Efficient Embeddings of Logical Variables for Query Answering over Incomplete Knowledge Graphs

Dingmin Wang[1]*, Yeyuan Chen[2]*, Bernardo Cuenca Grau[1]*

[1]Department of Computer Science, University of Oxford, UK
[2]The School of Computer Science and Technology, Xi'an Jiaotong University, China
dingmin.wang@cs.ox.ac.uk, yychen9961@gmail.com, bernardo.cuenca.grau@cs.ox.ac.uk

Abstract

The problem of answering complex First-order Logic queries over incomplete knowledge graphs is receiving growing attention in the literature. A promising recent approach to this problem has been to exploit neural link predictors, which can be effective in identifying individual missing triples in the incomplete graph, in order to efficiently answer complex queries. A crucial advantage of this approach over other methods is that it does not require example answers to complex queries for training, as it relies only on the availability of a trained link predictor for the knowledge graph at hand. This approach, however, can be computationally expensive during inference, and cannot deal with queries involving negation.

In this paper, we propose a novel approach that addresses all of these limitations. Experiments on established benchmark datasets demonstrate that our approach offers superior performance while significantly reducing inference times.

Introduction

Knowledge graphs (KGs) are graph-structured knowledge bases where nodes and edges represent entities of interest and their relations (Hogan et al. 2021; Heist et al. 2020). Formally, a KG can be seen as a set of triples (or logical facts) and can be represented in standard formats such as the Resource Description Framework (RDF). KGs are increasingly been used in a wide range of applications, such as Web search, question answering in digital assistants, recommender system, scientific discovery, or data integration (Li et al. 2020; Xu et al. 2020; Noy et al. 2019). Many large-scale KGs used in applications, however, are highly incomplete; as a result, *knowledge graph completion*—the problem of completing a KG with missing facts that are likely to hold in the domain of interest—has received a great deal of attention in recent years (Rossi et al. 2021).

Link predictors are ML models that can predict whether individual missing facts hold in an incomplete KG. These models can be very effective for KG completion tasks. They typically work by first learning vector representations of the entities and relations in the KG during training, and then exploiting these embeddings to predict individual facts.

In recent years, however, there has been growing interest in going beyond the prediction of simple logical facts and tackle the more general problem of answering complex FOL queries over incomplete KGs. Roughly speaking, given a query $Q(\mathbf{x})$ with answer variables $\mathbf{x}$ and a KG $\mathcal{K}$, the goal is to compute the answers to $Q(\mathbf{x})$ with respect to the (unknown) completion $\mathcal{K}^*$ of $\mathcal{K}$. One possible solution to this problem is to exploit the availability of a neural link predictor to first compute the complete KG $\mathcal{K}^*$ and then evaluate $Q(\mathbf{x})$ directly over $\mathcal{K}^*$ using standard query answering technology in data management. This solution, however, can be problematic in practice as it may require an unmanageable number of link prediction tests and may involve the materialisation of a large number of facts in the KG.

As a result, research has focused on techniques for answering queries without the need of completing the graph first. *Query embedding methods*, such as box embeddings (Ren, Hu, and Leskovec 2020), beta embeddings (Ren and Leskovec 2020), and cone embeddings (Zhang et al. 2021), compute vector embeddings for the queries and then treat query answering as a nearest-neighbor search problem in the latent space. Training such models is, however, problematic, as they can only deal with query patterns seen during training, and hence training typically requires the availability of answers to a wide variety of queries.

To address this limitation, Arakelyan et al. proposed the CQD model, which relies only on the availability of vector embeddings provided by a trained neural link predictor and hence does not require example answers to complex queries for training (Arakelyan et al. 2021). This approach, however, also comes with a number of limitations. First, it is restricted to a subclass of positive existential queries—FOL queries constructed using only conjunction, disjunction and existential quantification; thus, it cannot handle queries involving other important constructs such as negation. Second, CQD relies on a computationally expensive combinatorial optimisation problem to search for likely query answers using the embeddings provided by the neural link predictor; this can lead to slow inference times and limit the applicability of CQD to scenarios with low latency requirements.

In this paper, we propose a novel approach to query answering over incomplete KGs that addresses these limitations. Our approach extends the query language of CQD to support negation while at the same time significantly reduc-

*All authors make equal contributions to this paper.

ing the computational resources needed for both the training and inference stages. Furthermore, similarly to CQD and in contrast to query embedding methods, our approach relies solely on the availability of a trained neural link predictor and hence does not require example answers to complex queries for training. We have implemented our approach in a system called `Var2Vec`. Our extensive experiments on three standard benchmark datasets show that `Var2Vec` can deliver superior performance while significantly improving both training and inference times.[1]

Background

Knowledge Graphs. A knowledge graph $\mathcal{K}$ over a vocabulary consisting of pairwise disjoint sets of entities $\mathcal{E}$ and relations $\mathcal{R}$ is a finite set of triples $\langle e_1, r, e_2 \rangle$ where $e_1, e_2 \in \mathcal{E}$ and $r \in \mathcal{R}$. A triple $\langle e_1, r, e_2 \rangle$ is equivalent to a fact $r(e_1, e_2)$, and hence a KG $\mathcal{K}$ can also be equivalently seen a FOL knowledge base consisting of a finite set of facts.

Queries. We consider the subclass of FOL queries defined next. Consider a vocabulary consisting of entities $\mathcal{E}$ and relations $\mathcal{R}$, and let $\mathcal{V}$ be a set of variables pair-wise disjoint with $\mathcal{E}$ and $\mathcal{R}$. A term is either an entity in $\mathcal{E}$ or a variable in $\mathcal{V}$. An atom is a formula $r(t_1, t_2)$ where $r \in \mathcal{R}$ and each t_i is a term. A literal is an atom or the negation of an atom. The subclass of FOL formulas φ that we consider is then inductively defined according to the following grammar, where l is a literal and x is a variable:

$$\varphi ::= l \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x.\varphi. \tag{1}$$

Variable occurrences in the scope of a quantifier are bound, whereas other variable occurrences are free. A *query* is a formula with a single free variable, called the *answer variable*.

Note that we consider an extension of the class of monadic positive existential queries (i.e., FOL queries with a single answer variable constructed using only conjunction, disjunction and existential quantification) where, in addition, we allow for a restricted form of negation occurring only in front of atoms. This is in contrast to previous works, in which queries were typically restricted to monadic positive existential queries in disjunctive normal form.

The dependency graph of a query $Q(x)$ is a directed graph where the nodes are the terms occurring in the query and where there is a directed edge from term t_1 to term t_2 whenever an atom of the form $r(t_1, t_2)$ occurs as a sub-formula in the query. Following the literature, we restrict ourselves to queries where the dependency graph is connected and acyclic, and where the answer variable is the single sink node and each source node in the graph is an entity (and thus not a variable). In what follows, we will refer to the FOL queries satisfying all the aforementioned syntactic restrictions as *admissible*. To the best of our knowledge, the class of admissible queries covers all types of queries supported by existing embedding-based query answering approaches in the literature, as well as all types of queries in existing benchmarks. An example admissible query and its corresponding dependency graph are shown in Figure 1.

[1]Code available at https://github.com/wdimmy/Var2Vec.

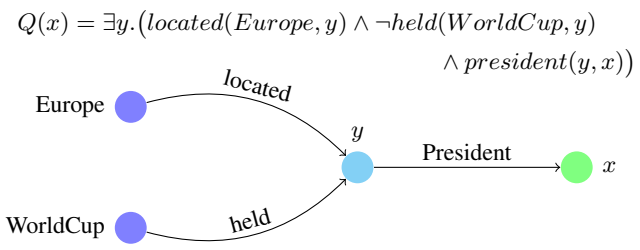


Figure 1: Formalisation of the query '*list the presidents of European countries that have never held the World Cup*' and the corresponding dependency graph. The sink node corresponding to the answer variable is depicted in green, whereas the source nodes (also called anchor nodes in the literature) are entities depicted in purple.

The semantics of query answering is standard. We say that an entity $e \in \mathcal{E}$ is an answer to query $Q(x)$ with respect to a KG $\mathcal{K}$ if $\mathcal{K} \models Q(e)$, and we denote the set of answers to Q with respect to $\mathcal{K}$ as $[\![Q]\!]_{\mathcal{K}}$.

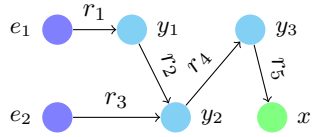
Neural Link Predictors. In the context of this paper, we define a neural link predictor $\mathcal{M}$ for a finite vocabulary of entities $\mathcal{E}$ and relations $\mathcal{R}$ as a pair $\langle \mathsf{enc}_{\mathcal{M}}, \mathsf{dec}_{\mathcal{M}} \rangle$ consisting of an encoding function $\mathsf{enc}_{\mathcal{M}}$ and a decoding function $\mathsf{dec}_{\mathcal{M}}$, respectively. The encoding function maps each entity $e \in \mathcal{E}$ to a k-dimensional real-valued vector and each relation $r \in \mathcal{R}$ to a k'-dimensional real-valued vector, where k and k' are hyper-parameters of the model representing the dimensions of entity and relation embeddings, respectively. In turn, the decoding function $\mathsf{dec}_{\mathcal{M}}$ is a function mapping each triple of vectors in $\mathbb{R}^k \times \mathbb{R}^{k'} \times \mathbb{R}^k$ to a real value between 0 and 1. The vector components in the encoding of each entity and relation of the vocabulary constitute the learnable parameters of the model, whereas the decoding function is used to assign to each possible fact $r(e_1, e_2)$ in the vocabulary a probability $\mathsf{dec}_{\mathcal{M}}(\mathsf{enc}_{\mathcal{M}}(e_1), \mathsf{enc}_{\mathcal{M}}(r), \mathsf{enc}_{\mathcal{M}}(e_2))$.

t-norms and t-conorms. Triangular norms (or *t-norms*) are generalisations of the logical conjunction operator that are typically adopted in fuzzy logics (Hájek 2013; Gupta and Qi 1991). Formally, a t-norm is a binary operation $\top$ on $[0, 1]$ satisfying the following properties:

- $\top(x, y) = \top(y, x)$ (commutativity);
- $\top(x, \top(y, z)) = \top(\top(x, y), z)$ (associativity);
- $y \leq z \rightarrow \top(x, y) \leq \top(x, z)$ (monotonicity); and
- $\top(x, 1) = x$ (neutral element 1).

Examples used in practice include the Gödel t-norm $\top(x, y) = min(x, y)$, the product t-norm $\top(x, y) = x \cdot y$ and the Lukasiewicz t-norm $\top(x, y) = max(x + y - 1, 0)$.

The notion of a *t-cornorms* is dual to that of a t-norm and is used to generalise logical disjunction. Formally, a t-conorm is a binary operation $\perp$ on $[0, 1]$ satisfying the aforementioned communatitivy, associativity and monotonicity properties and where the neutral element is 0 instead of 1 (that is, $\perp(x, 0) = x$). Examples of *t-conorms* used in practice are the Gödel t-conorm $\perp(x, y) = max(x, y)$, the prod-



$$\mathbf{y_1} = \mathbf{W} \cdot [\mathbf{e_1} : \mathbf{r_1}]$$
$$\mathbf{y_2} = \frac{\mathbf{W} \cdot [\mathbf{y_1} : \mathbf{r_2}] + \mathbf{W} \cdot [\mathbf{e_2} : \mathbf{r_3}]}{2}$$
$$\mathbf{y_3} = \mathbf{W} \cdot [\mathbf{y_2} : \mathbf{r_4}]$$

Figure 2: An example about how each intermediate variable (y_1, y_2 and y_3) node embedding in a complex logic query is calculated. Note that source nodes e_1 and e_2 and all relation nodes (r_1,r_2,r_3,r_4 and r_5) could be obtained directly from the trained neural link prediction model.

uct t-conorm $\perp(x, y) = x + y - x \cdot y$ and the Lukasiewicz t-conorm $\perp(x, y) = min(x + y, 1)$.

Method

In this section, we present our approach `Var2Vec` to query answering over incomplete KGs. Our model relies on the availability of a trained neural link predictor $\mathcal{M}$ for a finite vocabulary of entities $\mathcal{E}$ and relations $\mathcal{R}$; it is compatible with most transductive link prediction methods proposed in the literature, such as TransE (Bordes et al. 2013), DistMult (Yang et al. 2015), ComplEx (Trouillon et al. 2016), RotaE (Sun et al. 2018), and QuatE (Zhang et al. 2019). Indeed, the only requirement is that $\mathcal{M}$ can provide vector embeddings for each entity and relation in the vocabulary as well as a probability for each candidate fact. Then, given *any* admissible query $Q(x)$ and any KG $\mathcal{K}$ mentioning only relations from $\mathcal{R}$ and entities of $\mathcal{E}$, our model estimates the set of answers to $Q(x)$ with respect to the completion $\mathcal{K}^*$ of facts predicted to hold by $\mathcal{M}$.

At training time, we first train $\mathcal{M}$ in the usual way (or take a pre-trained link predictor instead); then, using the embeddings provided by $\mathcal{M}$, we train a weight matrix $\mathbf{W}$ that will be used to generate vector embeddings for the existentially quantified variables of any admissible input query at inference time.[2] The training of $\mathbf{W}$ is described in Section .

At inference time, we first traverse the input query $Q(x)$ to generate, using $\mathbf{W}$ and $\mathcal{M}$, vector embeddings for each existentially quantified variable; this is described in Section . Subsequently, as described in Section , we score each entity e in $\mathcal{E}$ to provide a likelihood of $Q(e)$ being true in $\mathcal{K}^*$; this allows us to rank the possible answers to the query.

Learning the Weight Matrix

Although a trained neural link predictor $\mathcal{M}$ can calculate a score for any fact $r(e_1, e_2)$ over its vocabulary, it cannot provide a score for atoms involving variables, such as $r(e_1, y)$ or $r(y_1, y_2)$, unless we first embed these variables as vectors.

To this end, we propose to train and exploit a weight matrix $\mathbf{W} \in \mathbb{R}^{(k+k') \times k}$, where k and k' are the dimensions of the entity embeddings and the relation embeddings in $\mathcal{M}$, respectively. For instance, given an atom $r(e, y)$, the embedding of variable y is $\mathbf{y} = \mathbf{W} \cdot \left[\mathsf{enc}_{\mathcal{M}}(e) : \mathsf{enc}_{\mathcal{M}}(r)\right]$, where $[:]$ is the vector concatenation operation.

[2]Here and in the rest of the paper we omit bias terms for clarity.

Given a set of training facts of the form $r(e_1, e_2)$, with $e_1, e_2 \in \mathcal{E}$ and $r \in \mathcal{R}$, we minimise a L2-norm loss given next, where the elements of $\mathbf{W}$ are the only parameters:

$$\mathcal{L} = ||(\mathbf{W} \cdot \left[\mathsf{enc}_{\mathcal{M}}(e_1) : \mathsf{enc}_{\mathcal{M}}(r)\right]) - \mathsf{enc}_{\mathcal{M}}(e_2))||_2. \quad (2)$$

Intuitively, $\mathbf{W}$ is trained to transform the embeddings of the first argument of a training fact and the fact's relation into the embedding of the fact's second argument. Note that, to train $\mathbf{W}$, we only require a set of training facts, which could be the same facts used to train $\mathcal{M}$; this is in contrast to query embedding models such as GQE (Hamilton et al. 2018), Query2Box (Ren, Hu, and Leskovec 2020), BetaE (Ren and Leskovec 2020) and ConE (Zhang et al. 2021), which usually require large amounts of training queries and answers in order to achieve good performance.

Generating Variable Embeddings

Given an admissible query $Q(x)$, a trained neural link predictor $\mathcal{M}$, and a trained weight matrix $\mathbf{W}$, we can generate vector embeddings for all existentially quantified variables in $Q(x)$. To this end, we traverse the query as described next where G_Q is the dependency graph of $Q(x)$. First, we mark the source terms t of G_Q as visited; these are entities because the query is admissible, and hence we can obtain their embedding $\mathbf{t} = \mathsf{enc}_{\mathcal{M}}(t)$; we also obtain the embedding of all relations r in the query as $\mathbf{r} = \mathsf{enc}_{\mathcal{M}}(r)$. Then, we repeat the following process until all existentially quantified variables in $Q(x)$ are marked as visited.

- Iterate through each unvisited existentially quantified variable y such that each atom in $Q(x)$ involving y is of the form $r_i(t_i, y)$ where each such $t_1, \ldots, t_n$ is an entity or an existentially quantified variable marked as visited.
  - compute the embedding $\mathbf{y}$ of y as
  $$\mathbf{y} = \frac{\sum_{i=1}^{n} \mathbf{W} \cdot [\mathbf{t_i} : \mathbf{r_i}]}{n} \quad (3)$$
  where $\mathbf{t}_i$ and $\mathbf{r_i}$ are the embeddings of term t_i (which is available at this stage due to admissibility of the query) and the relation r_i, respectively.
  - Mark y as visited.

Example 0.1. *Consider the admissible query*

$$Q(x) = \exists y_1 \exists y_2 \exists y_3 . (r_1(e_1, y_1) \wedge r_3(e_2, y_2) \wedge r_2(y_1, y_2) \wedge r_4(y_2, y_3) \wedge r_5(y_3, x)).$$

It's dependency graph is provided in Figure 2. We start by computing the embeddings $\mathbf{e}_1$ *and* $\mathbf{e}_2$ *of entities* e_1 *and* e_2 *and the embeddings* $\mathbf{r}_1, \ldots \mathbf{r}_5$ *of relations* $r_1, \ldots, r_5$ *using the link predictor* $\mathcal{M}$*. We then process variable* y_1 *and obtain its embedding* $\mathbf{y_1}$ *as* $\mathbf{W} \cdot [\mathbf{e_1} : \mathbf{r_1}]$*. The next variable to process is* y_2 *and we obtain its embedding* $\mathbf{y_2}$ *as the average of* $\mathbf{W} \cdot [\mathbf{e_2} : \mathbf{r_3}]$ *and* $\mathbf{W} \cdot [\mathbf{y_1} : \mathbf{r_2}]$*. Finally, we can now obtain the embedding* $\mathbf{y_3}$ *of variable* y_3 *as* $\mathbf{W} \cdot [\mathbf{y_2} : \mathbf{r_4}]$.

The process is clearly linear in the size of the query as each atom is considered only once. Note also that, to obtain variable embeddings at this stage, we do not define a special treatment of negation or disjunction as the query is traversed according to its dependency graph only.

Scoring for Query-Entity Pairs

Once we have obtained vector embeddings for all entities and existentially quantified variables mentioned in the input query $Q(x)$, we can exploit the decoding function $\mathsf{dec}_{\mathcal{M}}$ to compute a score for each entity $e \in \mathcal{E}$ estimating the likelihood of $Q(e)$ being true in the completion $\mathcal{K}^*$. For each entity e, the computation of the score is performed inductively on the structure of $Q(e)$. Formally, let us fix a t-norm function $\top$ and a t-conorm function $\bot$. Then, the scoring function σ maps each admissible FOL sentence φ constructed according to the grammar in Equation (1) over the relevant vocabulary to a real-valued score $\sigma(\varphi) \in [0, 1]$ as given next, where for a term t (constant or existentially quantified variable) and relation r occurring in φ, we use $\mathbf{t}$ and $\mathbf{r}$ to respectively denote their vector embedding computed as described in Section .

- if φ is an atom of the form $r(t_1, t_2)$, for t_1 and t_2 terms, then $\sigma(\varphi) = \mathsf{dec}_{\mathcal{M}}(\mathbf{t_1}, \mathbf{r}, \mathbf{t_2})$;
- if φ is a negative literal of the form $\neg r(t_1, t_2)$, then $\sigma(\varphi) = 1 - \sigma(r(t_1, t_2))$;
- if $\varphi = \varphi_1 \wedge \varphi_2$, then $\sigma(\varphi) = \top(\sigma(\varphi_1), \sigma(\varphi_2))$;
- if $\varphi = \varphi_1 \vee \varphi_2$, then $\sigma(\varphi) = \bot(\sigma(\varphi_1), \sigma(\varphi_2))$; and
- if φ is of the form $\exists t.\varphi_1$, then $\sigma(\varphi) = \sigma(\varphi_1)$.

Note that the computation of the score for a given entity is linear in the size of the query; furthermore, answering the query requires a score computation for each entity in the KG. Once, all scores have been computed, the possible answers can be ranked before they are returned to the user.

Experiments

We have implemented our `Var2Vec` model and evaluated its performance against state-of-the-art baselines.

In this section, we describe the results of our evaluation on a suite of an established benchmark for query answering over incomplete KGs.

Datasets and Queries We consider the query answering benchmark proposed in (Ren, Hu, and Leskovec 2020). The benchmark provides three standard KGs commonly used in the KG completion literature: FB15k (Bordes et al. 2013), FB15k-237 (Toutanova and Chen 2015), and a subset of the NELL995 KG (Xiong, Hoang, and Wang 2017). The benchmark splits the facts in each KG into training, validation and testing subsets, where the training subset is contained in the validation subset and the validation subset is in turn contained in the testing subset. Given a query Q and a benchmark KG $\mathcal{K}$, let us denote with $[\![Q]\!]^{\text{train}}_{\mathcal{K}}$, $[\![Q]\!]^{\text{val}}_{\mathcal{K}}$ and $[\![Q]\!]^{\text{test}}_{\mathcal{K}}$ (where $[\![Q]\!]^{\text{train}}_{\mathcal{K}} \subseteq [\![Q]\!]^{\text{val}}_{\mathcal{K}} \subseteq [\![Q]\!]^{\text{test}}_{\mathcal{K}}$) the set of answers to Q with respect to the training, validation, and testing subsets of $\mathcal{K}$, respectively. We evaluate on $[\![Q]\!]^{\text{train}}_{\mathcal{K}} \setminus [\![Q]\!]^{\text{val}}_{\mathcal{K}}$ at the validation stage and the reported results are calculated based on $[\![Q]\!]^{\text{val}}_{\mathcal{K}} \setminus [\![Q]\!]^{\text{test}}_{\mathcal{K}}$ at the testing stage.

We consider the 14 query patterns shown in Figure 3, where 9 of them correspond to queries without negation considered in (Ren, Hu, and Leskovec 2020), and the remaining 5 query patterns involving negation were later proposed in (Ren and Leskovec 2020). Query pattern $1p$ is the simplest pattern and it corresponds to facts, whereas the remaining patterns represent increasingly complex query structures. The queries used for training, validation, and testing are obtained by instantiating each query pattern multiple times for each benchmark KG; this is done by choosing suitable relations and entities from the KG. Table 1 summarises the number of instantiations of each query pattern and each KG provided by the benchmark.

KG	Training		Validation		Test	
	1p	others	1p	others	1p	others
FB15k	273,710	273,710	59,097	8,000	67,016	8,000
FB15k-237	149,689	149,689	20,101	5,000	22,812	5,000
NELL995	107,982	107,982	16,927	4,000	17,034	4,000

Table 1: Number of training, validation, and test queries generated for different query patterns.

Method	Supported Operators
GQE (Hamilton et al. 2018)	$\exists, \wedge$
Query2Box (Ren, Hu, and Leskovec 2020)	$\exists, \wedge, \vee$
BetaE (Ren and Leskovec 2020)	$\exists, \wedge, \vee, \neg$
MLP (Amayuelas et al. 2021)	$\exists, \wedge, \vee, \neg$
ConE (Zhang et al. 2021)	$\exists, \wedge, \vee, \neg$
CQD (Arakelyan et al. 2021)	$\exists, \wedge, \vee$
FuzzQE (Chen, Hu, and Sun 2022)	$\exists, \wedge, \vee, \neg$
`Var2Vec`	$\exists, \wedge, \vee, \neg$

Table 2: Considered baselines and our approach `Var2Vec`.

Baselines We compare `Var2Vec` against the state-of-the-art baselines indicated in Table 2. To perform the experiments, we used the code for the relevant baseline provided by the authors; the only exception was FuzzQZ (Chen, Hu, and Sun 2022), where we reported the results provided in their paper since the code for the system is not publicly available. All baselines support only admissible queries as described in the preliminaries; however, GQE, Query2Box and CQD are restricted to admissible queries without negation. We did not include the beam search scheme of CQD scheme as a baseline in our experiments since, as show in Figure 4, it is computationally too expensive; for example, a single 12GB Titan X GPU returned a memory error for a 5000-batch query inference on NELL995; as a result, we used only the continuous optimisation scheme of CQD.

It is worth reiterating that both our approach and CQD require only facts (i.e., queries of pattern 1p) for training, whereas training for the remaining baselines also requires the complex queries and their answers in the benchmark.

Model Details For each dataset, we pretained a link predictor (ComplexE) only using the $1p$ queries from their

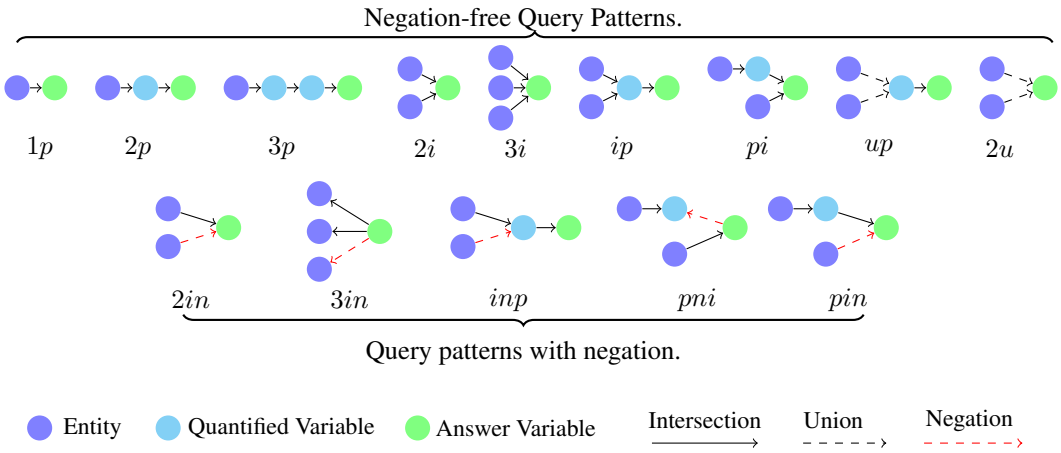


Figure 3: Query patterns considered in the experiments. In the naming of query structures, p, i, u and n stand for Projection, Intersection, Union and Negation, respectively.

training dataset. The number of training epochs was 100 and the embedding size of both the entity and the relation was 2000. The dimension of $\mathbf{W}$ was 4000×2000. We used the Adam Optimizer (Kingma and Ba 2014) with learning rates lr=0.1/0.01/0.001 to optimize $\mathbf{W}$ and used the best one on the validation set. The batch size was 1000. In our experiments, by default, we use product T-norm ($\perp(x, y) = x \cdot y$) and product T-conorm ($\top(x, y) = x + y - x \cdot y$).

Evaluation Metrics Following prior work (Ren, Hu, and Leskovec 2020; Ren and Leskovec 2020), we adopted the Mean Reciprocal Rank (MRR) metric for our evaluation, which is calculated as given next for a test query Q and a test KG $\mathcal{K}$, where $rank_e$ denotes the rank of entity e as a query answer amongst all entities in the KG:

$$\mathsf{MRR}(Q) = \frac{1}{|[\![Q]\!]^{\text{test}}_{\mathcal{K}} \setminus [\![Q]\!]^{\text{val}}_{\mathcal{K}}|} \sum_{e \in [\![Q]\!]^{\text{test}}_{\mathcal{K}} \setminus [\![Q]\!]^{\text{val}}_{\mathcal{K}}} \frac{1}{rank_e}.$$

Main Results. Table 3 summarises the results of our evaluation on all three benchmark KGs, where Avg_e (respectively, Avg_n) represents the average MRR for all queries in the benchmark corresponding to negation-free patterns (respectively, queries corresponding to patterns with negation). Overall, the performance of our approach is highly competitive. Compared with GQE, Query2Box and BetaE, our method achieves an average improvement on MRR of 37.7%, 22.9% and 46.2% respectively for queries without negation, and also outperforms significantly all of these approaches for queries with negation. Our approach also outperforms ConE and MLP both for queries with and without negation, especially for complex query patterns such as ip and pi, which are patterns unseen during training for all models; this suggests that our approach generalises better to new query patterns than the baselines. Compared to CQD, our approach also displays better average performance for all three benchmark KGs. Finally, we compared our approach to FuzzQE, which also exploits the availability of a neural link predictor; we observe that our approach also outperforms FuzzQE in most cases, with the only exception of queries with negation on NELL995.

Figure 4 provides training and average inference times for all the evaluated approaches on FB15k. We can see that our approach can be trained and applied very efficiently. In particular, our approach is 12/46 times faster than CQD-CO/CQD-Beam during inference and significantly faster than embedding-based models during training.

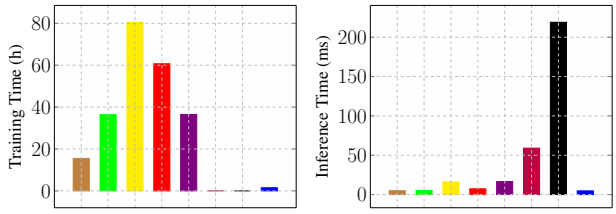


Figure 4: Training and inference times on FB15k. Inference time refers to the average time per query. From left (brown) to right (blue), it represents GQE, Query2Box, BetaE, MLP, ConE, CQD-CO, CQD-Beam and `Var2Vec`, respectively.

Ablation Studies. We first analysed the influence of the chosen link predictor on our results. Our system uses ComplexEx by default. If we choose DisMult instead, we can observe in Table 4 a slight decrease in performance. This is unsurprising given that ComplexEx ourperforms DistMult in link prediction tasks over the benchmark KGs.

We also analysed the influence of the choice of t-norm and t-conorms, which play an important role in computing the scores for query answers. Our system uses product t-norms and t-conorms by default, and we also experimented with Gödel t-norms and t-conorms. Table 5 shows that using product t-norms and t-conorms leads to superior performance on all benchmark KGs.

Finally, we explored the possibility of replacing the weight matrix $\mathbf{W}$ with a fully-connected neural network

Method	Avg_e	Avg_n	1p	2p	3p	2i	3i	ip	pi	2u	up	2in	3in	inp	pin	pni
NELL995																
GQE	18.6	–	32.8	11.9	9.6	27.5	35.2	14.4	18.4	8.5	8.8	–	–	–	–	–
Q2B	22.9	–	42.2	14.0	11.2	33.3	44.5	16.8	22.4	11.3	10.3	–	–	–	–	–
CQD-CO	28.8	–	60.8	18.3	13.2	41.0	41.5	22.5	30.3	17.6	13.7	–	–	–	–	–
BetaE	24.6	5.9	53.0	13.0	11.4	37.6	47.5	14.3	24.1	12.2	8.5	5.1	7.8	10.0	3.1	3.5
ConE	27.2	6.4	53.1	16.1	13.9	40.0	50.8	17.5	26.3	15.3	11.3	5.7	8.1	10.8	3.5	3.9
MLP	25.0	6.0	52.7	15.4	14.0	36.4	45.4	15.8	22.1	13.2	10.0	5.1	8.0	10.0	3.6	3.6
FuzzQE	27.1	7.3	57.6	17.2	13.3	38.2	41.5	27.0	19.4	16.9	12.7	9.1	8.3	8.9	4.4	5.6
Var2Vec	**30.1**	**7.5**	60.8	18.0	11.9	42.2	52.2	22.7	30.9	19.2	12.2	6.8	8.8	10.6	5.4	5.9
FB15k-237																
GQE	16.3	–	35.0	7.2	5.3	23.3	34.6	10.7	16.5	8.2	5.7	–	–	–	–	–
Q2B	20.1	–	40.6	9.4	6.8	29.5	42.3	12.6	21.2	11.3	7.6	–	–	–	–	–
CQD-CO	21.8	-	46.7	9.5	6.3	31.2	40.6	23.6	16.0	14.5	8.2	–	–	–	–	–
BetaE	20.9	5.4	39.0	10.9	10.0	28.8	42.5	12.6	22.4	12.4	9.7	5.1	7.9	7.4	3.6	3.4
ConE	23.2	5.9	42.3	12.7	10.7	32.6	46.9	13.6	25.2	14.2	10.6	5.4	8.6	7.8	4.0	3.6
MLP	22.1	**6.7**	41.4	11.7	9.9	32.1	44.6	13.0	24.5	12.6	9.3	6.4	10.6	8.0	4.6	4.4
FuzzQE	21.8	6.6	44.0	10.8	8.6	32.3	41.4	22.7	15.1	13.5	8.7	7.7	9.5	7.0	4.1	4.7
Var2Vec	**23.4**	6.4	46.7	10.8	8.4	33.5	46.5	16.3	22.5	16.6	8.8	5.4	10.5	6.2	4.4	5.3
FB15k																
GQE	28.0	–	54.6	15.3	10.8	39.7	51.4	19.1	27.6	22.1	11.6	–	–	–	–	–
Q2B	38.0	–	68.0	21.0	14.2	55.1	66.5	26.1	39.4	35.1	16.7	–	–	–	–	–
CQD-CO	48.6	–	89.4	27.5	15.0	76.9	80.8	35.1	46.4	42.7	23.5	–	–	–	–	–
BetaE	41.6	11.8	65.1	25.7	24.7	55.8	66.5	28.1	43.9	40.1	25.2	14.3	14.7	11.5	6.5	12.4
ConE	49.7	14.8	73.2	33.7	29.0	64.6	73.6	35.5	50.9	55.3	31.4	17.9	18.7	12.5	9.8	15.1
MLP	41.6	13.9	66.9	29.3	24.4	56.1	66.2	26.7	44.8	33.5	26.4	16.0	17.3	13.2	8.3	14.6
FuzzyQE*	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Var2Vec	**51.0**	**18.9**	89.4	26.0	16.6	79.0	82.6	34.2	46.0	62.8	22.1	23.6	31.4	9.8	9.8	19.7

* The original paper of FuzzyQE does not report the result on the FB15k dataset.

Table 3: MRR results (%) of baselines and our model on benchmark queries grouped by query pattern. Avg_e and Avg_n denote the average MRR on queries with and without negation, respectively.

Method	Avg	1p	2p	3p	2i	3i	ip	pi	2u	up
NELL995										
DisMult	28.1	58.6	13.9	10.3	40.7	51.0	18.9	28.9	18.7	11.5
ComplexE	**30.1**	60.8	18.0	11.9	42.2	52.2	22.7	30.9	19.2	12.8
FB15k-237										
DisMult	21.6	45.4	8.0	6.6	31.5	44.0	13.2	21.0	16.2	8.1
ComplexE	**23.2**	46.7	10.3	7.1	33.5	46.5	16.3	22.5	16.6	8.8
FB15k										
DisMult	41.8	73.4	17.6	15.4	60.5	68.8	30.1	39.5	50.5	20.5
ComplexE	**51.0**	89.4	26.0	16.6	79.0	82.6	34.2	46.0	62.8	22.1

Table 4: MRR results (%) with DisMult and ComplexE .

equipped with ReLU activations. We considered architectures with one and two hidden layers in which ReLU is applied following matrix application in each hidden layer. The results we obtained on the FB15k KG are summarised in Table 6. We can observe that, as the number of layers increases, the training times increase significantly but the performance of the model does not show a noticeable improvement. These results thus speak in favour of using a simple weight matrix instead of a more complex neural architecture for computing variable embeddings.

Method	Avg	1p	2p	3p	2i	3i	ip	pi	2u	up
NELL995										
MM	25.9	60.8	18.0	7.1	39.7	46.4	15.8	13.8	19.9	11.3
PP	**30.1**	60.8	18.0	11.9	42.2	52.2	22.7	30.9	19.2	12.8
FB15k-237										
MM	19.4	46.7	8.8	6.9	30.9	39.6	11.5	6.7	17.2	6.5
PP	**23.2**	46.7	10.3	7.1	33.5	46.5	16.3	22.5	16.6	8.8
FB15k										
MM	45	89.4	16.8	11.4	80.5	84.1	21.8	10.5	74.1	16.4
PP	**51.0**	89.4	26.0	16.6	79.0	82.6	34.2	46.0	62.8	22.1

Table 5: MRR results (%) with different T-norms. MM represents the Min-Max Gödel combination and PP for the Product-Product combination.

Case Study. In this section we discuss the variable embeddings obtained by our approach. We chose as a case study the test query in the NELL995 benchmark listing "all countries

Setting	Time	Avg	1p	2p	3p	2i	3i	ip	pi	2u	up
Matrix	86m	**51.0**	89.4	26.0	16.6	79.0	82.6	34.2	46.0	62.8	22.1
2-layer	248m	50.4	89.8	26.3	16.6	77.8	79.6	34.5	44.5	61.7	22.8
3-layer	539m	**51.0**	90.9	25.4	16.3	80.5	83.0	35.1	44.2	63.0	20.8

Table 6: MRR results (%) with different neural layers on FB15K. Times indicate training times.

where an official language of the country is used for teaching Divinity at universities". This is a conjunctive query written as follows, with $Divinity$ an entity NELL995:

$$Q(x) = \exists y_1 \exists y_2 . (CourseOf(Divinity, y_1) \wedge TeachLanguage(y_1, y_2) \wedge OfficialLanguage(y_2, x))$$

An answer to this query can be obtained by matching the answer variable x to US and the existentially quantified variables y_1 and y_2 to $Harvard\ University$ and $English$, respectively. Intuitively, we would expect the embedding of variables y_1 and y_2 obtained as described in Section to be close to the embeddings for university entities and language entities, respectively.

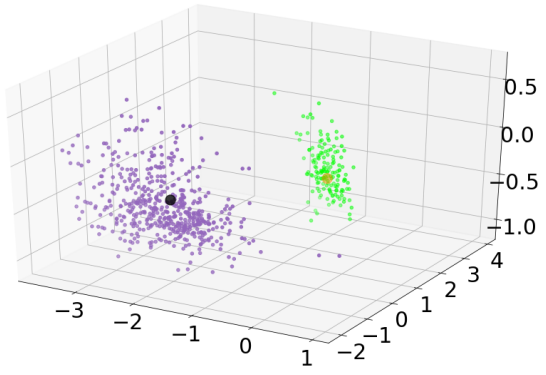


Figure 5: Visualisation of entity and variable embeddings for the case study. Th black and yellow points represents the embedding of variable y_1 and y_2, respectively; and the purple and green clusters denote the embeddings of university entities and language entities, respectively.

To verify this intuition, we have performed a Principal Component Analysis (PCA) (Abdi and Williams 2010) on the embeddings. The results are depicted in Figure 5. Going from left to right, the first cluster of points represents the embeddings of University entities (in purple) and the embedding of variable y_1 (in black), whereas the second cluster of points represent embeddings of language entities (green points) and the embedding of y_2 (yellow point). We can observe that variable embeddings show strong correspondence with the embedding of their representative entities in terms of their relative positions in Figure 5. This illustrates that the learnt weight matrix $\mathbf{W}$ is able to adequately embed variables in queries with joins.

Related Work

Knowledge graph completion is the problem of completing a KG with missing facts that are likely to hold in the domain of interest. Neural link predictors are models capable of scoring the likelihood of a particular fact and/or ranking the most likely answers to an atomic query of the form $Q(x) = r(e, x)$ or $Q(x) = r(x, e)$ for r a relation and e an entity in the KG. Knowledge graph completion has received a great deal of attention in recent years and a wide range of neural link predictors have been proposed. Prominent examples include TransE (Bordes et al. 2013), DistMult (Yang et al. 2015), ComplEx (Trouillon et al. 2016), and RotaE (Sun et al. 2018). We refer the readers to (Rossi et al. 2021) for a more detailed discussion about these models. It is worth mentioning, however, that the neural link predictors that are compatible with our approach are *transductive*—that is, they score individual facts by first learning embeddings for entities and relations in the KG and as a result they can only make predictions for entities seen during training. In recent years, however, there has also been increasing interest on *inductive* link predictors, which are able to make predictions for KGs involving arbitrary entities (Hao et al. 2020; Teru, Denis, and Hamilton 2020; Liu et al. 2021).

In this paper, we focus on the query answering problem over incomplete KGs, which generalises link prediction to FOL queries that go beyond simple facts. This problem is receiving increasing attention, and prominent approaches include GQE (Hamilton et al. 2018), Query2box (Ren, Hu, and Leskovec 2020), BetaE (Ren and Leskovec 2020), MLP (Amayuelas et al. 2021) and ConE (Zhang et al. 2021). Specifically, GQE is one of the earliest approaches supporting only admissible conjunctive queries. Query2Box further supports disjunction by representing queries as a box in a latent space; box embeddings, however, cannot support negation since the complement of a box in the Euclidean space is no longer a box. Beta embeddings (Ren and Leskovec 2020) were later proposed so address this limitation and support admissible queries involving all four Boolean operations. Subsequent approaches, such as ConE and MLP, improved model performance by introducing increasingly complex architectures; these query embedding-based approaches are, however, data hungry and usually require millions of training queries to achieve a satisfying performance. CQD (Arakelyan et al. 2021) exploits trained neural link predictors together with fuzzy logic operators to avoid the need for complex queries during training. CQD requires only facts for training and displays improved out-of-distribution generalisation. CQD, however, does not support negation and has low inference efficiency.

Conclusion

We have presented a novel approach to query answering over incomplete KGs that addresses some of the key limitations in prior work. First, we support admissible queries involving all Boolean operators. Second, we require only facts for training, and training can be performed very efficiently. Finally, our approach displays superior accuracy while at the same time significantly reducing inference times.

Acknowledgments

This work was supported in whole or in part by the EPSRC projects OASIS (EP/S032347/1), ConCuR (EP/V050869/1) and UK FIRES (EP/S019111/1), the SIRIUS Centre for Scalable Data Access, and Samsung Research UK. For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission. The authors would like to thank Shuai Zhang, Qi Liu, and Yue Zhang for their insightful discussions and support.

References

Abdi, H.; and Williams, L. J. 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4): 433–459.

Amayuelas, A.; Zhang, S.; Rao, X. S.; and Zhang, C. 2021. Neural Methods for Logical Reasoning over Knowledge Graphs. In *International Conference on Learning Representations (ICLR)*.

Arakelyan, E.; Daza, D.; Minervini, P.; and Cochez, M. 2021. Complex Query Answering with Neural Link Predictors. In *International Conference on Learning Representations (ICLR)*.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. *Advances in Neural Information Processing Systems (NeurIPS)*.

Chen, X.; Hu, Z.; and Sun, Y. 2022. Fuzzy Logic Based Logical Query Answering on Knowledge Graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 3939–3948.

Gupta, M. M.; and Qi, J. 1991. Theory of T-norms and fuzzy inference methods. *Fuzzy sets and systems*, 40(3): 431–450.

Hájek, P. 2013. *Metamathematics of fuzzy logic*, volume 4. Springer Science & Business Media.

Hamilton, W. L.; Bajaj, P.; Zitnik, M.; Jurafsky, D.; and Leskovec, J. 2018. Embedding Logical Queries on Knowledge Graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2030–2041.

Hao, Y.; Cao, X.; Fang, Y.; Xie, X.; and Wang, S. 2020. Inductive Link Prediction for Nodes Having Only Attribute Information. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1209–1215.

Heist, N.; Hertling, S.; Ringler, D.; and Paulheim, H. 2020. Knowledge Graphs on the Web-An Overview. *Knowledge Graphs for eXplainable Artificial Intelligence*, 3–22.

Hogan, A.; Blomqvist, E.; Cochez, M.; d'Amato, C.; Melo, G. d.; Gutierrez, C.; Kirrane, S.; Gayo, J. E. L.; Navigli, R.; Neumaier, S.; et al. 2021. Knowledge graphs. *Synthesis Lectures on Data, Semantics, and Knowledge*, 12(2): 1–257.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, F.-L.; Chen, H.; Xu, G.; Qiu, T.; Ji, F.; Zhang, J.; and Chen, H. 2020. AliMeKG: Domain knowledge graph construction and application in e-commerce. In *ACM International Conference on Information and Knowledge Management (CKIM)*, 2581–2588.

Liu, S.; Grau, B. C.; Horrocks, I.; and Kostylev, E. V. 2021. INDIGO: GNN-Based Inductive Knowledge Graph Completion Using Pair-Wise Encoding. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Noy, N.; Gao, Y.; Jain, A.; Narayanan, A.; Patterson, A.; and Taylor, J. 2019. Industry-scale Knowledge Graphs: Lessons and Challenges: Five diverse technology companies show how it's done. *Queue*, 17(2): 48–75.

Ren, H.; Hu, W.; and Leskovec, J. 2020. Query2box: Reasoning over Knowledge Graphs in Vector Space Using Box Embeddings. In *International Conference on Learning Representations (ICLR)*.

Ren, H.; and Leskovec, J. 2020. Beta embeddings for multi-hop logical reasoning in knowledge graphs. *Advances in Neural Information Processing Systems (NeurIPS)*, 33: 19716–19726.

Rossi, A.; Barbosa, D.; Firmani, D.; Matinata, A.; and Merialdo, P. 2021. Knowledge graph embedding for link prediction: A comparative analysis. *Acm Transactions on Knowledge Discovery from Data (TKDD)*, 15(2): 1–49.

Sun, Z.; Deng, Z.-H.; Nie, J.-Y.; and Tang, J. 2018. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *International Conference on Learning Representations (ICLR)*.

Teru, K.; Denis, E.; and Hamilton, W. 2020. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning (ICML)*, 9448–9457.

Toutanova, K.; and Chen, D. 2015. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, 57–66. Beijing, China: Association for Computational Linguistics.

Trouillon, T.; Welbl, J.; Riedel, S.; Gaussier, É.; and Bouchard, G. 2016. Complex embeddings for simple link prediction. In *International Conference on Machine Learning (ICML)*, 2071–2080. PMLR.

Xiong, W.; Hoang, T.; and Wang, W. Y. 2017. DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning. In *Empirical Methods in Natural Language Processing (EMNLP)*, 564–573.

Xu, D.; Ruan, C.; Korpeoglu, E.; Kumar, S.; and Achan, K. 2020. Product knowledge graph embedding for e-commerce. In *International Conference on Web Search and Data Mining (WSDM)*, 672–680.

Yang, B.; Yih, S. W.-t.; He, X.; Gao, J.; and Deng, L. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *International Conference on Learning Representations (ICLR)*.

Zhang, S.; Tay, Y.; Yao, L.; and Liu, Q. 2019. Quaternion knowledge graph embeddings. *Advances in Neural Information Processing Systems (NeurIPS)*, 32.

Zhang, Z.; Wang, J.; Chen, J.; Ji, S.; and Wu, F. 2021. Cone: Cone embeddings for multi-hop reasoning over knowledge graphs. *Advances in Neural Information Processing Systems (NeurIPS)*.