Rule Induction in Knowledge Graphs Using Linear Programming

Sanjeeb Dash, João Gonçalves

IBM Research, Yorktown Heights, New York, USA
{sanjeebd, jpgoncal}@us.ibm.com

Abstract

We present a simple linear programming (LP) based method to learn compact and interpretable sets of rules encoding the facts in a knowledge graph (KG) and use these rules to solve the KG completion problem. Our LP model chooses a set of rules of bounded complexity from a list of candidate first-order logic rules and assigns weights to them. The complexity bound is enforced via explicit constraints. We combine simple rule generation heuristics with our rule selection LP to obtain predictions with accuracy comparable to state-of-the-art codes, even while generating much more compact rule sets. Furthermore, when we take as input rules generated by other codes, we often improve interpretability by reducing the number of chosen rules, while maintaining accuracy.

Introduction

Knowledge graphs (KG) represent a collection of known facts via labeled directed edges. A *fact* is a triplet of the form (a, r, b), where a and b are nodes representing *entities*, and r is a binary relation labeling a directed edge from a to b indicating that $r(a, b)$ is true. Practical knowledge graphs are often incomplete (they do not contain all true representable facts). Knowledge graph completion (KGC) involves using known facts in a KG to infer additional (missing) facts. Other common tasks for extracting implied information from KGs are triple classification, entity recognition, and relation prediction. See the survey by Ji et al. (2021).

One approach for KGC is to learn first-order logic (FOL) rules that approximately encode known facts (in practice, KGs have inconsistencies). Consider a KG where nodes represent individuals and the relations are one of {*son_of*, *father_of*, *grandson_of*}. Suppose we learn a "rule" *(X, son_of, Y)* and *(Y, son_of, Z)* $\rightarrow$ *(X, grandson_of, Z)* of *length* two, where *X, Y, Z* are variables. If there are entities *P, Q, R* in the KG such that *(P, son_of, R)* and *(R, son_of, Q)* are facts in the KG then we infer that *P* is a grandson of *Q*. KGC deals with finding answers to queries of the form *(P, grandson_of, ?)*. We learn multiple FOL rules of the type above for this task along with rule weights, where weights indicate importance.

Learning logic rules is a well-studied area. In a paper (Lao and Cohen 2010) on path ranking and another (Richardson and Domingos 2006) on Markov logic networks, candidate logic rules are obtained via relational path enumeration. Yang, Yang, and Cohen (2017) use neural logic programming to simultaneously obtain rules and rule weights. In (Qu et al. 2021), rules and rule weights are computed in sequence, but there is a feedback loop from the latter learning problem to the former. Recursive neural networks (RNN) are widely used to learn rules, though rule-mining approaches can be very effective (Meilicke et al. 2019).

Embedding-based methods for KGC consist of representing nodes by vectors, and relations by vector transformations that are consistent with the KG facts. They exhibit better scaling with KG size, and yield more accurate predictions. See the surveys by Ji et al. (2021) and Wang et al. (2017a). However, they are less interpretable than logical rules.

Besides predictive accuracy, Michie (1988) identified *comprehensibility* as an important property of a machine learning (ML) method and proposed the *strong criterion* and *ultra-strong criterion*. The former is satisfied if the ML system generates an explicit symbolic representation of its hypotheses, and the latter is satisfied if humans can understand and effectively apply such symbolic representations. Interpretable ML systems form an important current research area; see (Rudin et al. 2022) and (Kovalerchuk, Ahmad, and Teredesai 2021). In socially sensitive applications of ML, e.g., in the criminal justice system, interpretability is as an essential feature to allow audits for fairness and unbiasedness (Mehrabi et al. 2022). Experimental results in (Schmid et al. 2017; Muggleton et al. 2018) show that for logic programs *inspection time* (the time taken by humans to study and understand the program before applying it) is negatively correlated with human predictive performance. Many rule-based methods satisfy Michie's strong criterion but generate a lot of rules leading to long inspection times and low comprehensibility and interpretability. The tradeoff between accuracy and rule simplicity (or compactness) is well-studied in the setting of classification; it is shown in (Dash, Günlük, and Wei 2018; Wang et al. 2017b) that one can obtain interpretable rule sets without significantly sacrificing accuracy.

Inspired by these classification methods, we focus on learning compact sets of entity independent FOL rules for KGC (as KGC has *no negatively* labeled examples, one cannot apply classification techniques directly). We combine rule enumeration with linear programming (LP) and avoid solving difficult nonconvex optimizaton models inherent in training RNNs, though the difficulty is transferred to rule enumeration. We describe an LP formulation with one variable per candidate FOL rule and its associated weight. Nonzero variable values in the solution correspond to the weights of chosen rules. The goal of this LP formulation is to return a scoring function that is a linear combination of rule truth values and gives high scores to known facts and low scores to (a sample of) missing facts. Linear combinations are also used in NeuralLP (Yang, Yang, and Cohen 2017), DRUM (Sadeghian et al. 2019), and RNNLogic (Qu et al. 2021) though we calculate scores differently.

To promote interpretability, we add a constraint limiting the complexity of the chosen rule set. We show that initializing our LP with rules generated via simple heuristics leads to high-quality solutions with a small number of chosen rules for some benchmark KGs; we either obtain better accuracy or more compact rule sets in all cases compared to well-known rule-based methods. Our algorithm has better scaling with KG size than the above rule-based methods and can scale to YAGO3-10, a large dataset that is difficult for many rule-based methods (though AnyBURL (Meilicke et al. 2019) is faster). For YAGO3-10, where setting up the LP becomes expensive, we use column generation ideas from linear optimization and start off with a small initial set of rules, find the best subset of these and associated weights via the partial LP defined on these rules, and then generate new rules which can best augment the existing set of rules.

We explore taking as input rules generated by other rule-based methods and then choosing the best subset using our LP. In some cases, we improve accuracy even while reducing the number of rules. As compact rules do well in an inductive setting, we compare against GraIL (Teru, Denis, and Hamilton 2020) – a method that uses subgraph reasoning for KGC – in such a setting and obtain better performance.

Related Work

A motivation for learning rules for KG reasoning is that they form an explicit symbolic representation of the KG and are amenable to inspection when there are few rules.

Inductive Logic Programming (ILP). In this approach, one takes as input positive and negative examples and learns logic programs that entail all positive examples and none of the negative examples. See (Cropper and Muggleton 2016) and (Cropper and Morel 2021). FOL programs in the form of a collection of chain-like Horn clauses are a popular output format. Negative examples are not available in typical knowledge graphs, and some of the positive examples can be mutually inconsistent. Evans and Grefenstette (2018) developed a differential ILP framework for noisy data.

Statistical Relational Learning (SRL). SRL aims to learn FOL formulas from data and to quantify their uncertainty. Markov logic, which is a probabilistic extension of FOL, is a popular framework for SRL. In this framework, one learns a set of weighted FOL formulas. Kok and Domingos (2005) use beam search to find a set of FOL rules, and learn rule weights via standard numerical methods. In knowledge graph reasoning, *chain-like* rules which correspond to relational paths (and to chain-like Horn clauses) are widely studied. A recent, bottom-up rule-learning algorithm with excellent predictive performance is AnyBURL. We learn FOL rules corresponding to relational paths and rule weights; our scoring function and learning model/algorithm are different from prior work.

Neuro-symbolic methods. In NeuralLP rules and rule weights are learned simultaneously by training an appropriate RNN. Further improvements can be found in DRUM. NTP (Rochstätel and Riedel 2017) is another neuro-symbolic method. More general rules (than the chain-like rules in NeuralLP) are obtained in (Yang and Song 2020), (Sen et al. 2022b), (Sen et al. 2022a) along with better scaling behavior. Simultaneously solving for rules and rule-weights is difficult, and a natural question is how well the associated optimization problem can be solved. We use an easier-to-solve LP formulation.

Reinforcement Learning (RL). Some recent codes that use RL to search for rules are MINERVA (Das et al. 2018), MultiHopKG (Lin, Socher, and Xiong 2018), M-Walk (Shen et al. 2018) and DeepPath (Xiong, Hoang, and Wang 2017). The first three papers use RL to explore relational paths conditioned on a specific query, and use RNNs to encode and construct a graph-walking agent.

Rule types/Rule combinations. NLIL (Yang and Song 2020) goes beyond simple chain-like rules. Subgraphs are used in (Teru, Denis, and Hamilton 2020) to perform reasoning, and not just paths. We generate *weighted, entity-independent, chain-like rules* as in NeuralLP. Our scoring function combines rule scores via a linear combination. For a rule r and a pair of entities a, b, the rule score is just 1 if there exists a relational path from a to b following the rule r and 0 otherwise. We use rule weights as a measure of importance (but not as probabilities). For Neural LP, DRUM, RNNLogic and other comparable codes, the scoring functions depend on the set of paths from a to b associated with r. AnyBURL uses maximum confidence scores.

Scalability/Compact Rule sets. As noted above, many recent papers use RNNs in the process of finding chain-like rules and this can lead to expensive computation times. On the other hand, bottom-up rule-learners such as AnyBURL are much faster. The main focus of our work is obtaining compact rule sets for the sake of interpretability while maintaining scalability via LP models and column generation. NeuralLP usually returns compact rule sets while AnyBURL returns a large number of rules (and does not prune discovered rules for interpretability), and RNNLogic is somewhere in between (the number of output rules can be controlled).

Model

We propose an LP model inspired by LP boosting methods for classification using classical *column generation* techniques (Demiriz, Bennett, and Shawe-Taylor 2002; Eckstein and Goldberg 2012; Eckstein, Kagawa, and Goldberg 2019; Dash, Günlük, and Wei 2018). Our goal is to create a weighted linear combination of first-order logic rules to be used as a scoring function for KGC. In principle, our model has exponentially many variables corresponding to the possible rules. In practice, we initialize the LP with few initial candidate rules. If the solution is satisfactory, we stop, otherwise we use column generation and generate additional rules that can improve the overall solution.

Knowledge graphs: Let V be a set of entities, and let $\mathcal{R}$ be a set of n binary relations defined over $V \times V$. A *knowledge graph* represents a set of *facts* $\mathcal{F} \subseteq V \times \mathcal{R} \times V$ as a labeled, directed multigraph $\mathcal{G}$. Let $\mathcal{F} = \{(t^i, r^i, h^i) : i = 1, \dots |\mathcal{F}|\}$ where $t^i \neq h^i \in V$, and $r^i \in \mathcal{R}$. The nodes of $\mathcal{G}$ correspond to entities in $\mathcal{F}$ and the edges to facts in $\mathcal{F}$: a fact (t, r, h) in $\mathcal{F}$ corresponds to the directed edge (t, h) in $\mathcal{G}$ labeled by the relation r, depicted as $t \xrightarrow{r} h$. Here t is the *tail* of the directed edge, and h is the *head*. Let E stand for the list of directed edges in $\mathcal{G}$. In practical KGs missing facts that can be defined over V and $\mathcal{R}$ are not assumed to be incorrect. The *knowledge graph completion* task consists of taking a KG as input and answering a list of queries of the form $(t, r, ?)$ and $(?, r, h)$, constructed from facts (t, r, h) in a test set. The query $(t, r, ?)$ asks for a head entity h such that (t, r, h) is a fact, given a tail entity t and a relation r. A collection of facts $\mathcal{F}$ is divided into a training set $\mathcal{F}_{tr}$, a validation set $\mathcal{F}_v$, and a test set $\mathcal{F}_{te}$, the KG $\mathcal{G}$ corresponding to $\mathcal{F}_{tr}$ is constructed and a scoring function is learnt from $\mathcal{G}$ and evaluated on the test set.

Goal: For each relation r in $\mathcal{G}$, we wish to learn a scoring function $f : (t, r, h) \to \mathbb{R}$ that returns high scores for true facts (in $\mathcal{F}$) and low scores for facts not in $\mathcal{F}$. To do this, we find a set of *closed, chain-like* rules $R_1, \dots, R_p$ and positive weights $w_1, \dots w_p$ where each rule R_i has the form

$$r_1(X, X_1) \wedge r_2(X_1, X_2) \wedge \dots \wedge r_l(X_{l-1}, Y) \to r(X, Y). \tag{1}$$

Here $r_1, \dots, r_l$ are relations in $\mathcal{G}$, and the *length* of the rule is l. The interpretation of this rule is that if for some entities (or nodes) X, Y of $\mathcal{G}$ there exist entities $X_1, \dots X_l$ of $\mathcal{G}$ such that $r_1(X, X_1), r_l(X_{l-1}, Y)$ and $r_j(X_{j-1}, X_j)$ are true for $j = 2, \dots l-1$, then $r(X, Y)$ is true. We refer to the conjunction of relations in (1) as the clause associated with the rule R_i. Thus each clause C_i is a function from $V \times V$ to $\{0, 1\}$, and we define $|C_i|$ to be the number of relations in C_i. Clearly, $C_i(X, Y) = 1$ for entities X, Y in $\mathcal{G}$ if and only if there is a *relational path* of the form

$$X \xrightarrow{r_1} X_1 \cdots X_{l-1} \xrightarrow{r_l} Y.$$

Our learned scoring function for relation r is simply

$$f_r(X, Y) = \sum_{i=1}^{p} w_i C_i(X, Y) \text{ for all } X, Y \in V. \tag{2}$$

We learn weights of the linear scoring function above by solving an LP that rewards high scores (close to 1) to training facts and penalizes positive scores given to missing facts. Given a query $(t, r, ?)$ constructed from a fact (t, r, h) from the test set, we calculate $f_r(t, v)$ for every entity $v \in V$, and the *rank* of the correct entity h is the position of h in the sorted list of all entities v (sorted by decreasing value of $f_r(t, v)$). We similarly calculate the rank of t for the query $(?, r, h)$. We then compute standard metrics such as MRR (mean reciprocal rank), Hits@1, and Hits@10 (Sun et al. 2020) in the filtered setting (Bordes et al. 2013). An issue in rank computation is that multiple entities (say e' and e'') can get the same score for a query and different treatment of equal scores can lead to very different MRR values. In *optimistic* ranking, an entity is given a rank equal to one plus the number of entities with strictly larger score. We use *random break* ranking (an option available in NeuralLP), where ties in scores are broken randomly.

New LP model for rule learning for KGC. Let $\mathcal{K}$ denote the set of clauses of possible rules of the form (1) with maximum rule length L. Clearly, $|\mathcal{K}| = n^L$, where n is the number of relations. Let E_r be the set of edges in $\mathcal{G}$ labeled by relation r, and assume that $|E_r| = m$. Let the ith edge in E_r be (X_i, Y_i). We compute a_{ik} as $a_{ik} = C_k(X_i, Y_i)$: a_{ik} is 1 if and only if there is a relational path associated with the clause C_k from X_i to Y_i. Furthermore, let neg_k be a number associated with the number of "nonedges" (X', Y') from $(V \times V) \setminus E_r$ for which $C_k(X', Y') = 1$. We calculate neg_k for the kth rule as follows. We consider the tail node t and head node h for each edge in E_r. We compute the set of nodes S that can be reached by a path induced by the kth rule starting at the tail. If there is no edge from t to a node v in S labeled by r, we say that v is an invalid end-point. Let right_k be the set of such invalid points. We similarly calculate the set left_k of invalid start-points based on paths ending at h induced by the kth rule. The total number of invalid start and end points for all tail and head nodes associated with edges in E_r is $\text{neg}_k = |\text{right}_k| + |\text{left}_k|$. For a query of the form $(t, r, ?)$ where t is a tail node of an edge in E_r, the scoring function defined by the kth rule alone gives a positive and equal score to all nodes in right_k.

Our model for rule-learning is given below.

(LPR)

$$z_{min} = \quad \min \quad \sum_{i=1}^{m} \eta_i + \tau \sum_{k \in \mathcal{K}} \text{neg}_k w_k \tag{3}$$

$$s.t. \quad \sum_{k \in \mathcal{K}} a_{ik} w_k + \eta_i \geq 1 \text{ for all } i \in E_r \tag{4}$$

$$\sum_{k \in \mathcal{K}} (1 + |C_k|) w_k \leq \kappa \tag{5}$$

$$w_k \in [0, 1] \text{ for all } k \in \mathcal{K} \tag{6}$$

$$\eta_i \geq 0 \text{ for all } i \in E_r. \tag{7}$$

The variable w_k is restricted to lie in $[0, 1]$ and is positive if and only if clause $k \in \mathcal{K}$ is a part of the scoring function (2). The parameter κ is an upper bound on the *complexity* of the scoring function (defined as the number of clauses plus the number of relations across all clauses). η_i is a penalty variable which is positive if the scoring function defined by positive w_ks gives a value less than 1 to the ith edge in E_r. Therefore, the $\sum_{i=1}^m \eta_i$ portion of the objective function attempts to maximize $\sum_{i=1}^m \min\{f_r(X_i, Y_i), 1\}$, i.e., it attempts to approximately maximize the number of facts in E_r that are given a "high-score" of 1 by f_r. In addition, we have the parameter $\tau > 0$ which represents a tradeoff between how well our weighted combination of rules performs on the known facts (gives positive scores), and how poorly it performs on some "missing" facts. We make this precise shortly. Maximizing the MRR is a standard objective for KGC and thus the objective function of LPR is only an approximation; see the next Theorem. We still obtain high-quality prediction rules using LPR.

Theorem 1. *Let IPR be the integer programming problem created from LPR by replacing equation (6) by $w_k \in \{0, 1\}$ for all $k \in \mathcal{K}$, and letting $\tau = 0$. Given an optimal solution with objective function value γ, one can construct a scoring function such that $1 - \gamma/m$ is a lower bound on the MRR of the scoring function calculated by the optimistic ranking method, when applied to the training set triples.*

The theorem above justifies choosing IPR as an optimization formulation to find good rule sets for a relation, assuming MRR calculation via optimistic ranking. When using random break ranking, it is essential to perform negative sampling and penalize rules that create paths when there are no edges in order to produce good quality results. This is why we use $\tau > 0$ in LPR. We will now give an interpretation of $\sum_k \text{neg}_k w_k$. To compute the MRR of the scoring function f_r in (2) applied to the training set, for each edge $(t, r, h) \in E_r$ we need to compute the rank of the answer h to the query $(t, r, ?)$ – by comparing $f_r(t, v)$ with $f_r(t, h)$ for all nodes v in $\mathcal{G}$ – and the rank of answer t to the query $(?, r, h)$ – by comparing $f_r(v, h)$ with $f_r(t, h)$ for all nodes v. But $\sum_k \text{neg}_k w_k$ is exactly the sum of scores given by f_r to all nodes in right_k and left_k and therefore we have the following remark.

Remark 2. *Let (t, r, h) be an edge in E_r, and let $U(?, r, h)$ and $U(t, r, ?)$ be the set of invalid answers for $(?, r, h)$ and $(t, r, ?)$, respectively. Then*

$$\sum_{(t,r,h)\in E_r} \left(\sum_{v\in U(?,r,h)} f_r(v, h) + \sum_{v\in U(t,r,?)} f_r(t, v) \right) = \sum_{k\in\mathcal{K}} \text{neg}_k w_k.$$

In other words, rather than keeping individual scores of the form $f_r(v, h)$ and $f_r(t, v)$ for missing facts (v, r, h) or (t, r, v) small, we minimize the sum of these scores in LPR.

It is impractical to solve LPR given the exponentially many variables w_k, except when n and L are both small. An effective way to solve such large LPs is to use column generation where only a small subset of all possible w_k variables is generated explicitly and the optimality of the LP is guaranteed by iteratively solving a *pricing* problem. We do not attempt to solve LPR to optimality. We start with an initial set of candidate rules $\mathcal{K}_0 \subset \mathcal{K}$ (and implicitly set all rule variables from $\mathcal{K} \setminus \mathcal{K}_0$ to 0). Let LPR_0 be the associated LP. If the solution of LPR_0 is not satisfactory, we dynamically augment the set of candidate rules to create sets $\mathcal{K}_i$ such that $\mathcal{K}_0 \subset \mathcal{K}_1 \subset \cdots \subset \mathcal{K}$. If LPR_i is the LP associated with $\mathcal{K}_i$ with optimal solution value z^i_{min}, then it is clear that a solution of LPR_i yields a solution of LPR_{i+1} by setting the extra variables in LPR_{i+1} to zero, and therefore $z^{i+1}_{min} \le z^i_{min}$. We attempt to have $z^{i+1}_{min} < z^i_{min}$ by taking the dual solution associated with an optimal solution of LP_i, and then trying to find a *negative reduced cost* rule, which we discuss shortly.

Datasets	# Rel.	# Entities	# Train	# Test	# Valid
Kinship	25	104	8544	1074	1068
UMLS	46	135	5216	661	652
WN18RR	11	40943	86835	3134	3034
FB15k-237	237	14541	272115	20466	17535
YAGO3-10	37	123182	1079040	5000	5000

Table 1: Sizes of datasets.

Setting up the initial LP. To set up $\mathcal{K}_0$ and LP_0, we develop two heuristics. In Rule Heuristic 1, we generate rules of lengths one and two for a relation r. We create a one-relation rule from $r' \in \mathcal{R} \setminus \{r\}$ if it labels a large number of edges from tail nodes to head nodes of edges in E_r. We essentially enumerate the length-two rules $r_1(X, Y) \wedge r_2(Y, Z)$ and keep those that frequently create paths from the tail nodes to head nodes of edges in E_r. In Rule Heuristic 2, we take each edge (X, Y) in E_r and find a shortest path from X to Y contained in the edge set $E \setminus \{(X, Y)\}$ where the path length is bounded by a pre-determined maximum length. We then use the sequence of relations associated with the shortest path to generate a rule. We also use a path of length at least one more than the shortest path.

Adding new rules. Each $\mathcal{K}_i$ for $i > 0$ is constructed by adding new rules to $\mathcal{K}_{i-1}$. We use a modified version of Heuristic 2 to generate the additional rules. Let $\delta_i \ge 0$ for all $i \in E_r$ be dual variables corresponding to constraints (4). Let $\lambda \le 0$ be the dual variable associated with the constraint (5). Given a variable w_k which is zero in a solution of LPR_i, dual solution values $\bar{\delta}$ and $\bar{\lambda}$ associated with the optimal solution of LPR_{i-1}, the reduced cost red_k for w_k is

$$\text{red}_k = \tau\, \text{neg}_k - \sum_{i\in E_r} a_{ik}\bar{\delta}_i - (1 + |C_k|)\bar{\lambda}$$

If $\text{red}_k < 0$, then increasing w_k from zero may reduce the LP solution value. In our heuristic, we sort the dual values $\bar{\delta}_j$ in decreasing order, then go through the associated indices j and create rules k such that $a_{jk} = 1$ via a shortest path calculation. That is, we take the corresponding edge (X, Y) in E_r, find the shortest path between X and Y and generate a new rule with the sequence of relations in that path. We limit the number of rules generated so that $|\mathcal{K}_i| - |\mathcal{K}_{i-1}| \le 10$. We do not add new rules to $\mathcal{K}_{i-1}$ if their reduced cost is nonnegative. We describe our overall method in Algorithm 1 and give more details in (Dash and Gonçalves 2023).

Experiments

We conduct KGC experiments with 5 datasets: Kinship (Denham 1973), UMLS (McCray 2003), FB15k-237 (Toutanova and Chen 2015), WN18RR (Dettmers et al. 2018), and YAGO3-10 (Mahdisoltani, Biega, and Suchanek 2015). The partition of FB15k-237, WN18RR, and YAGO3-10 into training, testing, and validation data sets is standard. We use the partition for UMLS and Kinship in Dettmers et al. (2018). In Table 1, we give the number of entities and relations in each dataset, and facts in each partition.

Problem	metric	ComplEx-N3	TuckER	†ConvE	AnyBURL	NeuralLP	DRUM	RNNLogic	LPRules
Kinship	MRR	0.889	0.891	0.83	0.626	0.652	0.566	*0.687*	**0.746**
	H@1	0.824	0.826	0.74	0.503	0.52	0.404	*0.566*	**0.639**
	H@10	0.986	0.987	0.98	0.901	0.925	0.91	*0.929*	**0.959**
UMLS	MRR	0.962	0.914	0.94	**0.940**	0.75	0.845	0.748	*0.869*
	H@1	0.934	0.837	0.92	**0.916**	0.601	0.722	0.618	*0.812*
	H@10	0.996	0.997	0.99	*0.985*	0.958	**0.991**	0.928	0.97
FB15K-237	MRR	0.362	0.353	0.325	0.226	0.222	0.225	♯**0.288**	*0.255*
	H@1	0.259	0.259	0.237	0.166	0.16	0.16	**0.208**	*0.17*
	H@10	0.555	0.538	0.501	0.387	0.34	0.355	**0.445**	*0.402*
WN18RR	MRR	0.469	0.464	0.43	*0.454*	0.381	0.381	0.451	**0.459**
	H@1	0.434	0.436	0.4	**0.423**	0.367	0.367	0.415	*0.422*
	H@10	0.545	0.517	0.52	*0.527*	0.409	0.41	0.524	**0.532**
YAGO3-10	MRR	0.574	0.265	0.44	**0.449**				**0.449**
	H@1	0.499	0.184	0.35	**0.381**				*0.367*
	H@10	0.705	0.426	0.62	*0.598*				**0.684**

Table 2: Comparison of results on standard datasets. The results for NeuralLP, DRUM, RNNLogic, LPRules use the random break metric. † ConvE results are from Dettmers et al. (2018). ♯ We could not run RNNLogic on FB15k-237, and report numbers from Qu et al. (2021). We run the *light* version of AnyBURL which yields entity-independent rules only. The best and second best values in any row for rule-based methods are given in bold and italics, respectively.

Experimental Setup

We denote the reverse relation for $r \in \mathcal{R}$ by r^{-1}. For each fact (t, r, h) in the training set, we implicitly introduce the fact (h, r^{-1}, t). For each *original* relation r in the training set, we create a scoring function $f_r(X, Y)$ of the form in (2) and calculate performance metrics as described right after equation (2).

We compare our results with the rule-based methods AnyBURL, NeuralLP, DRUM, and RNNLogic. We use default settings for NeuralLP and DRUM and the settings proposed by the authors of RNNLogic. We modify RNNLogic to implement the random break method to break ties. AnyBURL uses a lexicographic method to break ties; we run it for 100 seconds with the *light* settings which cause AnyBURL to only generate entity-independent rules. We give results for the embedding-based methods ConvE (Dettmers et al. 2018), ComplEx-N3 (Lacroix, Usunier, and Obozinski 2018), and TuckER (Balažević, Allen, and Hospedales 2019), mainly to show the maximum achievable MRR on our datasets. We ran ComplEx-N3 and TuckER on our machines with the best published hyperparameters (if available). ConvE results are taken from Dettmers et al. (2018).

We ran two variants of our code which we call "LPRules"[1] (see Algorithm 1). In the first variant, we create LPR_0 by generating rules using Rule Heuristics 1 and 2, and then solve LPR_0 to obtain rules (thus MaxIter = 0). In the second variant (used only for YAGO3-10) we create LPR_0 with an empty set of rules and then perform 15 iterations consisting of generating up to 10 rules using the modified version of Rule Heuristic 2 followed by solving the new LP. In other words, we create and solve LPR_i for $i = 0, \ldots 15$. To compute neg_k, we sample 2% of the edges from E_r, and compute the number of invalid paths that start at tails or end at heads of these edges. We search for the best τ (from an input list) and κ for each relation. We dynamically let $\bar{\kappa}$ equal the length of the longest rule generated plus one. We then perform 20 iterations where, at the ith iteration, we set κ to $i\bar{\kappa}$. We use the validation data set to select those τ and κ that yield the best MRR. We set the maximum rule length to 6 for WN18RR, and 3 for YAGO3-10, and 4 for the other datasets. Thus $\kappa \leq 100$ except for WN18RR.

Results

We run the rule-based codes on a 60 core machine with 128 GBytes of RAM, and four 2.8 Intel Xeon E7-4890 v2 processors, each with 15 cores. In our code, we execute rule generation for each relation on a different thread, and solve LPs with CPLEX (IBM 2019). If one relation has many more facts than the others, as in YAGO3-10, then our code essentially uses one thread.

In Table 2, we give values for different metrics obtained with the listed codes, first for embedding methods, then for rule-based methods (if available), and then for our code. We could not run RNNLogic on FB15k-237 and use the published result. The best embedding method is better across all metrics than rule-based methods. Our method is the best

[1] https://github.com/IBM/LPRules

Algorithm 1: LPRules

Input: Train, test, & validation datasets.
Parameters: Sets of τ values & complexity bounds κ.
Output: Rules & evaluation metrics.

for each relation r in train **do**
  Call Rule Heur. 1 & 2 to generate initial rule set $\mathcal{K}_0$.
  Set up LPR$_0$ from $\mathcal{K}_0$ using min τ & min κ & solve.
  Set bestMRR $= 0$, bestκ $=$ bestτ $= -\infty$.
  for $i \leftarrow 1$ to MaxIter **do**
    Generate new rules with modified Rule Heur. 2.
    Add new rules with reduced cost < 0 to $\mathcal{K}_{i-1}$ to form $\mathcal{K}_i$, set up LPR$_i$ from $\mathcal{K}_i$ & solve.
  end for
  for each τ **do**
    for each κ **do**
      Set up LPR$_i$ from $\mathcal{K}_i$ using current value of τ & κ & solve.
      Use soln & compute MRR on validation facts.
      if MRR $>$ bestMRR **then**
        bestMRR = MRR.
        bestτ = current τ, bestκ = current κ.
      end if
    end for
  end for
  Set up LPR$_i$ from $\mathcal{K}_i$ using bestτ, bestκ & solve.
  Output rules with corresponding weights in LPR$_i$.
end for
Compute evaluation metrics on test set.

rule-based code across all metrics on Kinship, obtains better MRR and Hits@10 values on WN18RR and YAGO3-10, and second best values for FB15K-237 (and MRR and Hits@1 numbers for UMLS). This is notable as we use very simple rule generation heuristics, generate relatively compact rules, and take significantly less computing time than DRUM, NeuralLP, and RNNLogic (see Table 3). These three codes are unable to produce results for YAGO3-10 within a reasonable time frame. For YAGO3-10, our column generation approach, where we generate a small number of rules, and then use the dual values to focus on "uncovered" facts (not implied by previous rules) is essential.

In Table 3, we compare the average number of rules per relation (we could not extract rules from DRUM) in the final solution and the running times (DRUM has comparable running time to NeuralLP). Our code always obtains Pareto optimal solutions (when measured on MRR and average number of rules). For WN18RR and YAGO3-10, we obtain the best MRR values with few rules: we get excellent results for YAGO3-10 with just 7.8 rules on average.

In Figure 1, we show how MRR varies with average number of rules in the solution. RNNLogic chooses top-K rules for testing (K is an input parameter), and we run it with different values of K. For AnyBURL, we take the rules generated at 10, 50, and 100 seconds. We are unable to control the output number of rules in NeuralLP. Figure 1 (b) is especially striking and demonstrates a much better MRR to number of rules tradeoff than RNNLogic on WN18RR.

Problem	AnyBURL	NeuralLP	RNNLogic	LPRules
# rules/relation				
Kinship	6653.1	10.4	200.0	21.0
UMLS	1837.6	15.1	100.0	4.2
FB15k-237	79.9	8.1		14.2
WN18RR	47.3	14.3	200.0	15.6
YAGO3-10	63.0			7.8
running time (minutes)				
Kinship	1.7	1.6	108.8	0.5
UMLS	1.9	1.1	133.4	0.2
FB15k-237	3.9	14565.9		234.5
WN18RR	1.8	399.9	104.0	11.0
YAGO3-10	34.3			1648.4

Table 3: Average number of rules selected per relation and wall clock running time on a 60 core machine.

Our LP formulation can be initiated with *any input candidate set* of rules. In Figure 2 (a) and (b), we show the effect of combining rules generated by AnyBURL and RNNLogic with our rules. MRR values are shown on the y-axis and the letters A,B,C,D on the x-axis stand for four scenarios. In Scenario A, we run another rule-based code. In Scenario B, we take as input the rules and rule-weights from Scenario A to build our scoring function. In Scenario C, we give these rules to our LP formulation, and recalculate weights, while limiting solution complexity. Finally, in Scenario D, we give the rules in Scenario C along with our heuristically generated rules to our LP formulation. Our scoring function is neither better nor worse than those of AnyBURL and RNNLogic (or NeuralLP). Using the same rules and rule weights generated by AnyBURL (in Scenario A), our scoring function produces (in Scenario B) better MRR in two cases and worse in three cases. Just using our scoring function instead of AnyBURL's, we get an MRR of **0.267** (instead of 0.226) for FB15K-237, and an MRR of **0.480** (instead of 0.449) for YAGO3-10; these values are better than those obtained by either AnyBURL or LPRules. We get better MRR values in Scenario C than AnyBURL for Kinship and FB15K-237 with much more compact solutions. That is we choose a subset of the AnyBURL rules, give different weights, and yet get a better solution. It is hard to see a trend going from Scenario B to C. We conclude that our LP approach can combine rules generated by other methods with our rules, and get similar or larger MRR values while choosing few rules.

Entity-independent rules have a strong inductive bias, especially when very few rules are generated per relation. We compare our code on an inductive benchmark dataset with GraIL (Teru, Denis, and Hamilton 2020), which learns the entity-independent subgraph structure around the edges associated with each relation. The relational paths we use are special cases of the subgraphs learnt by GraIL. A low-quality solution of the much harder learning problem in GRAIL could lead to worse predictions than a high-quality solution of our simpler learning problem. We indeed obtain

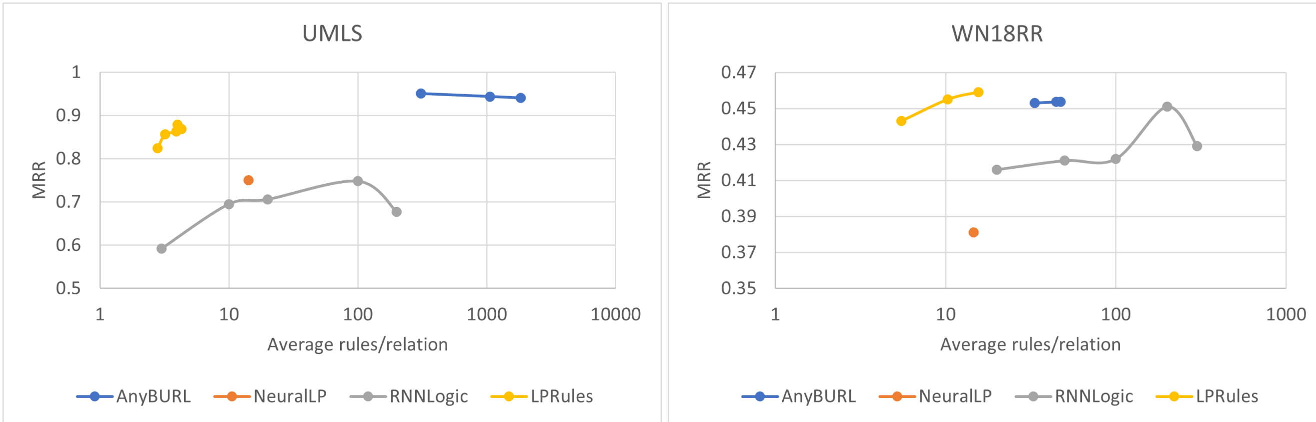

Figure 1: Change in MRR with change in average number of rules per relation for (a) UMLS and (b) WN18RR

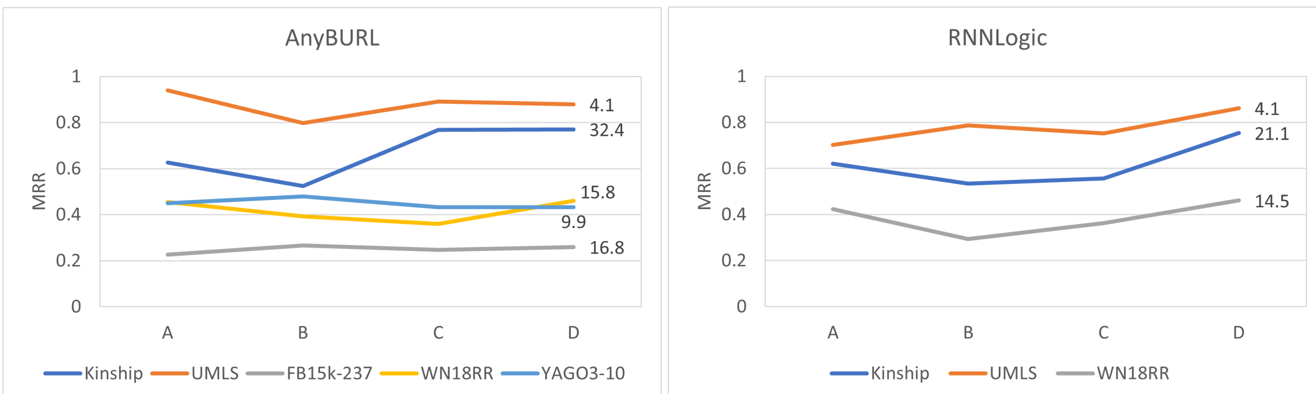

Figure 2: MRR values in different scenarios when using rules generated by (a) AnyBURL and (b) RNNLogic. MRR is shown on the y-axis. The number to the right of each curve is the average number of rules per relation in Scenario D.

better results than GraIL on its inductive benchmarks, see Table 4. Each dataset is split into a KG for training and a KG for testing that has a subset of the training KG relations, but no common entities. All GrailL results were obtained with the parameter "Negative Sampling Mode" set to "all".

	GraIL			LPRules		
Ver.	MRR	H@1	H@10	MRR	H@1	H@10
			WN18RR			
v1	0.556	0.434	0.769	0.682	0.636	0.745
v2	0.587	0.497	0.734	0.646	0.609	0.700
v3	0.303	0.254	0.391	0.398	0.359	0.460
v4	0.533	0.456	0.671	0.609	0.577	0.668
			FB15k-237			
v1	0.238	0.173	0.339	0.313	0.271	0.380
v2	0.203	0.129	0.328	0.396	0.315	0.539

Table 4: Results on inductive datasets. We were not able to obtain results for GraIL on FB15k-237_v3/v4.

Conclusion

Most rule-based methods do not focus on compact rule sets. Our relatively simple LP based method for selecting weighted logical rules returns state-of-the-art results for a number of standard KG datasets even with compact (and more interpretable) rule sets. It scales better than many neuro-symbolic methods. Our method also yields better results in an inductive setting than a recent solver that performs subgraph reasoning. Our work can be improved further in several areas such as accuracy and scaling with KG size. To improve scaling, one can sample facts when dealing with large KGs in order to obtain LPs of manageable size and also process different goups of facts on different machines. As demonstrated in Figure 2, accuracy can be improved by generating more rules using different algorithms. Handling ontologies, more complex queries, and more general rules would be other natural extensions of our work.

References

Balažević, I.; Allen, C.; and Hospedales, T. M. 2019. TuckER: Tensor Factorization for Knowledge Graph Completion. In *EMNLP 2019*.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NeurIPS 2013*.

Cropper, A.; and Morel, R. 2021. Learning programs by learning from failures. *Machine Learning*, 110: 801–856.

Cropper, A.; and Muggleton, S. H. 2016. Learning Higher-Order Logic Programs Through Abstraction and Invention. In *IJCAI 2016*.

Das, R.; Dhuliawala, S.; Zaheer, M.; Vilnis, L.; Durugkar, I.; Krishnamurthy, A.; Smola, A.; and McCallum, A. 2018. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *ICLR 2018*.

Dash, S.; and Gonçalves, J. 2023. Rule Induction in Knowledge Graphs Using Linear Programming. arXiv:2110.08245.

Dash, S.; Günlük, O.; and Wei, D. 2018. Boolean decision rules via column generation. In *Advances in Neural Information Processing Systems*, 4655–4665.

Demiriz, A.; Bennett, K. P.; and Shawe-Taylor, J. 2002. Linear programming boosting via column generation. *Machine Learning*, 46: 225–254.

Denham, W. 1973. *The detection of patterns in Alyawarra nonverbal behavior*. Ph.D. thesis, University of Washington.

Dettmers, T.; Pasquale, M.; Pontus, S.; and Riedel, S. 2018. Convolutional 2D Knowledge Graph Embeddings. In *AAAI 2018*.

Eckstein, J.; and Goldberg, N. 2012. An Improved Branch-and-Bound Method for Maximum Monomial Agreement. *INFORMS Journal on Computing*, 24(2): 328–341.

Eckstein, J.; Kagawa, A.; and Goldberg, N. 2019. REPR: Rule-Enhanced Penalized Regression. *INFORMS Journal on Optimization*, 1(2): 143–163.

Evans, R.; and Grefenstette, E. 2018. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61: 1–64.

IBM. 2019. IBM ILOG CPLEX Optimization Studio 12.10.0. https://www.ibm.com/docs/en/icos/12.10.0. Accessed: 2023-03-20.

Ji, S.; Pan, S.; Cambria, E.; Marttinen, P.; and Yu, P. S. 2021. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. *IEEE Transactions on Neural Networks and Learning Systems*, 1–21.

Kok, S.; and Domingos, P. 2005. Learning the structure of Markov Logic Networks. In *ICML 2005*.

Kovalerchuk, B.; Ahmad, M.; and Teredesai, A. 2021. Survey of Explainable Machine Learning with Visual and Granular Methods Beyond Quasi-Explanations. In Pedrycz, W.; and Chen, S., eds., *Interpretable Artificial Intelligence: A Perspective of Granular Computing*, 217–267. Springer.

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

Lao, N.; and Cohen, W. W. 2010. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, 81: 53–67.

Lin, X. V.; Socher, R.; and Xiong, C. 2018. Multi-hop knowledge graph reasoning with reward shaping. In *EMNLP 2018*.

Mahdisoltani, F.; Biega, J.; and Suchanek, F. M. 2015. YAGO3: A Knowledge Base from Multilingual Wikipedias. In *CIDR 2015*.

McCray, A. T. 2003. An upper level ontology for the biomedical domain. *Comparative and Functional Genomics*, 4: 80–84.

Mehrabi, N.; Morstatter, F.; Saxena, N.; Lerman, K.; and Galstyan, A. 2022. A Survey on Bias and Fairness in Machine Learning. *ACM Computing Surveys*, 54: 1–35.

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

Michie, D. 1988. Machine learning in the next five years. Proceedings of the Third European Working Session on Learning, 107–122. Pitman.

Muggleton, S. H.; Schmid, U.; Zeller, C.; Tamaddoni-Nezhad, A.; and Besold, T. 2018. Ultra-Strong Machine Learning: comprehensibility of programs learned with ILP. *Machine Learning*, 107: 1119–1140.

Qu, M.; Chen, J.; Xhonneux, L.-P.; Bengio, Y.; and Tang, J. 2021. RNNLogic: Learning Logic Rules for Reasoning on Knowledge Graphs. In *ICLR 2021*.

Richardson, M.; and Domingos, P. 2006. Markov logic networks. *Machine Learning*, 62: 107–136.

Rochstätel, T.; and Riedel, S. 2017. End-to-end differential proving. In *NeurIPS 2017*.

Rudin, C.; Chen, C.; Chen, Z.; Huang, H.; Semenova, L.; and Zhong, C. 2022. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys*, 16: 1 – 85.

Sadeghian, A.; Armandpour, M.; Ding, P.; and Wang, D. Z. 2019. DRUM: End-To-End Differentiable Rule Mining On Knowledge Graphs. In *NeurIPS 2019*.

Schmid, U.; Zeller, C.; Besold, T.; Tamaddoni-Nezhad, A.; and Muggleton, S. H. 2017. How does predicate invention affect human comprehensibility? In *ILP 2016*.

Sen, P.; Carvalho, B. W. S. R. d.; Riegel, R.; and Gray, A. 2022a. Neuro-Symbolic Inductive Logic Programming with Logical Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8): 8212–8219.

Sen, P.; de Carvalho, B. W. S. R.; Abdelaziz, I.; Kapanipathi, P.; Roukos, S.; and Gray, A. 2022b. Logical Neural Networks for Knowledge Base Completion with Embeddings & Rules. In *EMNLP 2022*.

Shen, Y.; Chen, J.; Huang, P.-S.; Guo, Y.; and Gao, J. 2018. M-Walk: Learning to Walk over Graphs using Monte Carlo Tree Search. In *NeurIPS 2018*.

Sun, Z.; Vashishth, S.; Sanyal, S.; Talukdar, P.; and Yang, Y. 2020. A Re-evaluation of Knowledge Graph Completion Methods. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 5516–5522.

Teru, K. K.; Denis, E. G.; and Hamilton, W. L. 2020. Inductive relation prediction by subgraph reasoning. In *ICML 2020*.

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 (CVSC)*, 57–66.

Wang, Q.; Mao, Z.; Wang, B.; and Guo, L. 2017a. Knowledge graph embedding: A survey of approaches and applications. *IEEE TKDE*, 29: 2724–2743.

Wang, T.; Rudin, C.; Doshi-Velez, F.; Liu, Y.; Klampfl, E.; and MacNeille, P. 2017b. A Bayesian framework for learning rule sets for interpretable classification. *Journal of Machine Learning Research*, 18: 1–37.

Xiong, W.; Hoang, T.; and Wang, W. Y. 2017. Deeppath: a reinforcement learning method for knowledge graph reasoning. In *EMNLP 2017*.

Yang, F.; Yang, Z.; and Cohen, W. W. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In *NIPS 2017*.

Yang, Y.; and Song, L. 2020. Learn to explain efficiently via neural logic inductive learning. In *ICLR 2020*.