowledge graph. In *IJCAI*, pages 1293–1299, 2016.

[66] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, volume 14, pages 1112–1119. AAAI, 2014.

[67] web.informatik.uni mannheim.de. Anyburl. http://web.informatik.uni-mannheim.de/AnyBURL/. [Online; accessed 10-October-2019].

[68] J. Wen, J. Li, Y. Mao, S. Chen, and R. Zhang. On the representation and embedding of knowledge bases beyond binary relations. In *IJCAI*, pages 1300–1307. IJCAI/AAAI Press, 2016.

[69] R. West, E. Gabrilovich, K. Murphy, S. Sun, R. Gupta, and D. Lin. Knowledge base completion via search-based question answering. In *Proceedings of the 23rd international conference on World wide web*, pages 515–526. ACM, 2014.

[70] R. Xie, Z. Liu, H. Luan, and M. Sun. Image-embodied knowledge representation learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 3140–3146, 2017. doi: 10.24963/ijcai.2017/438. URL https://doi.org/10.24963/ijcai.2017/438.

[71] B. Yang, W. Yih, X. He, J. Gao, and L. Deng. Embedding entities and relations for learning and inference in knowledge bases. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6575.

[72] W. Zhang, B. Paudel, W. Zhang, A. Bernstein, and H. Chen. Interaction embeddings for prediction and explanation in knowledge graphs. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019*, pages 96–104, 2019. doi: 10.1145/3289600.3291014. URL https://doi.org/10.1145/3289600.3291014.

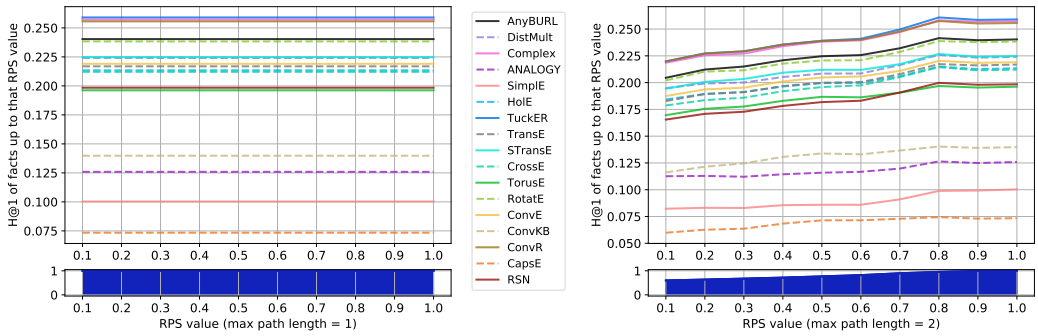
A HYPERPARAMETERS

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; Loss:self_adversarial; γ:1; LR:0.0005	BC:50; Ep:4000; D:200; Loss:nll; γ:1; LR:0.0005;	BC:50; Ep:4000; D:300; Loss:multiclass_nll; LR:0.00005; Reg:L3 (λ:0.0001);	BC:100; Ep:4000; D:350; Loss:multiclass_nll; LR:0.0001; Reg:L2 (λ:0.0001);	BC:100; Ep:4000; D:350; Loss:multiclass_nll; LR:5e-05; Reg:L3 (λ:0.0001);	
ComplEx	B: 100; Ep: 200; D:2000; LR: 1e-2; Reg:N3; Opt:Adagrad	B: 1000; Ep: 20; D:2000; LR: 0.1; Reg:N3; Opt:Adagrad	B: 100; Ep: 100; D:2000; LR: 0.1; Reg:N3; Opt:Adagrad	B: 100; Ep: 100; D:2000; LR: 0.1; Reg:N3; Opt:Adagrad	B: 1000; Ep: 100; D:2000; LR: 0.1; Reg:N3; Opt:Adagrad	
ANALOGY	B:1; Ep:500; D:200; LR:0.1; Decay:1e-3; Opt:Adagrad; N:6;	B:1; Ep:500; D:200; LR:0.1; Decay:1e-2; Opt:Adagrad; N:3;	B:1; Ep:500; D:150; LR:0.01; Decay:1e-2; Opt:Adagrad; N:6;	B:1; Ep:500; D:200; LR:0.1; Decay:1e-3; Opt:Adagrad; N:6;	B:1; Ep:500; D:100; LR:0.1; Decay:1e-3; Opt:Adagrad; N:6;	36
SimplE	B:4832; Ep:1000; D:200; LR:0.05; N:10; Reg:L2 (λ:0.1);	B:1415; Ep:1000; D:200; LR:0.1; N:1; Reg:L2 (λ:0.03);	B:4832; Ep:500; D:200; LR:0.1; N:3; Reg:L2 (λ:0.1);	B:1415; Ep:1000; D:150; LR:0.1; N:10; Reg:L2 (λ:0.03);	B:2048; Ep:1000; D:200; LR:0.2; N:10; Reg:L2 (λ:0.03)	1176
HolE	BC:50; Ep:4000; D:200; Loss:self_adversarial; γ:1; LR:0.0005;	BC:50; Ep:4000; D:200; Loss:self_adversarial; γ:1; LR:0.0005;	BC:64; Ep:4000; D:350; Loss:multiclass_nll; LR:0.0001; Reg: L2 (λ:0.0001);	BC:50; Ep:4000; D:200; Loss:self_adversarial; γ:1; LR:0.0005;	BC:100; Ep:4000; D:350; Loss:self_adversarial; γ:0.5; LR:0.0001;	
TuckER	B:128; Ep:500; D:200; LR:0.003; ε:0; Decay Rate:0.99 Drop:{in:0.2; h_1:0.2; h_2:0.3}	B:128; Ep:500; D:200; LR:0.0005; ε:0.1; Decay Rate:0.995 Drop:{in:0.3; h_1:0.4; h_2:0.5}	B:128; Ep:500; D_e:200; D_r:30; LR:0.005; ε:0.1; Decay Rate:1.0; Drop:{in:0.2; h_1:0.1; h_2:0.2}	B:128; Ep:500; D_e:200; D_r:30; LR:0.01; ε:0.1; Decay Rate: 1.0; Drop:{in:0.2; h_1:0.2; h_2:0.3}	B:128; Ep:500; D_e:200; D_r:30; LR: 0.001; ε:0; Decay Rate:1.0; Drop:{in:0.2; h_1:0.2; h_2:0.2}	3750
TransE	BC:100; Ep:4000; D:150; Loss:multiclass_nll; Norm:L1; LR: 5e-05; Reg:L3 (λ: 0.0001);	BC:100; Ep:4000; D:150; Loss:multiclass_nll; Norm: L1; LR: 5e-05; Reg:L3 (λ: 0.0001);	BC:64; Ep:4000; D:400; Loss:multiclass_nll; Norm: L1; LR:0.0001; Reg:L2 (λ: 0.0001);	BC:150; Ep:4000; D:350; Loss:multiclass_nll; Norm:L1; LR: 0.0001; Reg:L2 (λ: 0.0001);	BC:100; Ep:4000; D:350; Loss:multiclass_nll; Norm:L1; LR:0.0001; Reg:L2 (λ: 0.0001);	
STransE	B:1; Ep:2000; D=100; LR:0.0001; Opt:SGD; Norm:L1; γ:1;	B:1; Ep:2000; D=50; LR:0.0005; Opt:SGD; Norm:L1; γ:5;	B:1; Ep:2000; D=100; LR:0.001; Opt:SGD; Norm: L1; γ:5;	B:1; Ep:2000; D=50; LR:0.0005; Opt:SGD; Norm: L1; γ:5;	B:1; Ep:500; D=350; LR:0.01; Opt:SGD; Norm: L2; γ:5;	60
CrossE	B:4000; Ep:500; D:300 LR:0.01; Loss Weight:1e-6	B:2048; Ep:500; D:100 LR 0.01; Loss Weight:1e-4	B:4000; Ep:500; D:100 LR:0.01; Loss Weight:1e-5	B:4000; Ep:120; D:100 LR:0.01; Loss Weight:1e-4	B:1024*; Ep:150; D:300 LR:0.01; Loss Weight:1e-6	12
TorusE	BC:100; Ep:1000; D:10000; LR:0.0005; Norm:EL2; Opt:SGD; γ:500;	BC:100; Ep:1000; D:10000; LR:0.0005; Norm:L1; Opt:SGD; γ:2000;	BC:100; Ep:1000; D:10000; LR:0.001; Norm:L1; Opt:SGD; γ:2000;	BC:100; Ep:1000; D:10000; LR:0.0002; Norm:L1; Opt:SGD; γ:3000;	BC:100; Ep:250; D:10000; LR:0.0002; Norm:EL2; Opt:SGD; γ:100;	75
RotatE	B:1024; Steps:150000; D=1000; LR:0.0001; N:256; γ:24; AdvTemp:1.0;	B:512; Steps:80000; D=500; LR:0.0001; N:1024; γ:12; AdvTemp:0.5;	B:1024; Steps:100000; D=1000; LR:0.00005; N:256; γ:9; AdvTemp:1.0;	B:512; Steps:80000; D=500; LR:0.00005; N:1024; γ:6; AdvTemp:0.5;	B:1024; Steps:100000; D=500; LR:0.0002; N:400; γ:24; AdvTemp:1.0;	
ConvE	B:128; Ep:1000; D:200; LR:0.003; Kernel:3x3; K:256; Decay Rate:0.995; ε:0.1; Drop:{in:0.2;h:0.3;feat:0.2}	B:128; Ep:1000; D:200; LR:0.003; Decay Rate:0.995; ε:0.1; Kernel:3x3; K:256; Drop:{in:0.2;h:0.3;feat:0.2}	B:128; Ep:1000; D:200; LR:0.003; Decay Rate:0.995; ε:0.1; Kernel:3x3; K:256; Drop:{in:0.2;h:0.3;feat:0.2}	B:128; Ep:1000; D:200; LR:0.003; Decay Rate:0.995; ε:0.1; Kernel:3x3; K:256; Drop:{in:0.2;h:0.3;feat:0.2}	B:128; Ep:1000; D:200; LR:0.003; Decay Rate:0.995; ε:0.1; Kernel:3x3; K:256; Drop:{in:0.2;h:0.3;feat:0.2}	
ConvKB	B:128; Ep:200; D=100; LR:0.0005; K:200; Reg:L2 (λ:0.001); ConstantInit: Y;	B:256; Ep:200; D=50; LR:5e-05; K:500; Reg:L2 (λ:0.001); ConstantInit: N;	B:256; Ep:200; D=100; LR:0. 000005; K:50; Reg:L2 (λ:0.001); ConstantInit: Y;	B:256; Ep:200; D=50; LR:0.0001; K:500; Reg:L2 (λ:0.001); ConstantInit: N;	B:256; Ep:200; D=350; LR:1e-05; K:500; Reg:L2 (λ:0.001); ConstantInit: N;	50
ConvR	B:128; Ep:1000; D_e:200; LR:0.001; ε: 0.1; Drop:{in:0.1;h:0.4;feat:0.2} Kernel:3x3; K=100;	B:128; Ep:1000; D_e:200; LR:0.001; ε: 0.1; Drop:{in:0.4;h:0.3;feat:0.3} Kernel:3x3; K=100;	B:128; Ep:1000; D_e:100; LR:0.001; ε: 0.1; Drop:{in:0.3;h:0.2;feat:0.3} Kernel:5x5; K=200;	B:128; Ep:1000; D_e:200; LR:0.001; ε: 0.1; Drop:{in:0.2;h:0.2;feat:0.5} Kernel:3x3; K=200;	B:128; Ep:1000; D_e:200; LR:0.001; ε: 0.1; Drop:{in:0.1;h:0.2;feat:0.2} Kernel:3x3; K=100;	3000
CapsE	B:128; Ep:60; D:100; K:500; LR:0.0005; ConstantInit:Y	B:128; Ep:60; D:50; K:50; LR:0.00005; ConstantInit:N	B:128; Ep:30; D:100; K:50; LR:0.0005; ConstantInit:Y	B:128; Ep:60; D:100; K:400; LR:0.0005; ConstantInit:N	B:128; Ep:30; D:350; K:200; LR:0.0005; ConstantInit:Y	12
RSN	B: 2048; Ep: 200; D= 256; LR:0.0001; Drop:{h:0.5};	B: 2048; Ep: 200; D= 256; LR:0.00001; Drop:{h:0.5};	B: 2048; Ep: 200; D= 256; LR:0.0001; Drop:{h:0.5};	B: 2048; Ep: 200; D= 256; LR:0.00005; Drop:{h:0.5};	B: 2048; Ep: 200; D= 256; LR:0.0001; Drop:{h:0.5};	2
AnyBURL	Training time = 100s	Training time = 100s	Training time = 1000s	Training time = 1000s	Training time = 1000s	

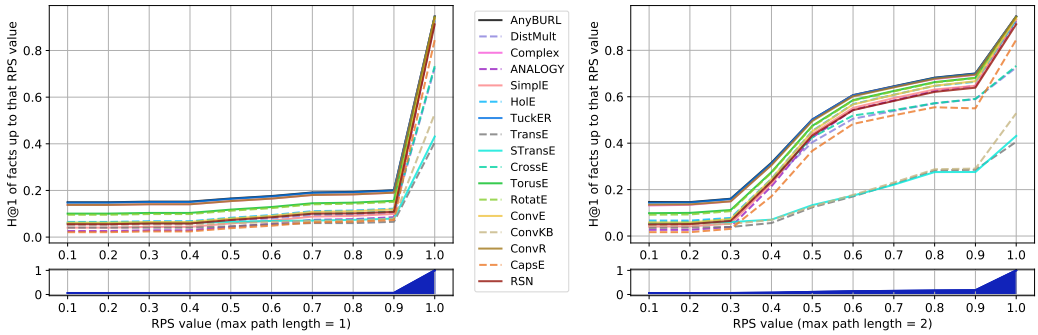
Table 6. Hyperparameters used to train all the models in our work. B: batch size; alternatively BC: batch count. Ep: training epochs; alternatively $Steps$: training steps. D: embedding dimension; alternatively, D_e and D_r: entity and relation embedding dimension. LR: learning rate. γ: regularization 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.

A RPS WITH PATHS OF MAXIMUM LENGTHS 1 AND 2

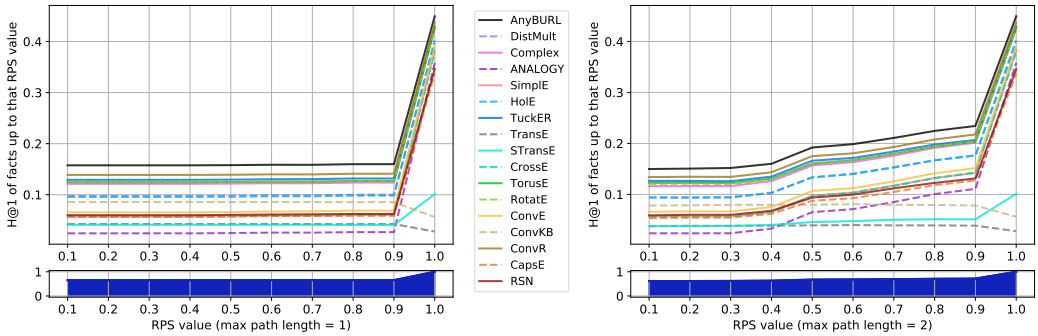


(a) FB15k-237

Fig. 18. H@1 results for each LP model on FB15k-237 varying the RPS of the test facts, computing RPS with paths up to length 1 and up to length 2.

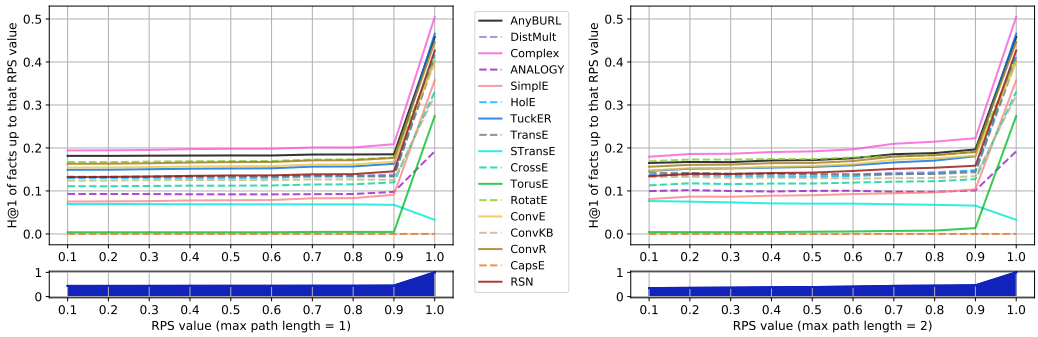


(a) WN18



(b) WN18RR

Fig. 19. 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. 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.