Active Knowledge Graph Completion

Pouya Ghiasnezhad Omran[1,2], Kerry Taylor[1,3], Sergio Rodriguez Mendez[1,4], and Armin Haller[1,5]

[1] Australian National University
[2] P.G.Omran@anu.edu.au
[3] kerry.taylor@anu.edu.au
[4] Sergio.RodriguezMendez@anu.edu.au
[5] armin.haller@anu.edu.au

Abstract. Knowledge Graphs (KGs) proliferating on the Web are well known to be incomplete. Much research has been proposed for automatic completion, sometimes by rule learning, that is known to scale well. All existing methods learn closed rules. In this paper, we introduce open path (OP) rules and present a novel algorithm, OPRL, for learning OP rules. While CP rules complete a KG by answering given queries, OP rules identify the incompleteness of a KG by generating such queries. For our learning to scale well, we propose a novel, efficient, embedding-based fitness function to estimate the quality of rules. We also introduce a novel, efficient vector computation to formally assess the quality of such rules against a KG. We use adaptations of Freebase, YAGO2, Wikidata, and a synthetic but complete Poker KG to evaluate OPRL. We find that OPRL mines hundreds of accurate rules from massive KGs with up to 8M facts. The learnt OP rules are used to generate queries with precision as high as 98% and recall of 62% on a complete KG, demonstrating the first solution for active knowledge graph completion.

Keywords: Knowledge Graph Completion · Open Path Rule · Knowledge Graph.

1 Introduction

Knowledge Graphs (KGs) are a convenient technology to model and store massive quantities of weakly-structured data. However, their intended scope is usually poorly defined and they fail to record relevant entities, as well as relevant relationships for the entities they do record [12]. The power of KGs arises from a data-first approach. They allow information to be added in a relatively arbitrary manner as structural constraints are few; unlike, for example, relational databases where type, not-null, and key constraints abound to enforce a kind of completeness. Also, KGs are often (semi-)automatically built from unstructured sources such as Wikipedia articles. The methods are prone to asserting some erroneous facts, while missing some others.

Techniques have been developed for knowledge graph completion and rule learning to curate KGs automatically [9]. In these approaches models, often

expressed as logical rules or vector embeddings, are learnt from a given KG. The models are then used for curating tasks including *link prediction* that predict missing facts for extant entities.

Rule learning methods for KGs [14] consider *closed* rules which are used to predict a ground fact that instantiates the triple at the head of the rule. For example, consider a rule defining a relationship between citizenship and the residence of a person, $citizenOf(x, y) \longleftarrow livesIn(x, z) \wedge locatedIn(z, y)$. Using this rule, someone's citizenship can be inferred from facts about a person's city of residence and the nation in which that city is located.

Closed rules enable inference of specific facts that, if true, are missing from the KG. They draw attention to a potential missing fact only if the fully specified fact is able to be inferred by the rule. KG completion is driven by the assumption that the KG *knows what it does not know*. In this paper we consider, for the first time, the problem of rules-based knowledge graph completion in the situation that it *does not know what it does not know*. This problem requires the KG, as it does for people, to look outside its own environment.

We propose learning *open path rules* (OP) from which we infer less constrained, open-ended questions to complete a knowledge graph. Our OP rules provide evidence that information is missing even when there is no evidence for a specific missing fact. The questions inferred from OP rules identify that a new fact is needed when the answer is not known, but also not obvious. The question could be posed to an active user engaged in a curating task or to a Web question-answering engine, where the answer might be found outside the KG. In particular, an answer to the question may well introduce previously unknown entities to the KG, and so to address a previously unstudied direction in knowledge graph completion, that is *missing entities*.

As a beneficial side-effect, our work addresses a long-standing gap in traditional link prediction systems (e.g. [1]), that use the KG to propose new links, but need to be seeded with questions about potential missing links. Conventionally, for evaluation purposes, these questions are generated from test facts that are held-back from the KG in the hope that a high-performing predictor will rediscover the missing fact. However, once a link predictor is deployed over a working KG, test facts cannot be held back, so whence does the question arise? We propose that the questions we infer from our OP rules can be widely used to generate the questions that link predictors need to repair KGs.

The contributions of this paper are as follows. We present a novel method for learning open path rules from a KG. These are Horn rules with a different form to the usual closed path rules that are used for knowledge graph completion tasks. We propose an algorithm for learning these rules, including novel fitness criteria for discarding poor rules early, and efficient vector computation of formal quality criteria. We show that, together with KG sampling, our algorithm is effective over very large KGs. As such, we introduce a first solution to the problem of *active knowledge graph completion* (AKGC), where we aim, instead of suggesting missing facts, to *ask the best questions* to complete a KG.

2 Background

2.1 Rule-Based KG Completion

An *entity* e is an identifier for an object such as a place or a person, and a *fact* (also known as a *link*) is an RDF triple (e, P, e'), which means that the *subject* entity e is related to an *object* entity e' via the binary *predicate* (also known as a *property*), P. Here we write facts in the form $P(e, e')$. A *knowledge graph* (KG) is a pair $K = (E, F)$, where E is a set of entities and F is a set of facts and E is the same as the set of entities that occur in the facts of F.

Rule learning systems deploy different rule languages to express the rules learnt. RLvLR [13] and ScaleKB [4] use so-called *closed path (CP)* rules that are a kind of closed rule with no free variables. Each consists of two parts, a *head* at the front of the arrow and a *body* at the tail. We say the rule is *about* the predicate of the head. The rule forms a closed path, or single unbroken loop of links between the variables. It has the following general form.

$$P_t(x, y) \leftarrow P_1(x, z_1) \wedge P_2(z_1, z_2) \wedge ... \wedge P_n(z_{n-1}, y) \tag{1}$$

We interpret this kind of rules with universal quantification of all variables at the outside, and so we can infer a fact that instantiates the head of the rule by finding an instantiation of the body of the rule in the KG. For example, from the rule $citizenOf(x, y) \leftarrow livesIn(x, z) \wedge locatedIn(z, y)$, if we have the facts in the KG, $livesIn(bronte, canberra)$ and $locatedIn(canberra, australia)$, then we can derive and assert the following new fact in the KG, $citizenOf(bronte, australia)$.

Rules are considered of more use if they generalise well, that is, they explain many facts. To quantify this idea we recall measures *support*, *head coverage* and *standard confidence* that are used in some major approaches to rule learning including [4] and [6].

Definition 1 (satisfies, support). *Let r be a CP rule of the form (1). A pair of entities (e, e')* satisfies *the body of r, denoted $body_r(e, e')$, if there exist entities $e_1, ..., e_{n-1}$ in the KG such that all of $\{P_1(e, e_1), P_2(e_1, e_2), ..., P_n(e_{n-1}, e')\}$ are facts in the KG. Further (e, e')* satisfies *the head of r, denoted $P_t(e, e')$, if $P_t(e, e')$ is a fact in the KG. Then the* support *of r counts the rule instances for which the body and the head are both satisfied in the KG.*

$$supp(r) = |\{(e, e') : body_r(e, e') \text{ and } P_t(e, e')\}|$$

Standard confidence (SC) and *head coverage (HC)* offer standardised measures for comparing rule quality.

Definition 2 (standard confidence, head coverage). *Let $r, e, e', body_r$ be as given in definition 1. Then*

$$SC(r) = \frac{supp(r)}{|\{(e, e') : body_r(e, e')\}|}, HC(r) = \frac{supp(r)}{|\{(e, e') : P_t(e, e')\}|}$$

2.2 Embedding-Based KG Completion

Representation learning methods have been developed to model KGs for tasks such as link prediction, entity resolution, and link-based clustering [9]. Representation learning typically consists of two main steps: (1) to *embed* the entities and predicates of the given KG into a latent space, and (2) to reconstruct the KG based on the obtained embeddings to predict new facts.

The KG is embedded into a low-dimensional vector space of *latent*, unnamed features not present in the KG vocabulary [11, 7, 10, 8, 1, 17, 15, 9]. The plausibility of each fact is defined by a scoring function over the embedded representations of its predicate and entities. Learning and operating on latent representations benefits from the use of unobserved but intrinsic properties of entities and their relations.

RESCAL [11] is a compositional-based embedding learner. It embeds each entity e_i by a vector $\mathbf{e}_i \in \mathbb{R}^d$ and each predicate P_k by a matrix $\mathbf{P}_k \in \mathbb{R}^{d \times d}$ where $\mathbb{R}$ is the set of real numbers and d is an integer (a parameter to the learner specifying the dimensionality of the latent feature space).

RESCAL learns two sets of embeddings, vectors $\{\mathbf{e}_i\}$ and matrices $\{\mathbf{P}_k\}$ by minimizing a loss function defined over the product of the entity and predicate embeddings. RESCAL captures rich interactions amongst entities and predicates because it learns a larger number of parameters than methods which embed the predicates into vectors [10]. We work with RESCAL for its empirical strength when used as a heuristic for mining logical axioms[14]. The embeddings learned by RESCAL are well suited for our novel heuristic function for mining OP rules introduced here.

3 Rules with Free Variables for AKGC

Unlike earlier work in rule mining for KG completion, for our *active* knowledge graph completion task we mine *open path* (OP) rules of the following form:

$$P_t(x, z_0) \leftarrow P_1(z_0, z_1) \wedge P_2(z_1, z_2) \wedge ... \wedge P_n(z_{n-1}, y) \tag{2}$$

Here, P_i is a predicate in the KG and each of $\{x, z_i, y\}$ are variables (x and y are free while the z_is are bound). Unlike CP rules, OP rules do not necessarily form a loop, but instantiations of a CP rule are also an instantiation of an OP rule. From an instantiation of the body of an OP rule, we can not infer a fact, but only a question. For example, the following OP rule, $citizenOf(x, t) \leftarrow livesIn(x, z)$. states that if an entity, x, lives in z, then that entity is citizen of somewhere (t). By instantiating the body of this rule as follows, $livesIn(bronte, canberra)$, we could infer the query, $citizenOf(bronte, ?)$.

To assess the quality of our mined open path rules, we introduce *open path standard confidence (OPSC)* and *open path head coverage (OPHC)* derived from the closed path forms (Definition 2).

Definition 3 (open path: OPsupp, OPSC, OPHC). *Let r be an OP rule of the form (2). Then a pair of entities* (e, e') satisfies *the body of r, denoted*

$body_r(e, e')$, if there exist entities $e_1, ..., e_{n-1}$ in the KG such that $P_1(e, e_1)$, $P_2(e_1, e_2), ..., P_n(e_{n-1}, e')$ are facts in the KG. A pair (e', e) satisfies *the head of r, denoted $P_t(e', e)$, if $P_t(e', e)$ is a fact in the KG. The* open path support*,* open path standard confidence*, and* open path head coverage *of r are given respectively by*

$$OPsupp(r) = |\{e : \exists e', e'' \text{ s.t. } body_r(e, e') \text{ and } P_t(e'', e)\}|$$
$$OPSC(r) = \frac{OPsupp(r)}{|\{e : \exists e' \text{ s.t. } body_r(e, e')\}|}$$
$$OPHC(r) = \frac{OPsupp(r)}{|\{e : \exists e' \text{ s.t. } P_t(e', e)\}|}$$

For example, consider the OP rule, $P_1(x, z_0) \leftarrow P_2(z_0, z_1) \wedge P_3(z_1, y)$. Assume we have 3 entities ($\{e_3, e_4, e_5\}$) which can instantiate z_0 to satisfy both $P_1(x, z_0)$ and $P_2(z_0, z_1) \wedge P_3(z_1, y)$. Assume the number of entities that can instantiate z_0 to satisfy the head part is 5 ($\{e_1, e_2, e_3, e_4, e_5\}$) and the number of entities that can instantiate z_0 to satisfy the body part is 7 ($\{e_3, e_4, e_5, e_6, e_7, e_8, e_9\}$). Hence we have for this rule, $OPsupp = 3$, $OPSC = 3/7$ and $OPHC = 3/5$.

4 OP Rule Learning

Our objective is to mine a KG for high-quality OP rules about a specific target predicate P_t. While we adhere to the architecture of RLvLR [14] that learns CP rules, we propose the following novelties for mining OP rules: (i) a novel fitness function which can estimate the quality of an OP rule based on the embedding representations of its predicates; and (ii) a novel vector computation which allows the system to evaluate the OP rules against a massive KG to compute quality measures, OPSC and OPHC.

Our OP rule miner, OPRL, is summarised in Algorithm 1. It takes user parameters for the maximum length of rules and least acceptable OPSC and OPHC.

First, we reduce the KG size because existing embedding-based methods cannot handle vast KGs. For instance, RESCAL is unable to handle YAGO2 [10]. We use the sampling algorithm, Sampling() proposed in RLvLR [13] to build a reduced KG. This means that embeddings are computed only for entities that are relevant to a target predicate.

We then compute embedding models to construct a fitness function to rapidly estimate a rule's quality, and so significantly improve scalability (see Section 4.1). Embeddings() obtains the embeddings $\mathcal{P}, \mathcal{A}$ for respectively predicates and arguments in the sample KG.

Then, in shortest-first order, we exhaustively generate OP rules for a target predicate P_i and its inverse P_i^{-1} in PathFinding(). The inverse is defined as $\forall e, e' P_i^{-1}(e', e) = P_i(e, e')$. Since the target predicate is fixed, generating an OP rule is reduced to generating a *path* to comprise the body, i.e. a sequence

Algorithm 1 OPRL

Input: a KG K, a target predicate P_t
Parameter: a max rule length l, $MinOPSC$ and $MinOPHC$
Output: a set of OP rules R

$K' :=$ Sampling(K, P)
$(\mathcal{P}, \mathcal{A}) :=$ Embeddings(K')
$R' := \emptyset$
for $2 \leq k \leq l$ **do**
    Add PathFinding$(K', P_t, \mathcal{P}, \mathcal{A}, k)$ to R'
end for
Add IncPathFinding$(K', P_t, \mathcal{P}, \mathcal{A}, k, R')$ to R'
$R :=$ Evaluation(R', K)
return R

of predicates $P'_1, P'_2, \ldots, P'_n$ with required OP rule variable patterns. We apply the proposed fitness function to each rule on generation to rapidly discard poor performers.

In IncPathfinding() we create additional candidate rules by extending some top-ranked candidates. We learn new short OP rules for the rightmost predicate using PathFinding(). If we find a good rule about that predicate then we extend the original rule by appending the new body to the tail, and we keep both rules. We then use a redundancy elimination method to make sure there is no repetition in all the mined rules and then evaluate candidate rules by OPSC and OPHC in Evaluate(). We use efficient matrix and vector multiplication (Section 4.2) that is crucial for scalability.

4.1 Rule Quality Estimation using Embeddings

Since the number of potential rules generated in PathFinding() is enormous, we rapidly filter out candidates of low quality. For this purpose, the quality is estimated by either of two fitness functions; *co-occurrence* or *open path*, both of which are derived from embedding representations. The former uses entity embeddings alone, while the latter incorporates predicate embeddings as well. We use RESCAL [11] to compute both.

Co-occurrence Fitness Function Each instance of an OP rule connects its head and body via a shared entity in place of z_0 in (2), so an OP rule tends to have high OPsupp (and so high OPSC and OPHC) if the entities which satisfy the second argument of P_t has a large intersection with the entities that satisfy the first argument of P_1. When predicate pairs associate similar entities this way, this induces a latent-feature relationship between the predicates that we call *co-occurrence*. For instance, the two predicates $liveIn(e'', e)$ and $isNeighbour(e, e')$ may co-occur because in both cases e is often a city.

Based on this observation, a co-occurrence fitness function for mining CP rules is defined using *argument embeddings* in RLvLR [13] and we adapt it

here. RLvLR also defines a similarity fitness function that is not applicable here because it relies on the head predicate to share a large number of entities with the body in *both* argument positions.

For argument embeddings, each predicate has a subject argument in the first position and an object argument in the second position. Each argument's embedding is a vector obtained by averaging the embeddings of all the entities in the argument position. For entity e we write its embedding vector as $\mathbf{e}$. For predicate P we write its embedding matrix as $\mathbf{P}$, also called a *weight matrix*.

Definition 4 (argument embedding). *Let $K = (E, F)$ be a KG. The* argument embeddings *of the* subject and object arguments *of a predicate P are vectors defined respectively as:*

$$\mathbf{P}^{(1)} = \frac{1}{n} \sum_{e \in S_P} s_e \mathbf{e} \quad \textit{and} \quad \mathbf{P}^{(2)} = \frac{1}{n} \sum_{e \in O_P} o_e \mathbf{e}$$

where n is the number of facts in the KG, S_P and O_P are the sets of entities occurring as subjects and objects of P, respectively (more precisely, $S_P = \{e \mid \exists e' \text{ s.t. } P(e, e') \in F\}$ and $O_P = \{e' \mid \exists e \text{ s.t. } P(e, e') \in F\}$), and s_e and o_e are the numbers of occasions that entity e occurs as a subject and an object of P in K respectively (more precisely, $s_e = |\{e' \text{ s.t. } P(e, e') \in F\}|$ and $o_e = |\{e' \text{ s.t. } P(e', e) \in F\}|$).

The co-occurrence fitness function for CP rules used in RLvLR [13] needs to be modified for the OP case here. In an OP rule of the form (2), the co-occurrences of z_0 as the object argument of P_t and subject argument of P_1, and z_i $(1 \leq i \leq n-1)$ as the object argument of P_i and subject argument of P_{i+1}, motivates us to highly value rules with the properties (where the symbol $\approx$ is read as *similar to*): $\mathbf{P}_t^{(2)} \approx \mathbf{P}_1^{(1)}$, and $\mathbf{P}_i^{(2)} \approx \mathbf{P}_{i+1}^{(1)}$ $(1 \leq i \leq n-1)$. Pairwise local fitness functions are defined accordingly.

Definition 5 (local co-occurrence fitness). *Let r be an OP rule of the form (2). Then*

$$f_{loc}^{(0)}(P_t, P_1) = sim(\mathbf{P}_t^{(2)}, \mathbf{P}_1^{(1)})$$
$$f_{loc}^{(i)}(P_i, P_{i+1}) = sim(\mathbf{P}_i^{(2)}, \mathbf{P}_{i+1}^{(1)}) \text{ for } 1 \leq i \leq n-1$$

where sim is defined by the Frobenius norm, i.e. for two matrices $\mathbf{M}_1$ and $\mathbf{M}_2$,

$$sim(\mathbf{M}_1, \mathbf{M}_2) = exp(-\|\mathbf{M}_1 - \mathbf{M}_2\|_F).$$

Co-occurrence for the whole rule can then be obtained by aggregating the pairwise local occurrences as follows.

Definition 6 (co-occurrence fitness). *Let r be an open path rule of the form (2). Then*

$$f_{coo}(r) = f_{loc}^{(0)}(P_t, P_1) + \sum_{i=1}^{n-1} f_{loc}^{(i)}(P_i, P_{i+1})$$

By its use of argument embeddings built from *entity* embeddings, the co-occurrence captures the weight of connections of sequential entities along the path. Next we introduce an alternative quality estimation function which uses both entity and predicate embeddings, called *open path fitness*, $f_{OP}(.)$.

Open Path Fitness Function An OP rule acts to connect entities satisfying the subject argument of the head predicate, P_t, to entities forming the object argument of the tail predicate, P_n, along a path of entities that satisfy a chain of predicates in the rule. The product of the predicate embeddings along the path acts as a low-dimensional representation of the latent features of a path that connects its endpoints, and therefore represents the overall rule from the perspective of the predicates. However, to anchor the rule, we also need to account for the entities that satisfy the free variables at the endpoints (as does the RESCAL [11] evaluation function for a single predicate). Conveniently, our argument embeddings for the subject argument of P_t and the object argument of P_n, give us what we need by averaging the embeddings of all the entities at the endpoints. Based on this observation, we propose the *open path fitness* function to estimate rule quality.

Definition 7. *Let r be an OP rule of the form (2). Then the* open path fitness *for r is defined by the product*

$$f_{OP}(r) = \mathbf{P}_t^{(1)^T} \mathbf{P}_t \mathbf{P}_1 \mathbf{P}_2 \dots \mathbf{P}_n \mathbf{P}_n^{(2)}$$

There is no clear reason to prefer either of the fitness functions, $f_{OP}(.)$ and $f_{coo}(.)$, over the other, although the first focuses on entities and the second on predicates. Experimentally, we find that they complement each other, and that a hybrid approach is preferable (cf. Table 3).

4.2 Evaluating Potential Rules through Matrices and Vectors

Now we are ready to explain the evaluation method, Evaluation() in Algorithm 1. For efficiency, we first evaluate the candidate rules based on the sampled KG, and select the rules with $OPsupp(r) \geq 1$. These rules may still contain a large number of redundant and low quality rules and so we do a further selection based on the two measures, OPSC and OPHC evaluated over the full KG. We show in the following how to efficiently compute the measures using an *adjacency matrix* representation of the KG.

Let $K = (E, F)$ with $E = \{e_1, \dots, e_n\}$ be the set of all entities and $\mathbb{P} = \{P_1, \dots, P_m\}$ be the set of all predicates in F. Like RESCAL [10], we represent K as a set of square $n \times n$ adjacency matrices by defining the function $\mathtt{A}$. Specifically, the $[i, j]$ element $\mathtt{A}(P_k)[i, j]$ is 1 if the fact $P_k(e_i, e_j)$ is in F; and 0 otherwise. Thus, $\mathtt{A}(P_k)$ is a matrix of binary values and the set $\{\mathtt{A}(P_k) : k \in \{1, \dots, m\}\}$ represents K.

We illustrate the method for computing OPSC and OPHC through an example. Consider the OP rule $r : P_t(x, z_0) \leftarrow P_1(z_0, z_1) \wedge P_2(z_1, y)$. Let $E =$

$\{e_1, e_2, e_3\}$ and

$$F = \{P_1(e_1, e_2), P_1(e_2, e_1), P_1(e_2, e_3), P_1(e_3, e_1), \\ P_2(e_1, e_2), P_2(e_3, e_2), P_2(e_3, e_3), P_t(e_1, e_3)\}$$

The adjacency matrices for the predicates P_1, P_2 and P_t are:

$$\mathtt{A}(P_1): \begin{bmatrix} 0\ 1\ 0 \\ 1\ 0\ 1 \\ 1\ 0\ 0 \end{bmatrix}, \mathtt{A}(P_2): \begin{bmatrix} 0\ 1\ 0 \\ 0\ 0\ 0 \\ 0\ 1\ 1 \end{bmatrix}, \mathtt{A}(P_t): \begin{bmatrix} 0\ 0\ 1 \\ 0\ 0\ 0 \\ 0\ 0\ 0 \end{bmatrix}$$

For OPSC and OPHC (Definition 3) we need to calculate (1) the number of entities that satisfy the head of the rule in the second argument position, i.e. $\#e : \exists e''$ *s.t.* $P_t(e'', e)$, (2) the number of entities that satisfy the body of a rule in the first argument position, i.e. $\#e : \exists e'$ *s.t.* $body_r(e, e')$ and, (3) the number of entities that join the head of a rule to its body i.e., $\#e : \exists e', e''$ *s.t.* $body_r(e, e')$ *and* $P_t(e'', e)$.

For (1) to find distinct es we sum each *column* of the adjacency matrix (corresponding to each value for the second argument) and transpose to obtain the vector $\mathtt{V}^{(2)}(P_t)$. Each non-zero element of this vector indicates a satisfying e and the number of distinct es is given by counting the number of non-zero elements in it. Formally, the satisfying es are $\{e_j : \sum_{i=1}^{n} \mathtt{A}(P_t)[i, j] > 0$ and $1 \leq j \leq n\}$ and the cardinality is the number we need.

For the example, we have the only non-zero element in $\mathtt{A}(P_t)$ is $\mathtt{A}(P_t)[1, 3]$ and after summing the columns we have $\mathtt{V}^{(2)}(P_t)^T = (0, 0, 1)$ so we have only $\mathtt{V}^{(2)}(P_t)[3]$ is non-zero and $\{e_3\}$ satisfies the head with count 1.

For (2) the pairs (e, e') satisfying the body are connected by the path $P_1, P_2, \ldots P_m$, and can be obtained directly from the matrix product $B = \mathtt{A}(P_1)\mathtt{A}(P_2) \ldots \mathtt{A}(P_m)$, being the elements with a non-zero value. To find distinct es we sum each *row* (corresponding to each value for the first argument) to obtain the vector $\mathtt{V}^{(1)}(B)$. Each non-zero element of this vector indicates a satisfying e and the number of distinct es is given by counting the number of non-zero elements in $\mathtt{V}^{(1)}B$. Formally, let $B = \mathtt{A}(P_1)\mathtt{A}(P_2) \ldots \mathtt{A}(P_m)$. Then the satisfying es are given by $\{e_i : \sum_{j=1}^{n} B[i, j] > 0$ and $1 \leq i \leq n\}$

For the example we have

$$B = \mathtt{A}(P_1)\mathtt{A}(P_2) = \begin{bmatrix} 0\ 0\ 0 \\ 0\ 2\ 1 \\ 0\ 1\ 0 \end{bmatrix}, \mathtt{V}^{(1)}(B) = \begin{bmatrix} 0 \\ 3 \\ 1 \end{bmatrix}$$

with satisfying entities e_2 and e_3 and count of 2.

Computing (3) is now straightforward. We have that the row index of non-zero elements of $\mathtt{V}^{(2)}(P_t)$ indicate entities that satisfy the second argument of the head and that the row index of non-zero elements of $\mathtt{V}^{(1)}(B)$ indicate entities that satisfy the first argument of the body. Therefore we can find the entities that satisfy both of these conditions by pairwise multiplication. That is, the entities we need are $\{e_i : (\mathtt{V}^{(2)}(P_t)[i] \times \mathtt{V}^{(1)}(\mathtt{A}(P_1)\mathtt{A}(P_2) \ldots \mathtt{A}(P_m))[i]) > 0$ and

$1 \leq i \leq n\}$, and the count is the cardinality of this set. For the example we have only e_3 in the set with count 1.

Hence, $OPsupp(r) = 1$. From (1), (2) we can easily obtain $OPHC(r) = 1/1$ and $OPSC(r) = 1/2$.

Minimum thresholds for OPSC and OPHC are supplied to Algorithm 1 at run-time and Evaluation() discards failing rules. The remaining rules are the final result of the algorithm.

In summary, we introduce the following novel components to mine OP rules from KGs:

- Proposing OP rules: we proposed a fragment of Horn rules which allows us to mine rules with free variables while keeping the complexity of the learning phase manageable.
- Learning OP rules based on an embedding representation: we introduced a novel method to estimate the feasibility of each candidate rule based on its embedding representation.
- Evaluating OP rules: we proposed an efficient method to exactly compute the quality of each rule by matrix and vector operations.

5 Inferring relevant queries for AKGC

Given predicate P, the AKGC task is to predict *queries* of the form $P(?, e')$ and $P(e, ?)$ for entities e and e' occurring in the KG. To find *relevant* queries, we implement an *inference module* that derives queries from KG facts together with OP rules found by Algorithm 1.

For an OP rule of the form (2), if an instance of the rule body such as $P_1(e, e_1), P_2(e_1, e_2), \ldots, P_n(e_{n-1}, e')$ exists in the KG, then the existence of an instance of the head with one free variable, viz the query $P_t(?, e)$, can be inferred with a quantifiable confidence.

We define the confidence degree (CD) of a query to be the maximum SC of all the rules inducing the query, thereby giving no weight to redundant rules inducing the same query. Formally, for a query $f = P(?, e')$ or $f = P(e, ?)$ and the set of rules R that can infer f from the given KG, the CD of f is defined:

$$CD(f) = \max_{r \in R}(OPSC(r))$$

In this way we go beyond link prediction to infer relevant queries for missing links, AKGC. Typically, a link predictor is given a query derived from the holdout test data to predict facts, and then uses test data for evaluating the predicted facts. This begs the question, whence the query arises in an industrial application of link prediction? You have a KG and a link-predicting model built for the KG, but do you hold-out facts from your KG in order to generate queries that predict those same facts: facts that are missing only because you need them to generate queries? For AKGC, we need only a named predicate (or all predicates) and use OP rules mined over training data to induce queries over the full KB.

Our proposed AKGC is not an alternative to KGC. However, it is a complementary front-end step and allows us to ask relevant questions via OP rules and to answer them via a link predictor or another way.

6 Experiments

We have conducted two sets of experiments to evaluate OPRL[6], demonstrating:

1. OPRL can mine quality OP rules from a range of KGs. OPRL can mine massive KGs in reasonable time. Our novel hybrid fitness function outperforms the fitness function adapted from RLvLR [13].
2. Queries generated from OPRL's rules are relevant with good recall and precision in multiple KGs. They far outperform a distribution-based baseline.

The four benchmark datasets are given in Table 1. FB15K SELECTED (which we call *FB15KSE*) is derived from Freebase and is widely adopted for link prediction [18]. YAGO2 core is often used for rule mining [6, 4]. Wikidata [16] is a multilingual, collaboratively-created KG to manage the factual information of a popular online encyclopedia; we use a copy dated December 2014 provided in AMIE+ [6]. Poker is a synthetic dataset adapted by the authors from the classic version [3, 5] to be a correct and complete KG for experiments. Each hand consists of 5 playing cards drawn from a reduced deck with 6 ranks and 2 suits.

All experiments were conducted on an Intel Xeon CPU E5-4620v2 @ 2.60GHz, 66GB RAM and running CentOS 7.

For sampling, we use similar parameters to those proposed in RLvLR [13]. We set the maximum size of each sample to 800 entities. We use RESCAL [9] to generate embeddings with the vector size set to 100. We retain the top 10% of the OP rules according to the fitness function. The number of possible rules grows significantly with increasing length, as does the runtime for mining. We use a maximum rule length of 4 for PathFinding() and we allow the extension to 6 by IncPathFinding(). These parameters are the optimum obtained by tuning.

6.1 OP Rule Learning

First, we assess how well OPRL finds high quality rules. We are not aware of other OP rule learners with which to compare, but we do compare the performance of fitness functions. The quality of rules are reported based on their OPSC/OPHC scores calculated against the full benchmark KGs, not the samples.

Later we will use the mined rules for generating queries, so we need some holdout facts for query evaluation. For FB15KSE, test and training sets are available [18]. For Poker and YAGO2 core we can find no previously prepared data, so we randomly partition 90% for training and 10% for testing.

Table 2 shows the average numbers of quality rules mined for all predicates (except for Wikidata for which we target 50 randomly selected predicates) and

[6] The datasets used in the experiments and detailed results can be found at www.dropbox.com/sh/y1f7zut09dheius/AADofv9c18Rzm-CFc64dw2yVa?dl=0

Table 1: Benchmark KG specifications

KG	# Facts	# Entities	# Predicates
FB15KSE	272K	15K	237
YAGO2 core	948K	470K	32
Poker	1M	95k	27
Wikidata	8.4M	4M	430

Table 2: Performance of OPRL on benchmark KGs

Benchmark	#Rules	#Arules	Time (hours)
FB15KSE	1029	261	0.17
YAGO2 core	84	9	0.20
Poker	603	509	0.52
Wikidata	175	56	1.76

the running times (in hours, averaged over the targets). Similarly to [6], only rules with quality OPSC$\geq$ 0.1 and OPHC$\geq$ 0.01 are included. The average number of accurate rules, i.e. the rules with OPSC$\geq$ 0.8, are given as #Arules. Figure 1 shows the distribution of mined rules by OPSC and length. We can see that OPRL can learn plausible rules over popular benchmark KGs of over eight million facts and four million entities in less than two hours.

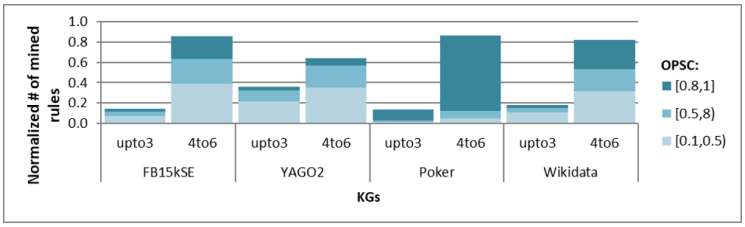

Fig. 1: The length and quality of mined rules.

For illustration we present the following two OP rules which are mined from Wikidata, with their OPSC and OPHC values respectively. The first rule states if there is a region (z) that its country (w) is known to the KG and the continent of its country (y) is also known, it is likely that its highest point (x) is known. The second rule states if the spouse of a person (z) is known, it is likely that the place of birth of that person is also known. Thus, if the body of each rule is instantiated but there is no fact to instantiate the head of the rule, a relevant query is generated.

$$0.19, 0.45\ \mathsf{highestPoint}(z, x) \leftarrow \mathsf{country}^{-1}(z, w) \wedge \mathsf{continent}(w, y).$$
$$0.50, 0.02\ \mathsf{placeOfBirth}(z, x) \leftarrow \mathsf{spouse}(z, y).$$

We conducted an experiment to assess the utility of our fitness functions, using random.org to select 20 random predicates from FB15KSE. The results summarised in Table 3 show that a hybrid fitness function that combines both $f_{coo}(.)$ and $f_{OP}(.)$ is capable of mining more quality rules (defined as for Table 2) than either of these functions individually.

Table 3: Comparison of Fitness Functions

Fitness Function	#Rules	#Arules
f_{coo}, co-occurrence	843	223
f_{OP}, open path fitness	796	193
OPRL, f_{coo} & f_{OP}	**1109**	**296**

Table 4: Accuracy of query generation, comparing OPRL with a random query generator

Benchmark	#Q	OPRL			*Prand*		
		P	R	F1	P	R	F1
FB15KSE	15k	**0.13**	**0.3**	**0.18**	0.02	0.05	0.03
YAGO2 core	9k	**0.14**	**0.01**	**0.03**	0.06	0.005	0.01
Poker	41k	**0.98**	**0.62**	**0.76**	0.17	0.07	0.1

6.2 Query Generation for Active Knowledge Graph Completion

Our second set of experiments assesses the relevance of queries induced from OPRL-generated rules. For this purpose we consider that a query with an answer present in the test set is a *relevant* query, having previously filtered out queries that can be answered from the training set.

In the absence of any comparative system for query generation, we developed three baseline query sets of the same cardinality as those generated from OPRL, *Prand*. *Prand* queries are generated by first selecting a bag of predicates, with each selected randomly with probability of its proportion in the test set. Then for half of the instances of each predicate, a subject (respectively object) entity is assigned by random selection of an entity with probability of its proportion as a subject (respectively object) of any predicate in the test set. The object (respectively subject) position is free (denoted "?").

We assess relevance over three KGs; but not Wikidata because there are no public test and train sets, and *Prand* cannot handle the massive size.

Table 4 shows average precision, recall and F_1, where a query is counted true if it has an instance in the test data, and false otherwise. The queries were induced by OPRL rules learnt over the training data with quality thresholds OPSC$\geq$ 0.8 and OPHC$\geq$ 0.01. We see that OPRL's performance exceeds *Prand* on FB15KSE, YAGO2 core and Poker by factors of approximately 6, 2 and 9 respectively. We suspect that YAGO2 induces fewer rules and has much weaker performance because it has significantly fewer repeatable patterns. This could be because it is quite correctly weakly structured, or because it has significantly more missing facts. If the latter, then the validity of the test set is questionable because genuinely missing facts will be treated as false instead of true, thereby incorrectly punishing precision and recall compared to their values on a complete test set. Supporting this explanation, we see that for synthetic Poker, which is naturally highly structured, and where all the missing facts are present in the test set, we have very close to 100% precision and excellent recall. Very high precision means our queries are very highly relevant as they ask questions for

which the answer facts are missing from the training set. Still, even for Poker, recall shows that there are relevant queries that were not generated, possibly due to the limitations of our OP language or to useful rules being discarded by the fitting function or the OPSC/OPHC thresholds.

Next we consider the sensitivity of the performance of OPRL queries to the OPSC threshold by varying it from 0.1 to 1, learning the previously selected predicates for FB15KSE (Table 3). In Figure 2, we see that increasing OPSC has the expected, healthy, behaviour of decreasing recall as poorer rules get through, and increasing precision as more missing facts are found. Observing a sharp anomaly where the OPSC threshold is 0.9, we suspect it might be caused by the incompleteness of FB15KSE (i.e. the generated queries may be valid but there is no answer in the test set). We repeated the experiment on the complete Poker KG and the anomaly is indeed absent as expected.

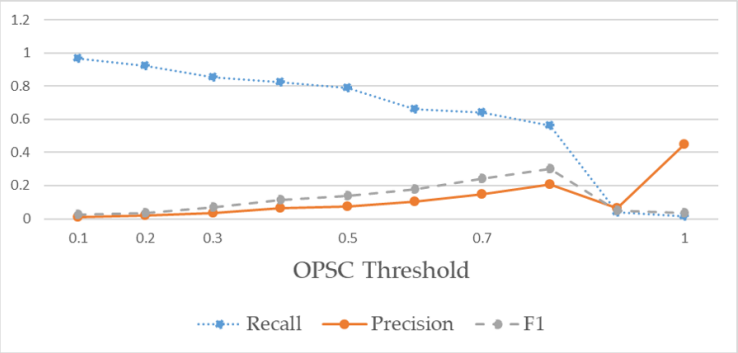

Fig. 2: Sensitivity analysis of OPRL on 20 predicates of FB15KSE

Finally, we compare OPRL queries with state-of-the-art CP-learning link predictor, RLvLR [13]. CP learners predict facts, not queries, so we cannot compare them directly. Instead, we translate each fact generated by RLvLR to two queries, by freeing the subject and object entities respectively. However, while generating facts, RLvLR uses a Noisy-OR operator to aggregate high-performing rules about a target predicate. This aggregation is not compatible with the query translation task where only the top prediction matters. Consequently, experiments show that RLvLR with Noisy-OR performs very poorly for query generation. To more fairly compare, we changed the aggregation in RLvLR to use a Max operator instead of the Noisy-OR, and we call this RLvLR*. We used 20 randomly selected predicates from FB15KSE in the query generation task. We plot the query performance of OPRL and RLvLR as ROC (Receiver Operating Characteristic) curves in Figure 3 using respectively minimum OPSC and minimum SC parameters as the ordering criteria.

While for OPRL we vary the minimum OPSC from 1 to 0.5 in 4 decrements and get a False Positive Rate of almost 50%, for RLvLR we vary SC from

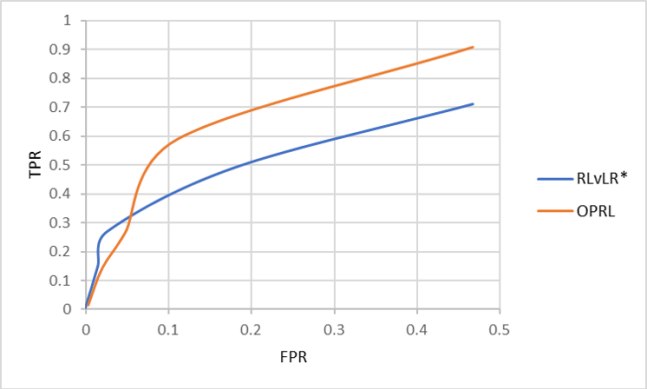

Fig. 3: ROC curves for OPRL and RLvLR, showing True Positive Rate vs False Positive Rate while varying OPSC and SC respectively

1 to 0.1 (the minimum and default) in 4 decrements to achieve the same False Positive Rate. Since SC and OPSC are similar measures, we say OPRL has higher confidence in its queries than RLvLR does. In Figure 3 the partial area under the curve (AUC) of RLvLR* is 0.23 while the partial AUC of OPRL is 0.30, showing OPRL outperforms RLvLR* on query generation by 30%.

7 Related Work and Conclusion

We are unaware of any previous work that produces relevant queries for link predictors in the knowledge graph completion process. However, CHAI [2] filters facts before supplying them to a link predictor, aiming to improve the prediction by focusing its attention on the more probable facts and discarding irrelevant ones. It does not generate queries as OPRL does, and is highly dependent on the initial set of facts which it aims to filter.

In this paper, we proposed a method for learning rules with free variables from Knowledge Graphs (KGs). Such rules can be used to generate queries soliciting missing facts. Notably, the queries could be fed to link predictors, so obtaining a fully automated framework for KG completion. We introduced the following novel components (1) we proposed OP rules, a fragment of Horn rules, which allows us to mine rules with free variables while keeping the complexity of the learning phase manageable; (2) we introduced a novel method to estimate the feasibility of each candidate rule based on its embedding representation; (3) we proposed an efficient method to evaluate OP rules by exactly computing the quality of each rule by matrix and vector operations; and (4) we showed how OP rules can be used to generate highly relevant queries for missing links, introducing the first work on active knowledge graph completion.

Our experiments show that OPRL can learn rules for KGs with varying sizes and degrees of incompleteness. We show the usefulness of the mined rules by applying them to three different KGs to infer relevant queries.

There remain some intriguing challenges for future work. We plan to extend OPRL to learn rules with more complex shapes such as stars, and with numerical attributes. We plan to pair OPRL with a link predictor to form a unified framework for fully automated KG completion. We will also trial OPRL in a setting for human-curated maintenance on an enterprise KG.

References

1. Bordes, A., Usunier, N., Weston, J., Yakhnenko, O., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in neural information processing systems. vol. 26, pp. 2787–2795 (2013)
2. Borrego, A., Ayala, D., Hernández, I., Rivero, C.R., Ruiz, D.: Generating Rules to Filter Candidate Triples for their Correctness Checking by Knowledge Graph Completion Techniques ACM Reference Format. In: K-Cap. vol. 19. ACM (2019)
3. Cattral, R., Oppacher, F., Deugo, D.: Evolutionary Data Mining with Automatic Rule Generalization., pp. 296–300. WSEAS Press (2002)
4. Chen, Y., Wang, D.Z., Goldberg, S.: ScaLeKB: scalable learning and inference over large knowledge bases. The International Journal on Very Large Data Bases pp. 893–918 (2016)
5. Dua, D., Graff, C.: UCI machine learning repository (2017), archive.ics.uci.edu/ml Retrieved Nov 2019
6. Galárraga, L., Teflioudi, C., Hose, K., Suchanek, F.M.: Fast rule mining in ontological knowledge bases with AMIE+. The International Journal on Very Large Data Bases pp. 707–730 (2015)
7. Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning Entity and Relation Embeddings for Knowledge Graph Completion. In: AAAI. pp. 2181–2187 (2015)
8. Liu, H., Wu, Y., Yang, Y.: Analogical Inference for Multi-Relational Embeddings. In: International Conference on Machine Learning (2017)
9. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A Review of Relational Machine Learning for Knowledge Graphs. In: IEEE. vol. 104, pp. 11–33 (2016)
10. Nickel, M., Rosasco, L., Poggio, T.: Holographic Embeddings of Knowledge Graphs. In: AAAI. pp. 1955–1961 (2016)
11. Nickel, M., Tresp, V., Kriegel, H.P.: A three-way model for collective learning on multi-relational data. In: International Conference on Machine Learning. pp. 809–816 (2011)
12. Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., Taylor, J.: Industry-scale knowledge graphs: Lessons and challenges. ACM Queue **17** (2019)
13. Omran, P.G., Wang, K., Wang, Z.: Scalable Rule Learning via Learning Representation. In: IJCAI. pp. 2149–2155 (jul 2018)
14. Omran, P.G., Wang, K., Wang, Z.: An Embedding-based Approach to Rule Learning in Knowledge Graphs. IEEE Transactions on Knowledge and Data Engineering pp. 1–1 (2019)
15. Shen, Y., Huang, P.S., Chang, M.W., Gao, J.: Link Prediction using Embedded Knowledge Graphs. arXiv preprint **arXiv:1611.04642** (2018)
16. Vrandečić, D., Krötzsch, M.: Wikidata: A Free Collaborative Knowledge Base. Communications of the ACM pp. 78–85 (2014)
17. Yang, B., Yih, W.t., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. In: International Conference on Learning Representations. p. 12 (2015)

18. Yang, F., Yang, Z., Cohen, W.W.: Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In: Neural Information Processing Systems (2017)