An Evolutionary Algorithm for Rule Learning over Knowledge Graphs

Lianlong Wu[1], Emanuel Sallinger[2,1], Evgeny Sherkhonov[1], Sahar Vahdati[1], and Georg Gottlob[2,1]

[1] University of Oxford
[2] TU Wien

Abstract. Logical rules allow us to declaratively encode expert knowledge, express patterns and infer new knowledge from existing one in Knowledge Graphs. However, the construction of rules is often a costly manual process. In this work, we address the problem of *learning* rules from the already exiting knowledge. Most of the existing rule learning algorithms for Knowledge Graphs are based on inefficient full search plus greedy pruning strategies that cover limited search space by considering rules of a predetermined shape. We propose a learning algorithm where a set of rules is learned via the process inspired by biological evolution and that is geared towards optimizing a fitness function. Such evolutionary algorithms typically cover larger search space efficiently and provide multiple near-optimal solutions. We evaluate the proposed algorithm on a number of public Knowledge Graphs and compare it to other rule learning algorithms.

Keywords: Rule Learning · Knowledge Graphs · Evolutionary Algorithms · Genetic Programming

1 Introduction

Knowledge Graphs (KGs) are the hyped technology of recent years and have gained huge interest in multiple AI-based applications. Logical rules are used as an intrinsic part of the KGs encoding knowledge, or for Knowledge Graph construction and completion as well as error detection and noise identification, entity resolution, query and answering, among other tasks. The main characteristic added to such downstream tasks originates from explainability power of rule-based languages as they are human and machine-friendly. This makes reasoning as deriving new knowledge from the existing ones also fully explainable.

However, the process of gaining logical rules is costly as in traditional process for obtaining high-quality rules in a KG (often addressed as knowledge engineering) significant human effort is required. Rule learning is the process of automatically learning rules from underlying KGs. The two manual and automated

processes are often complementary, allowing engineered background knowledge to be used at the same time as rules learned from data, jointly obtaining a higher-quality result compared to each process used in isolation.

Among the existing methods, the classical Inductive Logic Programming (ILP) methods for rule learning are based on the Closed World Assumption (CWA) and consider relatively small scale datasets. However, modern large scale KGs cause huge challenges for the efficiency of such rule learning methods with straightforward use of ILP methods [4].

There are recent tools which are specifically designed for Knowledge Graph rule extraction, considering the Open World Assumptions (OWA). For example, Ontological Pathfinding (OP) [2] and AMIE [4] are among the recent methods which are proposed based on exhaustive search with pruning strategies. Another tool named as RuDiK[8] uses a greedy algorithm and A^* graph traversal algorithm over the entire possible rule space. These tools have proven feasible performance over modern large scale KGs. However, greedy algorithms or search with pruning may suffer from being trapped in local optima. Also, the aforementioned tools often have high running times over large scale KGs with a limited length of rule where the rule types are also tied to the algorithm in these systems.

Approach and contribution. In this paper we propose an evolutionary rule learning algorithm that provides multiple near-optimal solutions efficiently, and, moreover, has fewer syntactic restrictions of the form of learned rules.

Rule learning can be considered as heuristic search on the concept description space. Evolutionary Algorithms (EA) are a good choice for this intrinsic search mechanism and the symbolic representation. In order to extend the search space, this work focuses on studying Evolutionary Algorithms, specifically in the area of Genetic Programming (GP) for the rule learning problem over large scale Knowledge Graph datasets. EAs are a family of biology-inspired search algorithms that optimize for the most promising preliminary solutions, while exploring a wide search space at the same time. In particular, in our setting, one rule or a set of rules can be treated as a chromosome, from which a new population of chromosomes can be derived via the operations of mutation, crossover and selection operators. While performing these operations, a quality measure, called fitness function is computed to judge whether an obtained new generation of chromosomes is fit for continuing the search.

Such evolutionary algorithms typically do not need to perform exhaustive search, and are less likely to fall into local optima[3]. Furthermore, they are flexible in that they might not require imposed template on the shape of rules, as it is typically the case in other approaches. In addition, EA supports multiple metrics as optimization target: Precision, Support (Recall) etc. and any combination of them. EAs are well balanced by optimizing the most promising solutions while exploring the wider search space at the same time. While greedy search may be trapped in local optima, the parallel search mechanism and mutation operations of Genetic Programming can avoid such traps. The robustness of EAs imposes fewer limitations on the rule learning system compared to other methods. It is easy to incorporate arbitrary constraints into the search space. Furthermore,

EAs are adaptable in a complex search space, which makes them increasingly utilized in the different learning systems [6]. In this work, we are combining the learning power of Genetic Programming (GP) and Knowledge Representation (KR) from logic-based languages. The main contributions of this paper are:

- We introduce a new evolutionary algorithm for rule learning which, to the best of our knowledge, is a first genetic programming algorithm applied in the context of learning rules over Knowledge Graphs.
- We report preliminary experimental results of the new algorithm on publicly available Knowledge Graphs YAGO [10] and YAGO2 [9].

Organization. Section 2 introduces the necessary preliminaries. In Section 3 we will introduce the approach and algorithm. Section 4 presents the initial experimental evaluation. We give concluding remarks in Section 5.

2 Preliminaries

In this section we present the necessary concepts that are required for better understanding of our rule learning algorithm. Let us assume a disjoint sets of *entities* indicated as $\mathcal{E}$, *literals* as $\mathcal{L}$, *predicates* as $\mathcal{P}$ of arity two, and *variables* depicted by $\mathcal{V}$.

Knowledge Representation is by *atoms* in the form of $p(X, Y)$, where $p \in \mathcal{P}$, $X \in \mathcal{E} \cup \mathcal{V}$ and $Y \in \mathcal{E} \cup \mathcal{V} \cup \mathcal{L}$. For example, $livesIn(X, \texttt{Oxford})$ is an atom, where $livesIn$ is a predicate, `Oxford` is an entity and X is a variable, and it denotes that person X lives in Oxford.

Knowledge Graphs (KGs) are sets of ground atoms indicated by $\mathcal{K}$, i.e., atoms $p(s, o)$ such that $s \in \mathcal{E}$ and $o \in \mathcal{E} \cup \mathcal{L}$. For instance,

$$\mathcal{K}_1 = \{livesIn(\texttt{John}, \texttt{Oxford}), isMarriedTo(\texttt{John}, \texttt{Mary})\}$$

is a small KG that states the fact that John lives in Oxford and that he is married to Mary.

A *Horn rule*, or simply a *rule*, is an expression of the form of $H \leftarrow \mathcal{B}$, where $\mathcal{B}$ is a set of atoms $\{B_1, \ldots, B_n\}$ called *body atoms*, and H is an atom called *head atom*. For instance, the expression

$$livesIn(Y, Z) \leftarrow livesIn(X, Z), isMarriedTo(X, Y) \tag{1}$$

is a rule that expresses the fact that whenever X lives in Z and X is married to Y, then Y also lives in Z. A *program* is a set of rules.

A mapping from $\mathcal{V}$ to $\mathcal{E} \cup \mathcal{L} \cup \mathcal{V}$ is called a *substitution*. For a rule r of the form $H \leftarrow A_1, \ldots, A_n$ and a substitution σ, by $\sigma(r)$ we denote the result of applying σ to r, i.e., the rule obtained by substituting the variables in $H, A_1, \ldots, A_n$ according to σ. For a KG $\mathcal{K}$ if for every $i \in \{1, \ldots, n\}$ it holds that $\sigma(A_i) \in \mathcal{K}$, then $\sigma(H)$ is called a *prediction* of r over $\mathcal{K}$. If additionally $\mathcal{K}$ contains $\sigma(H)$, then it is a *true* prediction, otherwise it is a *false* prediction. In our example $livesIn(\texttt{Mary}, \texttt{Oxford})$ is a false prediction of the rule (1) over $\mathcal{K}_1$.

We define the *support* of a rule r in a KG $\mathcal{K}$, denoted as $S_{rule}(r, \mathcal{K})$, as the number of true predictions of r over $\mathcal{K}$. Additionally, for a rule $r = H(X, Y) \leftarrow \mathcal{B}$, the *support* of the body $\mathcal{B}$, denoted as $S_{body}(r, \mathcal{K})$, is defined as the number of substitutions σ defined over X and Y that can be extended to a substitution σ' (i.e., $\sigma'(X) = \sigma(X)$ and $\sigma'(Y) = \sigma(Y)$) such that $\sigma'(\mathcal{B}) \subseteq \mathcal{K}$. In our example, for the rule (1) and KG $\mathcal{K}_1$, we have $S_{rule}(r, \mathcal{K}_1) = 0$ and $S_{body}(r, \mathcal{K}_1) = 1$.

The *standard confidence* is defined as the ratio $S_{rule}(r, \mathcal{K})/S_{body}(r, \mathcal{K})$ and reflects the measure how well the rule r fits the KG $\mathcal{K}$. However, it has been argued in [4] that the standard confidence is not particularly well suited to measure the quality of rules over KGs due to the open world assumption for KGs. Instead, we resort to the PCA confidence [4].

The *Partial Completeness Assumption* (PCA) asserts that if for some a and b it holds that $p(a, b) \in \mathcal{K}$, then for every fact $p(a, b')$ that is true it holds that $p(a, b') \in \mathcal{K}$. In other words, if a KG contains a p-fact of a, then it contains all true p-facts of a. Moreover, if no p-fact of a is known, then we consider all p-facts of a unknown. For instance, under the PCA, our KG $\mathcal{K}_1$ contains all the true $livesIn$-facts about `John`, while the $livesIn$-facts about `Mary` are unknown.

The *PCA confidence* $C_{pca}(r, \mathcal{K})$ is then a refined version of the standard confidence, where the denominator does not take into account the unknown H-facts of the first argument:

$$C_{pca}(r, \mathcal{K}) = \frac{S_{rule}(r, \mathcal{K})}{\#(a, b) : \exists \bar{c}.\mathcal{B} \in \mathcal{K} \wedge \exists b'.H(a, b') \in \mathcal{K}} \tag{2}$$

In our evolutionary algorithm introduced in the next section we use the PCA confidence as the fitness function.

3 Algorithm

In this section we introduce an algorithm that belongs to the family of Evolutionary Algorithms (EA), also known as Genetic Algorithms (GA). EAs are a family of biology-inspired parallel search algorithms that optimize for the most promising preliminary solutions, while exploring a wide search space at the same time. In particular, in our setting programs can be treated as chromosomes, from which the new population of chromosomes can be derived via the operations of selection, mutation and crossover.

We first introduce the basic elements of our algorithm, the operators that perform transformation of a given set of programs: selection, mutation and crossover.

Selection Operators. The selection operator is used for take one or two chromosomes from the current population. Fitness-Proportional (Roulette-Wheel) selection method is used, the chance of each individual being selected is proportional to its fitness score. Each individual is assigned a probability of

$$p_i = \frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)}, \tag{3}$$

where f is a given fitness function. By $selector(x, f)$ we denote the result of applying the selection operator with the fitness function f.

Mutation Operator. We say that a rule is obtained by *mutating* the input rule if either a variable or a predicate that occurs in the input rule is mutated. Mutating a variable X_i means replacing it with another variable $X_j, i \neq j$. For example, the rule $livesIn(X_1, X_2) \leftarrow wasBornIn(X_1, X_2)$ is obtained from $livesIn(X_1, X_2) \leftarrow wasBornIn(X_1, X_3)$ by mutating the variable X_3 to X_2. Similarly, mutating a predicate R_i means replacing it by another predicate $R_j, i \neq j$. For example, the rule $livesIn(X_1, X_2) \leftarrow worksAt(X_1, X_2)$ is obtained from $livesIn(X_1, X_2) \leftarrow wasBornIn(X_1, X_2)$ by mutating the body predicate $wasBornIn$ to $worksAt$. We say that a program (chromosome) P_1 is obtained from a program P_2 by *mutating*, if it is obtained by mutating a rule in P_2. For a given chromosome x, by $mutate(x, P_m)$ we denote the result of applying a mutation to x with probability P_m.

Crossover Operator. The crossover operator is applied to two given chromosomes. The operator randomly chooses a rule from each chromosome, and subsets of atoms in the body of each of the rules and swaps them respecting the variable renaming. For instance, for the programs $\{r_1\}$ and $\{r_2\}$ the crossover operator can select $\{isCitizenOf(X_1, X_2)\}$ from the body of r_1, and $\{livesIn(X_1, X_2)\}$ from the body of r_2.

$$\begin{aligned} &r_1 : isLeaderOf(X_1, X_2) \leftarrow wasBornIn(X_1, X_2), isCitizenOf(X_1, X_2). \\ &r_2 : isLeaderOf(X_1, X_2) \leftarrow livesIn(X_1, X_2). \end{aligned} \tag{4}$$

The result of swapping these sets is then the programs $\{r'_1\}$ and $\{r'_2\}$, where

$$\begin{aligned} &r'_1 : isLeaderOf(X_1, X_2) \leftarrow wasBornIn(X_1, X_2), livesIn(X_1, X_2). \\ &r'_2 : isLeaderOf(X_1, X_2) \leftarrow isCitizenOf(X_1, X_2). \end{aligned} \tag{5}$$

For two given chromosomes x_1 and x_2, by $crossover(x_1, x_2, P_c)$ we denote the result of applying the crossover operator to x_1 and x_2 with the probability P_c.

Initial Population consists of $|\mathcal{P}|$ chromosomes, each chromosome has one rule with a single body atom, each rule has a different predicate.

We are ready to describe our main algorithm as shown in Algorithm 1. As input it takes the chromosome population size N, the maximum generation number M, the crossover probability P_c, and the mutation probability P_m. The initial population consists of N chromosomes. For each generation, a set of chromosomes (or programs) is generated, and fitness score for each of them is calculated, using a rule reasoning engine. The reasoning engine stores the knowledge graph facts, evaluates candidate programs, and computes the fitness function. In our experiments, the Vadalog reasoning engine is used [1]. Once all the fitness scores are calculated, pairs of chromosomes are selected, the crossover operator produces two new offspring, and mutation operators are applied to them. The new chromosomes are placed in the new population. Duplicated chromosomes are removed, in order to preserve the diversity of the population. These steps are repeated until the new population size becomes equal to N. The algorithm iterates until maximum generation M is reached.

Algorithm 1 Rule Learning Evolutionary Algorithm

Input: Knowledge Graph $\mathcal{K}$, Population size N, Maximum Generation M
Input: Fitness Function $f(x) = C_{pca}(x, \mathcal{K})$,
Input: Crossover probability P_c, Mutation probability P_m
Initialize population set $S' \leftarrow \{r_1..r_{|\mathcal{P}|}\}$.
for $g = 1$ **to** M **do**
  $S \leftarrow S'$
  $S' \leftarrow \{x_m\}, x_m = argmax(f(x)), x \in S$
  repeat
    $x_i \leftarrow selector(S, f); x_j \leftarrow selector(S, f)$
    $x'_i, x'_j \leftarrow crossover(x_i, x_j, P_c)$
    $x'_i \leftarrow mutate(x_i, P_m); x'_j \leftarrow mutate(x_j, P_m)$
    $S' \leftarrow S' \cup x'_i, x'_j$
  until $|S'| \geq N$
end for
Output: $argmax(f(x)), x \in S'$

4 Experiments

In this section, we present our results from evaluation of the proposed algorithm on a set of benchmarks.

Dataset As our dataset, we use the the YAGO KG which contains entities such as persons, organizations, and cities, and relations (facts) between the entities. YAGO2 nearly has 1 million facts [10] and we use a complete version of it. We also included a different version of it namely YAGO2s [9] with 4 million facts. Table 1 provides a detailed statistics about these KGs.

Table 1. Knowledge Graph Dataset Characters

KG Name	# Triples	# Predicates
YAGO2	948,358	33
YAGO2s	4,122,426	37

Experimental Results As shown in Table 2, our algorithm found 18 rules for YAGO2 and 20 rules for YAGO2s, with PCA confidence threshold $> 10\%$, and no explicit rule length limitation. The PCA confidence for the top 5 rules in YAGO2 is 97.14% and 85.26% for YAGO2s, the average PCA confidence of all the rules is 57.72% and 52.34%, respectively. The precision rate is evaluated by sampling five facts predicted by the rule, then verifying through human judgement. For reference, top 10 rules from AMIE on YAGO2 are at precision of 39%[4], and top 5 rules from RuDiK on YAGO3 are at precision 79.17% and all rules average precision at 62.86%[8]. In general, compared with RuDiK and AMIE, we find fewer rules with higher confidence and quality in terms of precision.

Table 2. Knowledge Graph Rule Learning Precisions

KG Name	# Rules	PCA Confidence		Precision	
		Top-5	Average	Top-5	Average
YAGO2	18	97.14%	57.72%	48%	61.11%
YAGO2s	20	85.26%	52.34%	80%	64.00%

Hardware Specification In the *standalone* setup, experiments are run on 48-core machine with 2 × Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.5GHz, 64GB RAM.

Execution Time As one of the comparison criteria, we considered the execution time. As shown in Table 3, rule learning for all the 33 predicates on the YAGO2 takes 12 minutes, and takes 21 minutes for the YAGO2s dataset. The run time of AMIE, OP and RuDiK are quoted from their typical setup in the publications[3]. It shows our running time is at the same level with the state-of-the-art systems. It also proved that our algorithm takes less time for execution when the KG gets larger. This is particularly important in real world scenarios of large-scale KGs.

Table 3. Rule Learning Execution time comparison

	AMIE[4]	AMIE+[5]	OP[2]	RuDiK[7]	This Work
YAGO2	3.62m	8.35m	3.59m	18m	12m
YAGO2s	-	59.38m	19.4m	47m	21m

5 Conclusion

In this work, we highlighted the problem of rule learning in knowledge graphs and provided a formal definition for that. The undulate goal is to emphasis on the importance of explicit rule learning. A novel genetic programming algorithm is proposed for rule mining over large scale Knowledge Graphs. The initial experiments confirm the feasibility and efficiency of this algorithms. This work provides the foundation for the development of a comprehensive framework for rule learning into a new scope with longer rule length limitation. Our approach governs less language restrictions, so that a larger hypothesis space could be explored in order to provide high quality rules. In future work of this research, we aim at addressing complex rule extraction and learning within a single framework. The experiments are planned to be extended and comparisons will be provided to other rule extraction and learning tools.

[3] Note: Considering the differences in the problem definition, the outcome of learning specifications, the algorithm parameters and the database implementations, the run time comparison is for illustration purpose only.

Acknowledgements. The work on this paper was supported by EPSRC programme grant EP/M025268/1, the EU H2020 grant 809965, and the Vienna Science and Technology (WWTF) grant VRG18-013.

References

1. Bellomarini, L., Sallinger, E., Gottlob, G.: The Vadalog system: Datalog-based Reasoning for Knowledge Graphs. In: Proceedings of the VLDB Endowment. vol. 11, pp. 975–987 (2018). https://doi.org/10.14778/3213880.3213888
2. Chen, Y., Goldberg, S., Wang, D.Z., Johri, S.S.: Ontological pathfinding: Mining first-order knowledge from large knowledge bases. Proceedings of the ACM SIGMOD International Conference on Management of Data pp. 835–846 (2016). https://doi.org/10.1145/2882903.2882954
3. Fogel, D.B.: Evolutionary Computation: Toward a New Philosophy of Machine Intelligence: Third Edition. John Wiley & Sons (2005). https://doi.org/10.1002/0471749214
4. Galárraga, L., Teflioudi, C., Hose, K., Suchanek, F.M.: AMIE: Association rule mining under incomplete evidence in ontological knowledge bases. WWW 2013 - Proceedings of the 22nd International Conference on World Wide Web pp. 413–422 (2013)
5. Galárraga, L., Teflioudi, C., Hose, K., Suchanek, F.M.: Fast rule mining in ontological knowledge bases with AMIE+. VLDB Journal **24**(6), 707–730 (2015). https://doi.org/10.1007/s00778-015-0394-1
6. Giordana, A., Saitta, L., Campidoglio, M.E., Bello, G.L.: Learning relations using genetic algorithms. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 728 LNAI, pp. 218–229 (1993). https://doi.org/10.1007/3-540-57292-9_60
7. Ortona, S., Meduri, V.V., Papotti, P.: Robust discovery of positive and negative rules in knowledge bases. Proceedings - IEEE 34th International Conference on Data Engineering, ICDE 2018 pp. 1180–1191 (2018). https://doi.org/10.1109/ICDE.2018.00108
8. Ortona, S., Meduri, V.V., Papotti, P.: RuDiK: Rule discovery in knowledge bases. Proceedings of the VLDB Endowment **11**(12), 1946–1949 (2018). https://doi.org/10.14778/3229863.3236231
9. Suchanek, F.M., Hoffart, J., Kuzey, E., Lewis-Kelham, E.: YAGO2s: Modular high-quality information extraction with an application to flight planning. In: Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI). vol. P-214, pp. 515–518 (2013)
10. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A core of semantic knowledge. 16th International World Wide Web Conference, WWW2007 pp. 697–706 (2007). https://doi.org/10.1145/1242572.1242667