Searching for Embeddings in a Haystack: Link Prediction on Knowledge Graphs with Subgraph Pruning

Unmesh Joshi and Jacopo Urbani
{u.n.joshi,j.urbani}@vu.nl
Vrije Universiteit Amsterdam
The Netherlands

ABSTRACT

Embedding-based models of Knowledge Graphs (KGs) can be used to predict the existence of missing links by ranking the entities according to some likelihood scores. An exhaustive computation of all likelihood scores is very expensive if the KG is large. To counter this problem, we propose a technique to reduce the search space by identifying smaller subsets of promising entities. Our technique first creates embeddings of subgraphs using the embeddings from the model. Then, it ranks the subgraphs with some proposed ranking functions and considers only the entities in the top k subgraphs. Our experiments show that our technique is able to reduce the search space significantly while maintaining a good recall.

ACM Reference Format:
Unmesh Joshi and Jacopo Urbani. 2020. Searching for Embeddings in a Haystack: Link Prediction on Knowledge Graphs with Subgraph Pruning. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3366423.3380043

1 INTRODUCTION

Knowledge Graphs (KGs) are a popular representation model to publish factual knowledge on the Web. KGs are crucial assets for enhancing various tasks like question answering [7], ontology-based data access [5], task-oriented dialogs [19], data integration [16], or named entity recognition [24]. Although the largest public KGs contain billions of statements (e.g., Wikidata [26], DBpedia [18]), they are still far from being complete.

The problem of completing the KGs is addressed by numerous techniques which range from rule mining [11], extraction from unstructured sources [15], or ontological reasoning [1]. In this paper we consider embedding-based models [20], i.e., models where the entities and relations in the KG are "embedded" into high-dimensional numerical vectors (called *embeddings*) and potential new links are identified by computing numerical likelihood scores.

Multiple studies have shown that these techniques return good results for KG completion (see surveys at [4, 20]) but their application to large KGs is problematic for two reasons. First, large KGs can contain millions of entities and this leads to models with a huge number of parameters. The second problem concerns the discovery of links that complete partially-bounded statements like $(?, bornIn, UK)$, which can be seen as a query answering problem. To solve this problem, embeddings are used to construct an numerical representation of the query, which is then combined with the embeddings of *all* entities to assign likelihood scores to each of them. Then, only the entities with the best scores are considered as potential answers. The problem is that if the KG has many entities, then computing the likelihood score for every entity is too expensive.

In this paper, we propose a technique that addresses these two problems. The main idea is to restrict the ranking only to the subset of the most promising entities. This subset is computed in two stages: First, we compute the likelihood scores considering embeddings that represent *sets* of entities contained in star-shaped subgraphs. We refer to these embeddings as *subgraph embeddings*. Then, we compute the likelihood scores considering only the entities in the top-ranked subgraphs. Since typically there are far fewer subgraphs than the entities, computing the first ranking is fast and does not require loading the entire model. Therefore, our method introduces savings both in terms of runtime and resource utilization.

Our technique can be applied with several existing embedding models. We considered TransE [3], HolE [21], DistMult [30], and ConvE [6], which are currently among the most popular models. Our discussion focuses on three key aspects of our technique. *Firstly*, we study the benefits with two representations of the subgraph embeddings. The first representation views the subgraph embedding as the average of the embeddings of its entities. The second one constructs Gaussian embeddings, i.e., constructs a Gaussian probability distribution for each subgraph. *Secondly*, we introduce two different scoring functions to rank the best subgraphs. The first function reuses the likelihood score of the embedding model while the second one applies KL-divergence [17] between the distribution of the known answers of the query and of the subgraph embeddings. *Thirdly*, we describe how we can determine automatically the number of top k subgraphs using evidence in previous query executions.

Our experiments show that our method can reduce the search space to a fraction of all entities. In many cases this reduction does not compromise the recall, i.e., known correct answers are not ignored. In the best cases, the reduction can be up to one order of magnitude while preserving a recall $\geq$ 50%. This makes our method a valid alternative to perform an exhaustive ranking with all entities.

All the code and models are available at the link https://github.com/unmeshvrije/scikit-kge.

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380043

Model	Score function $\psi(\mathbf{h}, \mathbf{r}, \mathbf{t})$
TransE [3]	$\| \mathbf{h} + \mathbf{r} - \mathbf{t} \|_L$
DistMult [30]	$\mathbf{h}^\top \mathbf{W}_r \mathbf{t}$
HolE [21]	$\sigma(\mathbf{r}^\top (\mathbf{h} \star \mathbf{t}))$
ConvE [6]	$\sigma(nn(\mathbf{h}, \mathbf{r}) \cdot \mathbf{t})$

Table 1: Score functions with $\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{R}^d$, $\mathbf{W}_r$ is a diagonal matrix $\in \mathbb{R}^{d \times d}$, $\star$ is circular correlation, nn is 2D convolution. L is the L_1 or L_2 norms

2 PRELIMINARIES

We view a KG as a directed labeled graph $\mathcal{K} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ is a set of edges, and $\mathcal{R}$ is a set of named relations. We denote each edge in $\mathcal{E}$ as a triple (h, r, t) where h (head) is the outgoing node, t (tail) is the incoming one, and $r \in \mathcal{R}$ (relation name) is the label of the edge. Intuitively, nodes represent entities while edges indicate semantic relations between them, e.g., $(London, capitalOf, UK)$. The set of triples is divided into three sets: training ($\mathcal{E}_{train}$), validation ($\mathcal{E}_{val}$) and test ($\mathcal{E}_{test}$).

An embedding is a vector in $\mathbb{R}^d$ with $d > 0$. We use boldface fonts to denote them, i.e., we write $\mathbf{e}$ and $\mathbf{r}$ to refer to the vectors associated with the entity e and the relation r respectively. A *model* is a set of embeddings. To learn a model, techniques like TransE, ConvE, etc. define a *score function* $\psi(\mathbf{h}, \mathbf{r}, \mathbf{t})$ to compute the likelihood that (h, r, t) is true and then use it to specify a loss function (e.g., pairwise hinge loss [3], binary cross entropy [6]) that should be minimized during training. Table 1 shows the score functions of the models that we considered in this paper.

Once the embeddings are trained, they can be used for tasks like link prediction. The goal of link prediction is to predict the head or the tail entities given the relation and the other entity. For instance, if r and t are given, then the goal is to predict the correct h, which can also be seen as answering a query of the form $(?, r, t)$. This can be done by computing the score function for r and t and any other entity e and use it as a likelihood score that e is an answer.

Computing the likelihood score for every entity can be computationally expensive. The goal of our technique is to reduce this cost with an approximate ranking. This problem can also be seen as a k-nearest neighbour (kNN) search problem. Given an embedding as input, the goal of a kNN technique is to find the top k embeddings which are closest according to a distance function. In our evaluation, we will compare the performance of our method against IVFADC [13], one state-of-the-art technique of this kind.

3 APPROACH

Overview. We propose a two-stage approach to speed up link prediction. In the first stage, we create embeddings of the subgraphs. In the second stage, we rank the subgraphs based on the likelihood scores between the embedding of the query and *subgraph embeddings*. Optionally, a third stage can be applied to compute a suitable value of k to select the top k subgraphs.

Example 3.1. Consider the query $Q = (?, bornIn, UK)$ in Figure 1. It is likely that answers for this query are persons that are members of the UK parliament, or persons who work in London. These are entities contained in star-shaped subgraphs rooted in the entities *UK_Parliament* and *London* respectively. It is much less likely that an answer can be found in a subgraph that contains vehicles. First, our approach uses the embeddings of the people who are members

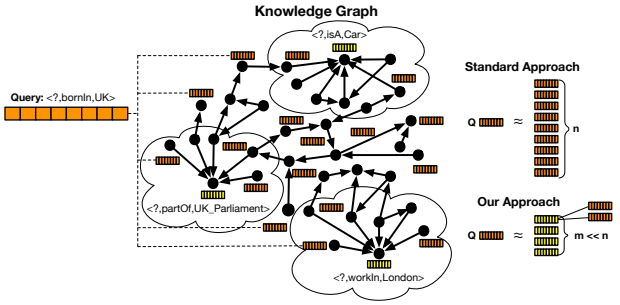


Figure 1: Overview of our approach. V is the number of entities in $\mathcal{K}$, U is the number of subgraphs

of the UK Parliament to construct a subgraph embedding for the subgraph rooted at *UK Parliament*. Similarly, it creates subgraph embeddings for other subgraphs. Then, it ranks the likelihood scores computed using these embeddings and the query to identify the top k subgraphs that most likely contain answers for the query.

Creation of Subgraph Embeddings (First stage). We use subgraphs to identify groups of similar entities. In principle, our method can be applied with any type of subgraph, provided that we can construct an embedding for it. In this paper, we consider only star-shaped subgraphs which are formed by sets of entities that share a common neighbor with a fixed relation. The reason for this restriction is that the similarity criterion in such subgraphs is naturally defined by the shared neighbor. For instance, all entities which share a connection to the entity *London* with the relation *worksIn* are similar to each other precisely because they all work in London. We distinguish two types of subgraphs: The ones which have an outgoing edge to the common neighbour and the ones that have an incoming one.

Definition 3.2. The *outgoing subgraph* rooted at entity e and relation r is defined as the set of entities $\{t \mid (e, r, t) \in \mathcal{E}_{train}\}$, i.e., the set of entities which are connected to e with an incoming arc with label r. Likewise, an *incoming subgraph* rooted at e and r is the set $\{t \mid (t, r, e) \in \mathcal{E}_{train}\}$

Since our goal is to reduce the search space for potential answers, we ignore subgraphs that are too small to lead to a significant reduction. To this end, we introduce a threshold value τ and ignore all subgraphs with less than τ entities.

In order to compute the subgraph embeddings, we rely on an external model that computes the embeddings of the entities. We refer to this model as the *embedding model*. In this paper, we considered four methods (TransE, HolE, DistMult, ConvE) but we believe that our approach can also be used with other models. Note that the embedding model provides not only the entity embeddings but also the score functions to determine the similarity between them.

Given the embeddings provided by the embedding model, we consider two approaches for constructing the subgraph embeddings. The first approach computes the subgraph embedding as the average of embeddings of the members of the subgraph. That is, given a subgraph S, the corresponding embedding $\mathbf{S}_\mu$ is computed

as

$$\mathbf{S}_\mu = \frac{1}{|S|}\sum_{e\in S}\mathbf{e} \tag{1}$$

With this method, the subgraph is embedded as a single "point" in the high-dimensional space, but the position is sensible to outliers as it is computed with the average. To counter this problem, the second approach represents the subgraphs using Gaussian embeddings [10, 25] so that they are no longer represented by single points but rather as areas where the subgraphs are more or less likely to be located. The idea behind Gaussian embeddings is to represent each symbol as a multi-dimensional Gaussian probability distribution. The probability distribution determines the likelihood that the symbol is positioned at certain coordinates. This likelihood is high around the average but diminishes as we move away with the rate defined in the normal distribution.

With this method, every subgraph is defined by an average and variance embeddings, i.e., a tuple $\langle \mathbf{S}_\mu, \mathbf{S}_\sigma\rangle$. The average embedding $\mathbf{S}_\mu$ is computed with Equation 1. Each element $\mathbf{S}_\sigma[i]$ of the variance embeddings $\mathbf{S}_\sigma$ is computed as

$$\mathbf{S}_\sigma[i] = \sum_{j=1}^{|S|}\frac{(\mathbf{e_j}[i]-\mathbf{S}_\mu[i])^2}{|S|-1} \tag{2}$$

where $1 \le i \le d$. Note that in (2) we divide by $|S|-1$ instead of $|S|$ following Bessel's correction [28].

Ranking Subgraph Embeddings (Second stage). After we computed the embeddings of all subgraphs with more than τ entities, our system is ready to perform link prediction. Let us consider an input query q of the form $(?, r, e)$ where $r \in \mathcal{R}, e \in \mathcal{V}$, and $\mathcal{U}$ is the set of all subgraphs that we extracted from $\mathcal{K}$. Moreover, let us assume that there exists an ideal KG $\mathcal{K}' \supseteq \mathcal{K}$ which contains all true facts over $\mathcal{V}$ and $\mathcal{R}$, and let $\mathcal{A} = \{e' \mid (e', r, e) \in \mathcal{K}'\}$ be the set of all admissible answers for q (the case for $q = (e, r, ?)$ is analogous).

Our goal is to rank the entities in a list $\rho = \langle e_1, \ldots, e_n\rangle$ that contains exactly the entities in $\mathcal{A}$. If we rank all the entity embeddings, then we obtain an approximate ρ' where $n = |\mathcal{V}|$ and possibly not all the first entities in ρ' are in $\mathcal{A}$. Since we are not interested in the answers not in $\mathcal{A}$, we would like to rank as few entities as possible without excluding the ones in $\mathcal{A}$. In other words, our objective is to compute ρ such that: 1) n is as close as possible to $|\mathcal{A}|$ and 2) ρ contains as many entities in $\mathcal{A}$ as possible.

We use the subgraph embeddings to achieve our goal. First, we create an embedding $\mathbf{q}$ of query q as it is defined by the embedding model. Then, we compute the likelihood scores between $\mathbf{q}$ and the embeddings of the subgraphs. The computation of the likelihood scores depends on the representation used for the subgraphs. We use the functions score_μ and score_{kl}, defined below, when the subgraphs are computed as average and Gaussian embeddings respectively.

Score score_μ **.** If we use the subgraph embeddings computed as the average of the subgraph's members (first approach), then we use the score function ψ provided by the embedding model to compute the likelihood score between the subgraph embedding and the query, effectively treating the subgraph embeddings like any other entities.

Score score_{kl} **.** The score function ψ cannot be applied as-is on Gaussian embeddings. In this case, we proceed as follows. First, we consider up to t (default value is 50) known answers of q from $\mathcal{E}_{train}$ to construct a Gaussian embedding of the query. Then, we use the Kullback–Leibler (KL) divergence [17] as scoring function. This function measures to what extent the two distributions differ from each other (a returned value of 0 means that the two distributions are equivalent). Therefore, it is a suitable measure to quantify the likelihood score between the query and the subgraph.

More formally, score_{kl} for query q and subgraph S is defined as

$$\text{score}_{kl}(\langle \mathbf{q}_\mu, \mathbf{q}_\sigma\rangle, \langle \mathbf{S}_\mu, \mathbf{S}_\sigma\rangle) = \sum_{i=1}^{d}\frac{(\boldsymbol{q_\mu}[i]-\mathbf{S}_\mu[i])^2 + (\boldsymbol{q_\sigma}[i]^2)}{2(\mathbf{S}_\sigma[i])^2} + \ln\frac{\sqrt{\mathbf{S}_\sigma[i]}}{\boldsymbol{q_\sigma[i]}} - \frac{1}{2} \tag{3}$$

where $\langle \mathbf{q}_\mu, \mathbf{q}_\sigma\rangle$ and $\langle \mathbf{S}_\mu, \mathbf{S}_\sigma\rangle$ are the Gaussian embeddings of q and S respectively.

Example 3.3. If we use TransE as embedding model and average subgraph embeddings, then the embedding for $q = (?, r, e)$ is $\mathbf{q} = \mathbf{e} - \mathbf{r}$. Then, for every subgraph $S \in \mathcal{U}$, the likelihood score is computed as:

$$\text{score}_\mu(\mathbf{q}, \mathbf{S}_\mu) = \| \mathbf{q} - \mathbf{S}_\mu \|_L \tag{4}$$

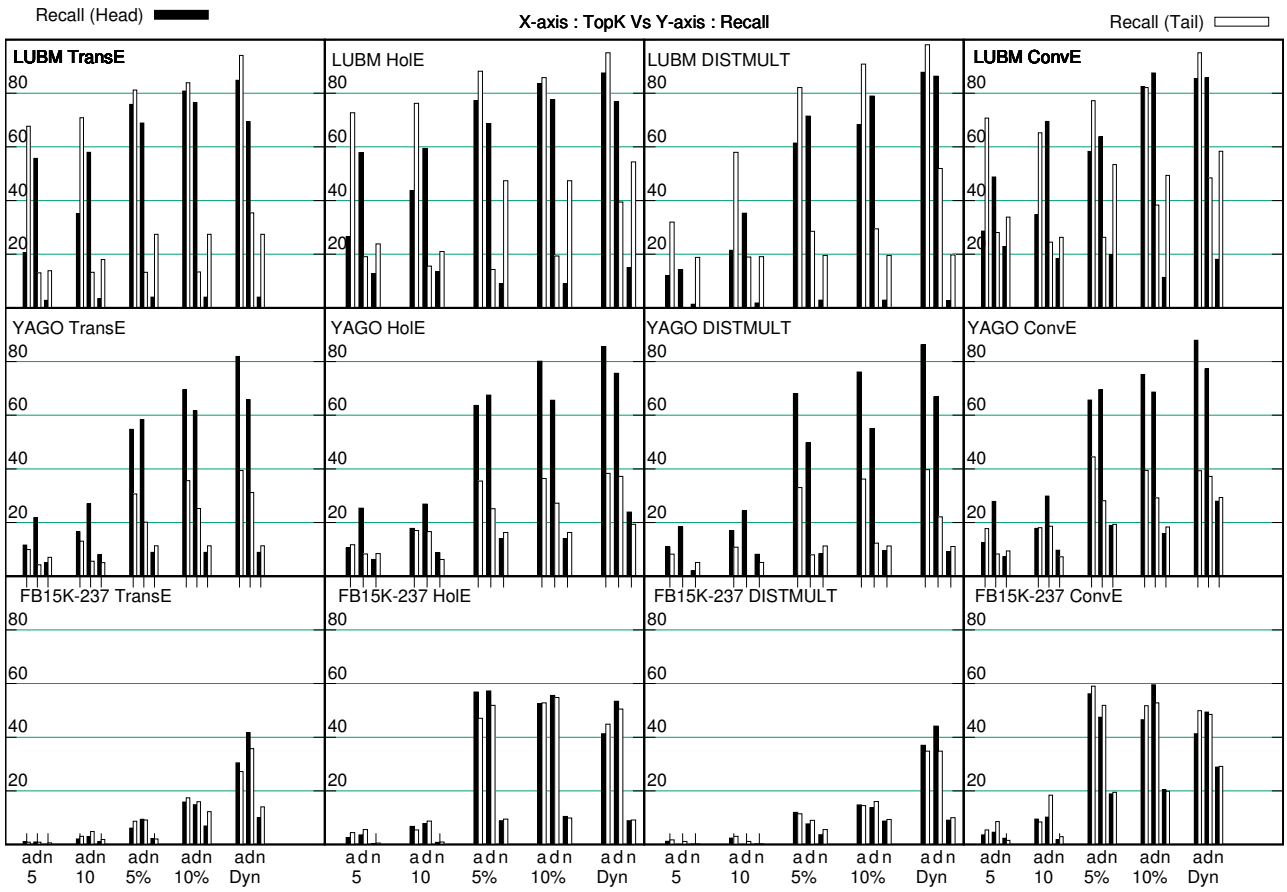
The ranking depends on a parameter k which determines the number of the top subgraphs that should be considered. Higher values of k will lead to higher recalls since they increase the chance that more entities are included. The downside is that the runtime will also increase. Lower values of k will have the opposite effect.

Computing k **Dynamically (Optional third stage).** Finding an optimal value for k might not be trivial. We propose the following procedure to dynamically compute such a value. For a given q, we select all known answers to q from $\mathcal{E}_{train} \cup \mathcal{E}_{val}$. Then, we compute the position of the first subgraphs that contain known answers and take the maximum value (up to a maximum threshold value of 50% of $|\mathcal{U}|$). If there is no subgraph that contains the answer, or there are no answers for q in $\mathcal{E}_{train} \cup \mathcal{E}_{val}$, then we set $k = \max(10, 0.1 \times |\mathcal{U}|)$, i.e., we set k equal to 10% of the number of subgraphs with a minimum value of 10 if there are fewer than 100 subgraphs.

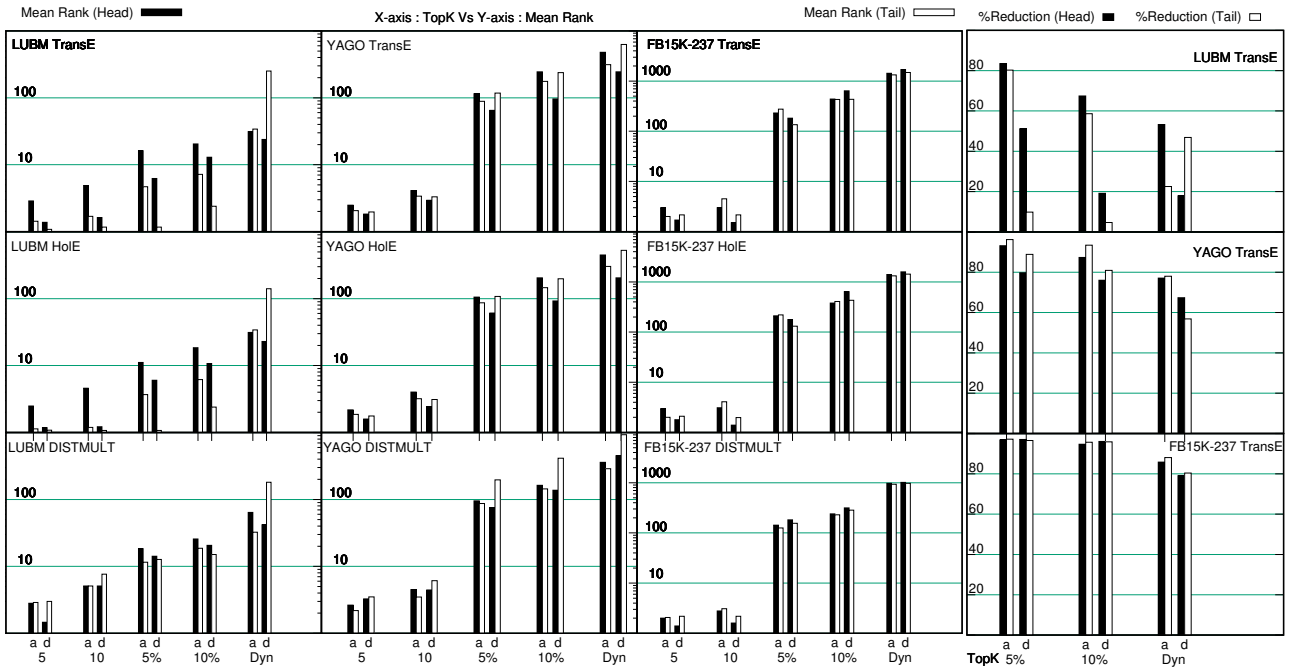
4 EVALUATION

Datasets. We considered LUBM [9], a well-known benchmark tool and three popular real-world KGs: a subset of YAGO [12], a subset of Freebase, and Wikidata [26]. We chose LUBM and YAGO because they are sparse KGs, thus are more challenging inputs [22], while Wikidata is a popular large KG. We used the subset FB15K-237 of Freebase, which is used in several related publications [6]. We used the same subsets of LUBM, YAGO as [22]. We used the April 2019 version of Wikidata (truthy statements) from the official repository. Figure 3d contains statistics about each dataset.

Model training. We split each KG in the train / validation / test (98%/1%/1%) subsets, as usual. To create the models, we considered TransE (model parameters: Adagrad [8], embedding size (d)



(a) Recalls (higher the better)



(b) Mean Ranks (lower the better)

(c) % Reductions (higher the better)

Figure 2: Recall/Mean Rank/%Reduction on LUBM, YAGO, FB15K237 with "a": score_{μ}, **"d":** score_{kl}, **"n":** score_{n}

50, learning rate (lr) 0.1, margin (γ) 1, batch size 1000), Distmult (Adagrad, d 50, lr 0.1, γ 1), HolE (Adagrad, d 50, lr 0.1, γ 0.2) and ConvE (embedding dropout 0.3, feature map dropout 0.2, projection layer dropout 0.3, d 50, batch size 128, lr 0.001 and label smoothing 0.1). These methods were used to learn the embeddings of all KGs except Wikidata, which was used only with TransE.

All models were trained for up to 500 epochs. To train LUBM, YAGO, and FB15K-237, we used a machine equipped with 64GB of RAM and two 8-core CPU 2.4GHz. The training terminated in a few hours in the longest case. The model of Wikidata is significantly larger than the other three. Therefore, we used another machine with 1TB of RAM and four 12-core Intel Xeon E5 CPUs. In this case, training the model with TransE and 32 HOGWILD! [23] threads took approximately 13 days.

We created subgraphs with different values of τ. The runtime for the creation with LUBM, FB15K-237, and YAGO is within a few seconds. For Wikidata, it took about five hours. Table 3d shows the number of created subgraphs.

To compare our method against approximate kNN techniques, we considered the state-of-the-art implementation of IVFADC [13] provided in the library FAISS by Facebook [14]. This technique depends on two important hyperparameters: The number of centroids (c) and the number of bytes (b) per code. Recommended settings are that the number of centroids is between $4\sqrt{N}$ and $16\sqrt{N}$ where N is the number of entities while the bytes per code should be between 5 and 25. We performed a grid search within these ranges and obtained the best results with $c = 4\sqrt{N}$ and $b = 5$.

Subgraph-based predictions. We performed a number of experiments to evaluate the predictions with our method on the testsets ($\mathcal{E}_{test}$). For each triple, we performed a *head* (H) prediction (i.e., we try to predict answers for queries of the form $(?, x, y)$) and a *tail* (T) prediction (i.e., queries of the form $(x, y, ?)$). In these experiments, we included all subgraphs created with $\tau \geq 10$.

We considered three metrics: *Recall*, *%Reduction*, and *Mean Rank*. With the recall, we measure how many times the test answer (head or tail) was among the selected subgraphs. This metric is important because it indicates how many times our method did not exclude true answers. We define %Reduction as $100 - |A|/|\mathcal{V}| * 100$ where A is the union of all entities in the selected subgraphs and $\mathcal{V}$ is the set of all entities. This metric shows how effective our method is in reducing the search space because a higher value indicates that our method selected a much smaller fraction of all entities as potential answers. The last metric measures the position of the first subgraph where the answer was found. The ideal case would occur when both recall and %reduction are maximum and the mean rank is minimum, but higher values of k will favor the recall instead of the %reduction.

We use the recall to compare the performance of our method against IVFADC. To this end, for every query we configure IVFADC to retrieve the top x similar entities where x is the number of entities contained in the top k subgraphs. For instance, suppose that our method is called to select the top three subgraphs, and for a given query it selected the subgraphs g_1, g_2, g_3 which contain x_1, x_2, x_3 sets of entities respectively. Then, IVFADC is configured to retrieve the top $x = |x_1 \cup x_2 \cup x_3|$ entities. In this way, we compare fairly because we check the recall with the same number of answers.

Figure 2a shows the recall with all models on LUBM, YAGO, and FB15k-237. The recall is shown for head and tail predictions using average and Gaussian embeddings (score_μ and score_{kl} resp.), and IVFADC (score_n). Our method used top 5, 10, 5%, 10% subgraphs or the dynamically computed *top k* ("Dyn") subgraphs.

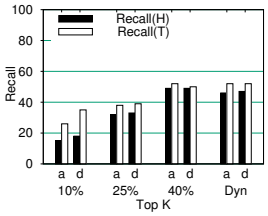
We make three observations. *First*, the recall of the average subgraphs (score_μ) is higher than the Gaussian embeddings and it outperforms IVFADC in all cases, while the Gaussian embeddings outperform it more than 80% of the cases. *Second*, the recall is poor if we consider only ≤ 10 subgraphs, especially with FB15k-237 which is the most challenging dataset. However, it increases significantly after selecting >5% of subgraphs. *Third*, with LUBM the recall of tail predictions is higher than the head predictions. The reason is that LUBM is a highly regular dataset and there are fewer objects. With the other two datasets, the recall of head predictions is higher in 70% of the cases (YAGO) and 58% of the cases (FB15K-237).

Figure 2b reports the mean rank of the first subgraph that contained the right answer (with τ=10). On LUBM and YAGO, score_{kl} returns lower ranks than score_μ in 86% and 76% of the cases respectively. On FB15K-237, the two subgraph embeddings return similar ranks. Note that the mean ranks using the dynamic strategy are always higher than the mean ranks using the fixed k. This is because a few high values of k that are selected dynamically can drastically increase the mean rank. From a more general perspective, we observe that the KL-divergence used by the Gaussian subgraph embeddings is more effective in discovering the subgraphs with potential answers than the scoring function used with the average subgraph embeddings (given the lower mean rank).

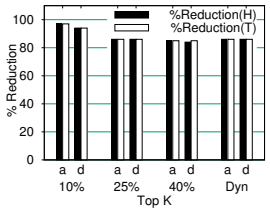
Finally, Figure 2c reports the %Reduction with TransE (the results with the other methods are analogous). The figure shows that our method is very effective in reducing the search space. For YAGO and FB15K-237, the reduction is >50% for all cases. For LUBM, score_μ gives higher reduction than the score_{kl} in five out of six cases. Moreover, we observe that the dynamic strategy returns the lowest reduction rates. This result was expected since this procedure is designed to give preference to recall than reduction.

Figures 3a, 3b, and 3c report the recall, %Reduction, and normalized mean ranks using Wikidata and TransE (with a normalized mean rank, 1 corresponds to the size of the selected subgraphs). With this dataset, the number of subgraphs is significantly higher and this makes the predictions more challenging. This leads to a lower recall than with the other KGs, but it still remains above 50% if we use our dynamic procedure for the *top k*. The reduction of the search space is significant as it is >80%. The observed mean ranks are higher than with the other datasets, which is a result that follows from the fact that this is a much more challenging dataset.

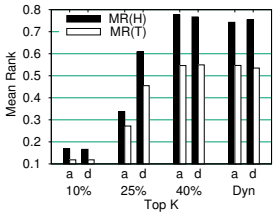
Changing τ. Figure 3e shows how the recall and %Reduction are affected when we consider more or fewer subgraphs on LUBM, YAGO and FB15K-237 with TransE. If τ is too high, then there will be only few subgraphs and our technique will not be effective. If τ is too small, then there will be too many subgraphs and it will be equally ineffective. A threshold value of 10 returns better recalls but a higher value leads to better reductions. From our experiments, it appears that $\tau = 10$ is a good value for better recall, otherwise $\tau = 50$ returns better reductions (and thus faster runtimes).



(a) Wikidata Recall



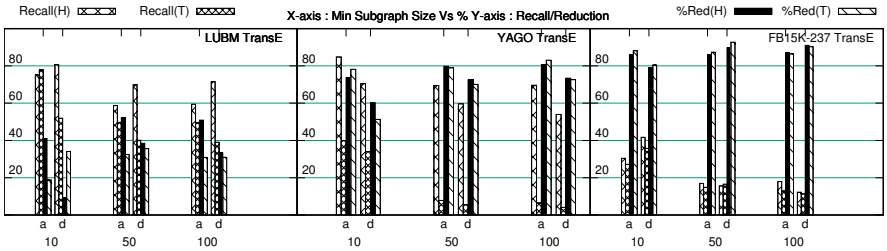
(b) Wikidata %Reduction



(c) Wikidata Mean Rank (Normalized)

Dataset	#Entities	#Triples	# Subgraphs τ=10	τ=50	τ=100
LUBM	17K	67K	1218	28	27
YAGO	266K	551K	9.9K	965	321
Wikidata	148M	3.5B	NA	1.5M	NA
FB15K237	14K	298K	8.8K	1052	374

(d) Datasets and Subgraphs statistics



(e) Recalls and %Reductions with varying subgraph Size (τ)

KG	All entities	kNN (IVFADC)	Our approach k=top10%	k=dynamic
LU (μ)	394ms	148ms	134ms	416ms
LU (KL)			5.8s	13.6s
FB23 (μ)	403ms	193ms	119ms	180ms
FB23 (KL)			7.4s	16.4s
YA (μ)	6.7s	2.4s	303ms	1.5s
YA (KL)			8.3s	14.7s

(f) Runtime of our approach vs. kNN and all entities on 150 queries

Figure 3: (1) Experiments with Wikidata and varying τ with a: score$_\mu$, d: score$_{kl}$; (2) Tables with statistics and Runtimes

Runtimes. The metrics that we used so far have the advantage that they are hardware-independent. We have also quantified the gain in terms of runtime that would be saved if we use our method to rank entities instead of considering all entities or IVFADC. The runtime comprises the (1) computation of the likelihood scores between the query and all the subgraphs, (2) ranking the subgraphs accordingly, and (3) ranking the entities in the top k subgraphs. Figure 3f reports the runtime needed for 150 random queries on FB15K-237, LUBM, YAGO using TransE and our smaller machine. We considered all three likelihood scores: score$_\mu$, score$_{kl}$, and score$_n$, $k \geq$10%, and the "Dyn" strategy. From the table, we observe that the KL divergence returns slower runtimes than IVFADC because the Equation 3 is much slower to compute than the other likelihood score and this cancels the gain obtained by considering less entities. We micro-benchmarked these runtimes and observed that while the computation of score$_\mu$ takes 200μsec, the computation of score$_{kl}$ takes 6000μsec. This makes score$_{kl}$ worthwhile only if the aggregation into subgraphs filters out some noise that would appear if we rank all entities, or if the input contains subgraphs which exclusion from the top k list yields a search space reduction that compensates for the higher ranking cost. In general, we observe that link prediction with the average-based embeddings is faster than IVFADC and than considering all entities.

From the results reported in Figures 2 and 3, we conclude that our method is able to reduce significantly the search space for relevant embeddings without excessively compromise the recall. Changing the number of top-k subgraphs leads to either a better recall or reduction. The dynamic procedure that automatically selects k appears to be a good compromise and lifts the user from the burden of finding an optimal value for this parameter. With this technique, average subgraph embeddings have a slight superior performance than the Gaussian ones. Finally, we observe that the performance of our method is not tied to a specific model. This suggests that it is a general method that can be applied to even more embedding models than the ones considered in this paper.

5 RELATED WORK AND CONCLUSION

Related work. The usage of star-shaped subgraph embeddings for KG completion was first proposed by Pal and Urbani [22]. In contrast to our work, the method at [22] adds special subgraph nodes to the original KG and then learns their embeddings like any other nodes. The main limitation of [22] is that adding extra links changes the topology of the graph and this affects the quality of the embeddings. Our approach does not suffer from this limitation.

More recently, the work at [27] proposes to find approximate answers to SPARQL queries using KG embeddings. This work is similar to ours since it also creates subgraph embeddings. However, the context and challenges are different since the method at [27] creates the embeddings on-the-fly for answering SPARQL queries while we create "query-independent" embeddings for link prediction.

Statistical relational learning methods have been applied to non-labeled graphs as well. A survey is available at [4]. Some of these methods create embeddings of subgraphs (e.g. [2, 29]) but the graphs are unlabeled; thus they are easier to handle.

Conclusion. In this paper, we showed how aggregations of KG embeddings in the form of subgraph embeddings can speed up significantly the search of similar embeddings. Thus, they can be used to perform link prediction on very large KGs. Moreover, our technique is particularly useful if the hardware resources (or other constraints) do not allow an extensive search that considers all embeddings.

Our experiments on realistic KGs (YAGO, Freebase, Wikidata) and benchmark dataset (LUBM) show that our technique outperforms k-nearest neighbor search and that it is able to significantly reduce the number of most similar entities while maintaining a good recall. Our results on Wikidata are particularly interesting because, as far as we know, they show for the first time how an embedding-based link prediction (TransE) can be applied to very large KGs with billions of facts. This enables the application of these techniques at a much larger scale than it is currently feasible.

It is interesting, as future work, to investigate whether there are other types of subgraphs that can reduce the search space. Moreover, our method returns, like all other similar methods, a ranked list of potential candidate entities, but we still need a procedure to make the final binary selection for link prediction. External sources (or other inference methods) can play a role in this process, and exploring such integration is another interesting topic for future work.

REFERENCES

[1] Grigoris Antoniou, Sotiris Batsakis, Raghava Mutharaju, Jeff Z. Pan, Guilin Qi, Ilias Tachmazidis, Jacopo Urbani, and Zhangquan Zhou. 2018. A survey of large-scale reasoning on the Web of data. *The Knowledge Engineering Review* 33 (2018), 1–43.

[2] Antoine Bordes, Sumit Chopra, and Jason Weston. 2014. Question Answering with Subgraph Embeddings. In *Proceedings of EMNLP*. ACL, Doha, Qatar, 615–620.

[3] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Proceedings of NIPS*. NIPS, Lake Tahoe, NV, USA, 2787–2795.

[4] H. Cai, V. W. Zheng, and K. C. Chang. 2018. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

[5] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, and Gestionale Antonio Ruberti. 2017. Ontology-Based Data Access and Integration. *Encyclopedia of Database Systems* 2 (01 2017), 39–71.

[6] Tim Dettmers, Minervini Pasquale, Stenetorp Pontus, and Sebastian Riedel. 2018. Convolutional 2D Knowledge Graph Embeddings. In *Proceedings of AAAI*. AAAI, New Orleans, LA, USA, 1811–1818.

[7] Dennis Diefenbach, Vanessa Lopez, Kamal Singh, and Pierre Maret. 2018. Core techniques of question answering systems over knowledge bases: a survey. *Knowledge and Information systems* 55, 3 (2018), 529–569.

[8] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *The journal of Machine Learning Research* 12 (2011), 2121–2159.

[9] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. 2005. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics* 3, 2 (2005), 158–182.

[10] Shizhu He, Kang Liu, Guoliang Ji, and Jun Zhao. 2015. Learning to Represent Knowledge Graphs with Gaussian Embedding. In *Proceedings of CIKM*. ACM, Melbourne, Australia, 623–632.

[11] Vinh Thinh Ho, Daria Stepanova, Mohamed H. Gad-Elrab, Evgeny Kharlamov, and Gerhard Weikum. 2018. Rule Learning from Knowledge Graphs Guided by Embedding Models. In *Proceedings of ISWC*. Springer, Monterey, CA, USA, 72–90.

[12] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artifificial Intelligence* 194 (2013), 28–61.

[13] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[14] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *CoRR* abs/1702.08734 (2017), 1–12. arXiv:1702.08734 http://arxiv.org/abs/1702.08734

[15] Aidan Hogan Jose L. Martinez-Rodriguez and Ivan Lopez-Arevalo. 2018. Information Extraction meets the Semantic Web: A Survey. *Semantic Web Journal* PrePress (2018), 1–81.

[16] Evgeny Kharlamov, Sebastian Brandt, Ernesto Jimenez-Ruiz, Yannis Kotidis, Steffen Lamparter, Theofilos Mailis, Christian Neuenstadt, Özgür Özçep, Christoph Pinkel, Christoforos Svingos, et al. 2016. Ontology-Based Integration of Streaming and Static Relational Data with Optique. In *Proceedings of SIGMOD*. ACM, San Francisco, CA, USA, 2109–2112.

[17] Solomon Kullback. 1978. *Information Theory and Statistics*. Dover Publications, Mineola, NW, USA.

[18] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and others. 2015. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* 6, 2 (2015), 167–195.

[19] Andrea Madotto, Chien-Sheng Wu, and Pascale Fung. 2018. Mem2Seq: Effectively Incorporating Knowledge Bases into End-to-End Task-Oriented Dialog Systems. In *Proceedings of ACL*. ACL, Melbourne, Australia, 1468–1478.

[20] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. 2016. A Review of Relational Machine Learning for Knowledge Graphs. *Proc. IEEE* 104, 1 (2016), 11–33.

[21] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. 2016. Holographic Embeddings of Knowledge Graphs. In *Proceedings of AAAI*. AAAI, Phoenix, AR, USA, 1955–1961.

[22] Soumajit Pal and Jacopo Urbani. 2017. Enhancing Knowledge Graph Completion By Embedding Correlations. In *Proceedings of CIKM*. ACM, Singapore, Singapore, 2247–2250.

[23] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. HOGWILD!: A Lock-free Approach to Parallelizing Stochastic Gradient Descent. In *Proceedings of NIPS*. NIPS, Granada, Spain, 693–701.

[24] Dominic Seyler, Tatiana Dembelova, Luciano Del Corro, Johannes Hoffart, and Gerhard Weikum. 2018. A Study of the Importance of External Knowledge in the Named Entity Recognition Task. In *Proceedings of ACL*. ACL, Melbourne, Australia, 241–246.

[25] Luke Vilnis and Andrew McCallum. 2014. Word Representations via Gaussian Embedding. *arXiv preprint arXiv:1412.6623* 1412 (2014).

[26] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledge base. *Commun. ACM* 57, 10 (2014), 78–85.

[27] Meng Wang, Ruijie Wang, Jun Liu, Yihe Chen, Lei Zhang, and Guilin Qi. 2018. Towards Empty Answers in SPARQL: Approximating Querying with RDF Embedding. In *Proceedings of ISWC*. Springer, Monterey, CA, USA, 513–529.

[28] Reichmann W.J. 1961. *Use and Abuse of Statistics*. Pelican, Oxford, UK. 320–321 pages.

[29] Pinar Yanardag and S.V.N. Vishwanathan. 2015. Deep Graph Kernels. In *Proceedings of KDD*. KDD, Sydney, NSW, Australia, 1365–1374.

[30] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proceedings of ICLR*. ICLR, San Diego, CA, USA.