Anytime Bottom-Up Rule Learning for Knowledge Graph Completion

Christian Meilicke[*] , **Melisachew Wudage Chekol** , **Daniel Ruffinelli** and **Heiner Stuckenschmidt**
University Mannheim
{christian, mel, daniel, heiner}@informatik.uni-mannheim.de

Abstract

We propose an anytime bottom-up technique for learning logical rules from large knowledge graphs. We apply the learned rules to predict candidates in the context of knowledge graph completion. Our approach outperforms other rule-based approaches and it is competitive with current state of the art, which is based on latent representations. Besides, our approach is significantly faster, requires less computational resources, and yields an explanation in terms of the rules that propose a candidate.

1 Introduction

Knowledge graph completion has become a vivid field of research within the last ten years. While current research is mainly concerned with latent representations that are based on the idea to embed a knowledge graph into a low dimensional vector space, symbolic approaches have attracted much less attention [Wang *et al.*, 2017]. However, such approaches have a big advantage, which is their ability to yield an explanation in terms of the rules that trigger a prediction.

In this paper we propose a bottom-up technique for efficiently learning logical rules from large knowledge graphs. Our work is inspired by bottom-up rule learning approaches Golem [Muggleton and Feng, 1990] and Aleph [Srinivasan, 2000], which have been developed in the early days of Inductive Logic Programming (ILP). A bottom-up approach is based on the idea that an example is a compact representation of a very specific rule that can be generalized to capture a comprehensive subset of all positive examples. Aleph is one of such approaches. It transforms a given positive example into a Horn rule called bottom rule. This rule is then generalized by dropping atoms from the rule body until a rule has been found that fulfills a chosen quality criteria. The examples covered by the rule are removed and the approach is again applied until all examples are covered.

Our approach differs in several ways. First of all, we do not have a strict border that tells us which observations belong to a given example and which do not belong to it. Instead, we have to decide what to treat as an example. We base our notion of an example on the concept of a path. In that sense our approach is similar to the path ranking algorithm (PRA) [Lao *et al.*, 2011]. However, PRA considers as features only a subset of the rules that we might learn.

Second, we are interested in learning uncertain rules, i.e., rules that cover at least some positive and usually also some negative examples. Even a rule with low confidence might still help us to create a better candidate ranking for the knowledge graph completion task. Moreover, we cannot remove the examples covered by a rule, because there might be other rules that cover also (some of) these examples and other examples, with a different confidence score.

Third, our algorithm is designed to be an efficient rule miner for knowledge graphs. Within a knowledge graph all facts (also referred to as triples) result from grounding binary predicates with constants. Thus, a knowledge graph can be deconstructed into a set of edge-labelled paths. This is the main reason why it makes sense to focus on paths in the absence of unary predicates or n-ary predicates with $n \geq 3$.

The problem of knowledge graph completion (or link prediction) is currently dominated by methods that embed a given knowledge graph in a latent feature space. Learning an explicit symbolic representation is rarely proposed as alternative. This might be related to the underlying assumption that a rule-based approach alone cannot solve more than just a trivial subset. See, for example the discussion of the inverse model proposed in [Dettmers *et al.*, 2018] and the critics related to the redundancies in FB15k [Toutanova and Chen, 2015]. Another assumption might be that rule learning cannot be applied to large datasets. For that reason current research is concerned with the combination of rules and embeddings (e.g., [Guo *et al.*, 2018]). We argue that the underlying assumptions are wrong and present results that support our claim. In particular, we propose an anytime bottom-up algorithm for learning rules and apply our approach to the knowledge completion task. We present results for five different datasets. Three of them have been proposed as hard cases for simple (rule-based) approaches that leverage symmetries and other redundancies. Our approach performs as good as and sometimes better than most models that have been proposed recently. The results of our approach are still very good if we stop the algorithm already after a short time period. Moreover, the required resources in terms of memory and runtime are significantly smaller compared to the resources required by latent approaches.

[*] Contact Author

2 Language Bias

A knowledge graph $\mathbb{G}$ is defined on top of a vocabulary $\langle \mathbb{C}, \mathbb{R} \rangle$ where $\mathbb{C}$ is a set of constants and $\mathbb{R}$ is a set of binary predicates. Hence, $\mathbb{G} = \{r(a, b) \mid r \in \mathbb{R},\ a, b \in \mathbb{C}\}$ is a set of ground atoms or facts. A binary predicate is called relation and a constant (or what the constant refers to) is also called entity. In the following we use lower-case letters for constants and upper-case letters for variables. Since we do not learn arbitrary Horn rules, there is a *language bias* towards what kind of rules can be learned as discussed below.

We call a rule as $h(c_0, c_1) \leftarrow b_1(c_1, c_2), ..., b_n(c_n, c_{n+1})$ a ground path rule of length n. The *head* of the rule is $h(...)$ and $b_1(...)$ through $b_n(...)$ is its *body*. We say that a ground path rule is straight if $c_k \neq c_l$ for $l, k \in \{1, ..., n\}$ with $l \neq k$ and if $c_0 \neq c_k$ with $0 < k < n + 1$. Such a rule does not have cycles in the body. In our formalization we abstract from the order of variables in the sense that we consider also rules with flipped variables without explicitly writing them down. Straight ground path rules can be divided into *cyclic rules* with $c_0 = c_{n+1}$ and *acyclic rules* with $c_0 \neq c_{n+1}$. We argue that any useful generalization from a straight ground path rule of length n, which is not also a generalization of a shorter path rule or a generalization of a path rule that is not straight, belongs to one of the three types $\mathbf{AC_1}$, $\mathbf{AC_2}$ or $\mathbf{C}$ defined below. We use X and Y for variables that appear in the head, while A_i is a variable that appears in the body only.

$$\begin{array}{lr} \mathbf{AC_1} & h(c_0, X) \leftarrow b_1(X, A_2), ..., b_n(A_n, c_{n+1}) \\ \mathbf{AC_2} & h(c_0, X) \leftarrow b_1(X, A_2), ..., b_n(A_n, A_{n+1}) \\ \mathbf{C} & h(Y, X) \leftarrow b_1(X, A_2), ..., b_n(A_n, Y) \end{array}$$

$\mathbf{AC_2}$ rules are generalizations of <u>a</u>cyclic ground path rules, $\mathbf{C}$ rules are generalizations of <u>c</u>yclic ground path rules, while $\mathbf{AC_1}$ rules can be generalized from both cyclic (with $c_0 = c_{n+1}$) and acyclic ones (with $c_0 \neq c_{n+1}$).

Any rule that is more specific than rules that belong to these three types must have a constant c_k with $k < n + 1$ instead of a variable A_k. With respect to $\mathbf{AC_1}$ and $\mathbf{AC_2}$ types we have to distinguish between two cases: (1) The conjunction of body atoms $b_k(...)$ to $b_n(...)$ evaluates to true and hence they can be removed from the rule. In this case, the rule that can also be created from a shorter path of length k. It will later be clarified that we learn this rule in a previous iteration of the overall algorithm. (2) The conjunction of atoms $b_k(...)$ to $b_n(...)$ evaluates always to false, which results in a rule that never fires.

For the $\mathbf{C}$ type such a constant c_k would split the rule into two parts where one part is related to X and the other part is related to Y. Such a rule might also have been created as the generalization of a rule $h(c_0, c_1) \leftarrow b_1(c_1, c_2), ..., b_{k-1}(c_{k-1}, c_k), b_n^{-1}(c_0, c_n), ..., b_k^{-1}(c_{k+1}, c_k)$. We would thus generate a rule that has a rule body which is the conjunction of the bodies of two shorter rules.

A small subset of a knowledge graph $\mathbb{G}$ is shown in Figure 1. Suppose that we are interested in finding rules that explain why Ed speaks Dutch, which corresponds to the fact $speaks(ed, d)$. To construct useful rules, we look at all paths of length n that start at ed or d. We will later explain that we create rules of length n until a certain degree of saturation is met, before we continue with $n + 1$. Note that we allow a path to be constructed by following in- and outgoing edges. We have marked three paths starting at ed in Figure 1. Two of these paths are acyclic paths ending somewhere in the knowledge graph, while the third path is, together with $speaks(ed, d)$, cyclic. Rule (1), (2), and (3) are the bottom rules which have to be generalized.

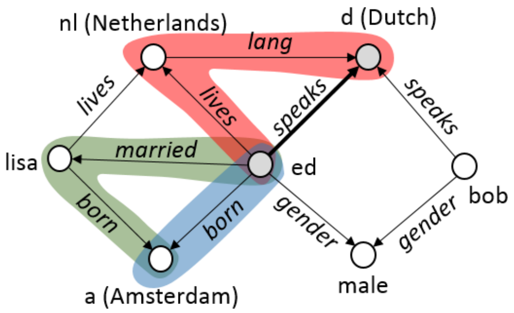

Figure 1: A small knowledge graph $\mathbb{G}$ used for sampling paths. We marked the body of Rule 1 (blue), Rule 2 (green), and Rule 3 (red).

$$speaks(ed, d) \leftarrow born(ed, a) \tag{1}$$

$$speaks(ed, d) \leftarrow married(ed, lisa), born(lisa, a) \tag{2}$$

$$speaks(ed, d) \leftarrow lives(ed, nl), lang(nl, d) \tag{3}$$

We argued above that any useful rule, which meets the criteria mentioned above, is of one of the three types. Thus, we can directly instantiate these types instead of building up a complete generalization lattice. We list in the following all rules that result from generalizing Rule 2 and Rule 3.

$$speaks(X, d) \leftarrow married(X, A_2), born(A_2, a) \tag{4}$$

$$speaks(X, d) \leftarrow married(X, A_2), born(A_2, A_3) \tag{5}$$

$$speaks(X, Y) \leftarrow lives(X, A_2), lang(A_2, Y) \tag{6}$$

$$speaks(X, d) \leftarrow lives(X, A_2), lang(A_2, d) \tag{7}$$

$$speaks(ed, Y) \leftarrow lives(ed, A_2), lang(A_2, Y) \tag{8}$$

Obviously, Rule 6 is more general than Rule 7. This means that Rule 6 fires always if Rule 7 fires. However, the rules might have different confidence scores. For that reason it makes sense to use both rules in the prediction phase.

The confidence of a rule is usually defined as number of body groundings, divided by the number of those body groundings that make the head true. Note that we count in terms of head groundings, e.g., we count the number of different $\langle X, Y \rangle$ groundings with respect to Rule 6 and the number of different X groundings with respect to Rule 7. Several subtle modifications of the basic definition have been introduced to cope with missing information, e.g., PCA confidence [Galárraga *et al.*, 2013] or completeness-aware scoring functions [Tanon *et al.*, 2017]. In this paper we stick to the standard definition, with a minor modification of additive smoothing explained later. To compute the exact confidence can be costly, because it requires to do many joins depending on the number of body atoms. For that reason we only sample body groundings, for which we compute the respective head groundings. The computed confidence is thus an approximation of the correct confidence.

3 Algorithm

In the following we propose an anytime bottom-up rule learning algorithm (Algorithm 1, called AnyBURL) that uses the generalization techniques from the previous section. The basic idea of the algorithm is to sample paths of length n from a given knowledge graph $\mathbb{G}$, starting with $n = 2$. From a path of length n, the algorithm learns rules of length $n - 1$. Remember that we count the rule length in terms of body atoms, the first edge in the path corresponds to the head atom. This is done until a certain saturation sat for rules of length $n - 1$ are reached. Then n is increased by one and the algorithm continues to learn longer rules. The required saturation sat is a parameter that needs to be specified. Another parameter is the quality criteria Q which is used to decide whether or not a rule is stored. Q can be, for example, a threshold on the confidence. We use a sampling strategy to efficiently compute the confidences of a rule using the function *score*. The size of the sample can be specified as parameter s.

Algorithm 1 Anytime Bottom-up Rule Learning

```
AnyBURL($\mathbb{G}, s, sat, Q, ts$)
$n = 2$
$R = \emptyset$
loop
    $R_s = \emptyset$
    $start = currentTime()$
    repeat
        $p = samplePath(\mathbb{G}, n)$
        $R_p = generateRules(p)$
        for $r \in R_p$ do
            $score(r, s)$
            if $Q(r)$ then
                $R_s = R_s \cup \{r\}$
            end if
        end for
    until $currentTime() > start + ts$
    $R'_s = R_s \cap R$
    if $|R'_s|/|R_s| > sat$ then
        $n = n + 1$
    end if
    $R = R_s \cup R$
end loop
return $R$
```

The learning process is conducted in a sequence of time spans of length ts. Within a time span (repeat-until loop) the algorithm learns as many rules as possible by iteratively sampling random paths. Once the given time span is over, the rules found within this span are evaluated. Note that R contains all rules that have been learned in the previous time spans, R_s contains all rules found in the current time span, and R'_s contains rules found in the current time span that have also been found in one of the previous iterations. We compute the fraction $|R'_s|/|R_s|$ and if this number is above the sat parameter, we increase the path (and thus rule) length by one and continue with the overall process. The higher n, the more time spans are usually required to reach the saturation sat.

It is important to understand that a saturation of, e.g, 99% does not mean that 99% of all possible rules instantiations have been found, but that 99% of the sampling activities result in already known rules. This is caused by the fact that there are rules that leave their traces (in terms of paths) more frequently in $\mathbb{G}$. Such an unbalanced distribution makes it relatively easy to reach a high saturation, which is at the same time one of the reasons why sampling works well.

4 Rule Application

We want to learn rules that allow us to predict candidates c with $c \in \mathbb{C}$ to replace the question mark in $r(a, ?)$ with $r \in \mathbb{R}$ and $a \in \mathbb{C}$ such that $r(a, c) \notin \mathbb{G}$ is true. Given a completion task as $r(a, ?)$, we have to compute a ranking of the top-k candidates that can substitute the question mark. It would be straight forward to create such a ranking if each entity would be generated by at most one rule. We could just order the proposed entities by the confidence values of the rules that suggested them. However, an entity is usually suggested by several rules.

If we would assume that these rules are independent, we could base our decision on multiplying confidences. This prediction strategy is called the Noisy-Or aggregation. However, the underlying assumption is often not valid. Suppose we have the following three rules, where $citizen(X, Y)$ describes that X is a citizen of country Y, $city(X, Y)$ describes that city X is located in country Y, and $capital(X, Y)$ says that X is the capital of Y; $born(X, Y)$ and $died$ express that X is born in (died in) city Y.

$$citizen(X, Y) \leftarrow born(X, A), city(A, Y)\ [.84] \tag{9}$$

$$citizen(X, Y) \leftarrow born(X, A), capital(A, Y)\ [.88] \tag{10}$$

$$citizen(X, Y) \leftarrow died(X, A), city(A, Y)\ [.82] \tag{11}$$

Even though, there is no direct logical entailment between these rules, Rule 9 would entail Rule 10 if we would have background knowledge that tells us that the capital relation is more specific than the city relation. However, such background knowledge is usually not available. Moreover, if the given dataset is noisy, then the rule $city(X, Y) \leftarrow capital(X, Y)$ will have a confidence of less than 1. There is a similar but weaker dependency between $died(X, Y)$ and $born(X, Y)$ (many people die in the city where they are born) which tells us that Rule 9 and Rule 11 are also not completely independent. However, within a noisy and incomplete dataset it is impossible to distinguish between (i) Rule 9 and Rule 10, where it makes most sense to chose the maximum of the two confidences, and (ii) Rule 9 and Rule 11, where it is more appropriate to compute the marginal probability based on the independence assumption of the rules bodies .

We apply a rather simple but efficient approach. We order the candidates via the maximum of the confidences of all rules that have generated the candidates. If the maximum score of several candidates is the same, we order these candidates via the second best rule that generates them, and so on, until we find a rule that makes a difference. This approach might rank an entity, for which two independent rules fire, too low, while the ranking is not negatively affected if two rules fire that have a strong dependency. Note also that the results of this approach can be computed without grounding all relevant

	WN18	WN18RR	FB15	FB15-237	YAGO
Entities	40943	40559	14951	14505	123143
Relations	18	11	1345	237	37
Triples	141442	86835	483142	272115	1079040
Testset	5000	3134	59071	20466	5000

Table 1: Dataset characteristics.

rules. We start with high confident rules computing results of lower confident rules until the correct order of the top-k ranking is determined. We also implemented a Noise-Or and report about results. In the Noisy-Or setting we need to compute for all rules whether or not they generate a candidate, which is in many cases less efficient.

We define the confidence of a rule, roughly, as head and body groundings divided by body groundings. According to this definition a rule with many body groundings can have the same confidence as a rule with only few body groundings, e.g., $3/4 = 750/1000$. Moreover, we will have usually many rules with few groundings, especially rules with constants, and few rules with many groundings. For that reason it can happen that some rules with few groundings have a confidence that is too high. In order to circumvent this, we add a constant $p_c > 0$ to the denominator as a kind of Laplace smoothing. This can be understood as a pessimistic variant of the standard confidence value. Rules with a large number of groundings are only slightly affected, while rules with a low number of groundings get a significantly lower confidence.

5 Experiments

We used in our experiments the FB15(k) dataset, its modified variant FB15-237, WN18, its modified variant WN18RR, and YAGO03-10 (in short YAGO). The FB (WN) datasets are based on a subset of FreeBase (WordNet). FB15 and WN18 have been first used in [Bordes *et al.*, 2013]. They have been criticised in several papers: due to redundancies a large fraction of testcases can be solved by exploiting rather simple rules. Both FB15-237 [Toutanova and Chen, 2015] and WN18RR [Dettmers *et al.*, 2018] have been proposed as modified variants where redundancies have been suppressed. FB15-237 goes even a step further. It contains no test cases that can be solved with **C** rules with one body atom, even though the training set shows regularities that induce these rules. This makes FB15-237 a bit unrealistic and a hard testset for a rule based approach. The dataset YAGO has been used in [Dettmers *et al.*, 2018] as additional dataset. An overview on the datasets is given in Table 1. The numbers reported in the first three lines refer to the training set. The last line refers to the number of triples in the test set. Each test triple can be divided into two test cases. In terms of triples and entities (constants) YAGO03-10 is the largest dataset.

As in [Bordes *et al.*, 2013] we compute the filtered hits@1 and filtered hits@10. In the following we simply refer to these values without the adjective 'filtered'. We do not compute the filtered MRR (mean rank reciprocal) exactly, because our approach is not designed to compute complete rankings but top-k rankings only. Instead of computing the exact value, we assume that any candidate which would be ranked at a position >k is not a correct prediction. This results into a lower bound of the filtered MRR, which we present in Table 2 prefixed with $\geq$. The exact value is usually slightly higher.

We have named our approach AnyBURL (**Any**time **B**ottom-**U**p **R**ule **L**earning). AnyBURL is written in Java and requires no external libraries. The source code and datasets, used in the experiments, can be found at http://web.informatik.uni-mannheim.de/AnyBURL/. We conducted all our experiments on a virtual machine with 4 cores (each 2400 MHz) and 16 GB RAM.

We have not made use of the validation sets to find dataset specific parameter settings. For all our experiments we used exactly the same parameter setting. We have chosen the quality criteria Q to allow only those rules that generate at least two correct predictions, which is a very lax criteria. We have set the required saturation rate sat to 99%. We set $ts = 1$ second, $p_c = 5$, and $s = 500$, which is the sample size that determines the precision of the confidence value. As default setting we used the maximum aggregation strategy. We have also conducted an experiment where we compared this strategy against a Noisy-Or aggregation. We have made two further modifications. Our implementation alternates between time spans where we sample only cyclic paths; and time spans where we sample all possible paths. This enables to use an algorithm that finds cycles more efficiently. Further, we have stopped explicitly searching cyclic paths once we reached the defined saturation for **C** rules of length 3, because longer cyclic rules can be very costly in terms of runtimes.

5.1 Anytime Behavior and Runtime

We have to distinguish between the time required to learn the rules and the time of applying them to solve the task. Obviously, the more time we spent in learning, the more rules we learn; and the more rules we learn, the longer will be the time required to apply these rules. Hits@1 and hits@10 results are specified for all five datasets in Table 2 based on the rule sets that have been learned after 10, 100, 1000, and 10000 seconds. For all datasets we observe a similar pattern. The results are already surprisingly good after a 10 seconds learning phase. For the smaller datasets (WN18 and WN18RR) these results are already as good as current state of the art results. When we learn rules for 100 and 1000 seconds (which is less than half an hour) the results are further improved, and this improvement is higher for larger datasets.

Learning rules for a long time (10000 seconds) can have a slightly negative impact on the results. This is especially the case for the datasets WN18 and FB15. These datasets are dominated by redundancies which are reflected in **C** rules of length 1. These rules can be found quickly at the beginning. If we learn longer, it might happen that there are some very specific rules among the very large set of learned rules that create noise. However, this trend does not continue if we run the tool even longer. We have also run AnyBURL on WN18 and FB15 for 20000 seconds and there were nearly no differences compared to the results reported for the 10000 seconds run (for instance, WN18 changed from 95.42 to 95.40).

For YAGO we show (i) the impact of the learning time on the results quality (y-axis on the left), (ii) the time required for applying the learned rules (projection to x-axis), and (iii) the

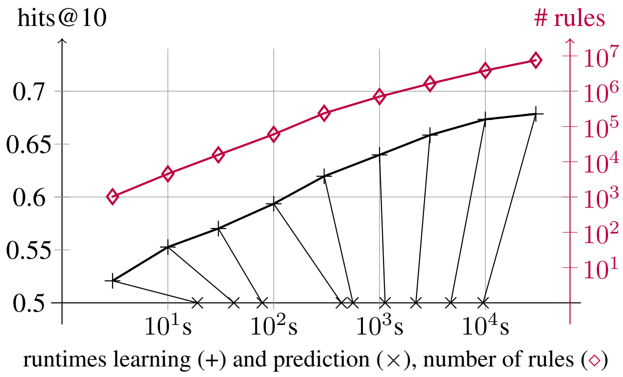

Figure 2: Anytime Behaviour of AnyBURL on the YAGO dataset.

number of rules that have been learned (purple line, y-axis on the right). The application time, which refers to the full test set, is shown as projection to the x-axis. For example, learning rules for 10 seconds, results in a hits@10 score of 55.3% and a rule set which consists of 4531 rules. It requires 42 seconds to apply this rule set to the 5000 test cases of the dataset. Note that the x-axis and the y-axis on the right use a logarithmic scale.

Figure 2 shows that AnyBURL starts at a high level and improves the results in terms of a logarithmic function. The set of learned rules seems to grow nearly linear with a slight reduction the longer we run AnyBURL. AnyBURL reaches a saturation of 0.99 for $\mathbf{C}$ rules of length 3 already after 84 seconds, while the saturation for $\mathbf{AC_1}$ and $\mathbf{AC_2}$ rules of length 1 is not reached within the overall time span of 30000 seconds. This is obviously caused by the high number of constants in the YAGO dataset. This indicates that the good results achieved in short time are mostly based on $\mathbf{C}$ rules, while the small but steady improvements that follow are based on $\mathbf{AC_1}$ and $\mathbf{AC_2}$ rules.

The time for applying the rules is for small rule sets higher than the time required for learning them on the whole test set. However, if the rules set gets larger less additional time for the prediction is required. A rule set that is learned in 1000 seconds, requires also around 1000 seconds to be applied. For larger rules sets the learning time dominates the application time. Note also that the application time is smaller for the four other datasets.

5.2 Comparison with Other Approaches

The first block in Table 2 lists the results of current state of the art completion techniques using embeddings. We have selected five models, which achieved very good results and have been published within the last year at top conferences. The results that AnyBURL achieves after a 1000 seconds learning phase are at the same level and sometimes slightly better than most of these models. AnyBURL has, for example, better hits@1, hits@10, and MMR results than ConvE [Dettmers *et al.*, 2018] on all datasets except FB15-237, where the results of AnyBURL are only slightly worse. AnyBURL outperforms SimplE [Kazemi and Poole, 2018] already when using the rules that have been learned after 10 seconds.

Exceptionally, the ComplEx-N3 (reciprocal) model proposed in [Lacroix *et al.*, 2018], originally introduced in [Trouillon *et al.*, 2016], achieves better results than any other model including AnyBURL. This model requires 485 seconds × 25 epochs (100 epochs) = 12125 (48500) seconds for a *single* run with a fixed set of hyper parameters on YAGO. Around half the time is required for FB15.[1] The hyper parameters are determined in a grid-search over 28 combinations, which requires in the worst case to multiply by 28, i.e. 28 × 12125 = 339500 $\approx$ 4 days, even though an early stopping criteria might save effort. While we ran AnyBURL on a standard laptop, the ComplEx-N3 runtimes are based on the use of a Quadro GP100 GPU.

The second block in Table 2 presents results for two alternative rule learning systems called RuleN and AMIE+. AMIE+ uses a complete top down approach and a similar language bias as RuleN. Within its language bias it constructs all rules of a certain length using a support threshold as a pruning mechanism. RuleN learns $\mathbf{C}$ rules and a rather specific type of constant rules. It uses sampling techniques that are similar to but less general than the approach we proposed. Both systems have been executed in [Meilicke *et al.*, 2018] in an environment similar to ours. None of the systems has an anytime behaviour, however, the parameters have been chosen dataset specific to exploit a time frame of 10 hours as good as possible. Thus, the results are roughly comparable to the 10000s results of AnyBURL. With the minor exception of the WN18 dataset, AnyBURL generates significantly better results than AMIE and RuleN. For RuleN this is partially related to its inability to learn certain rules. The improvements over AMIE are probably related to the sampling strategy used within the bottom-up approach. RLvLR [Omran *et al.*, 2018] is a rule learner that uses embeddings to guide the rule extraction process in order to reduce the search space. For this approach results are only available for FB15-237, where RLvLR achieves hits@10 of 39.3% and an MMR of 0.24. These results are similar to the results of AMIE and RuleN.

In the last row we compare the default Maximum aggregation against the Noisy-Or showing the difference Noisy-Or minus Maximum. The numbers are based on applying the rules learned after 1000 seconds. Noisy-Or performs on nearly all datasets clearly worse. We observe the highest drop on the datasets WN18 and FB15. Both datasets contain many redundancies and as a result a candidate might be proposed by several highly dependent (equivalent) rules, which distorts the generated candidate ranking.

6 Related Work

We discussed AMIE and RuleN in the section above. RuDiK [Ortona *et al.*, 2018], an example for another rule learner, can learn both positive and negative rules (constraints). Contrary to our approach, RuDiK is designed to find a small set of rules that cover the majority of positive and as few negative examples as possible. This differs from our objective, where we try to learn every possible rule that might be relevant for creating a top-k candidate ranking.

[1] Numbers reported in https://github.com/facebookresearch/kbc.

	WN18			WN18RR			FB15			FB15-237			YAGO03-10		
Approach	h@1	h@10	MRR	h@1	h@10	MRR	h@1	h@10	MRR	h@1	h@10	MRR	h@1	h@10	MRR
SimplE [Kazemi and Poole, 2018]	93.9	94.7	94.2				66.0	83.8	72.7						
ConvE [Dettmers *et al.*, 2018]	93.5	95.5	94.2	39	48	46	67.0	87.3	74.5	23.9	49.1	31.6	45	66	52
ComplEx-N3 [Lacroix *et al.*, 2018]		96	95		57	48		91	86		56	37		71	58
R-GCN+ [Schlichtkrull *et al.*, 2018]	69.7	96.4	81.9				60.1	84.2	69.6	15.1	41.7	24.9			
CrossE [Zhang *et al.*, 2019]	74.1	95.0	83.0				63.4	87.5	72.8	21.1	47.4	29.9			
AMIE+ [Galárraga *et al.*, 2015]	87.2	94.8		35.8	38.8		64.7	85.8		17.4	40.9				
RuleN [Meilicke *et al.*, 2018]	94.5	95.8		42.7	53.6		77.2	87.0		18.2	42.0				
RLvLR [Omran *et al.*, 2018]											39.3	24			
AnyBURL, 10s	94.2	94.9	≥94	43.2	52.7	≥46	79.6	83.8	≥81	13.4	25.9	≥17	33.4	55.3	≥40
AnyBURL, 100s	94.6	95.9	≥95	44.5	54.9	≥48	80.8	87.6	≥83	19.6	41.0	≥26	37.6	59.4	≥44
AnyBURL, 1000s	93.9	95.6	≥95	44.6	55.5	≥48	80.4	89.0	≥83	23.0	47.9	≥30	42.9	63.9	≥49
AnyBURL, 10000s	93.5	95.4	≥94	44.1	55.2	≥47	79.6	88.7	≥82	23.3	48.6	≥31	47.7	67.3	≥54
Δ Noisy-Or vs. Max Aggregation	-12.5	-1.7	-9	-8.8	-0.3	-6	-25.5	-9.4	-10.1	-1.1	+/-0	-0.6	-1.2	+0.1	+/-0

Table 2: Comparing our approach against current state of the art results. The runtimes of AnyBURL refer to rule learning only. Results have been taken from the publications listed in the first column. An exception are the results for AMIE+ published in [Meilicke *et al.*, 2018].

The Path Ranking Algorithm [Lao *et al.*, 2011] is based on sampling paths that correspond to $\mathbf{C}$ rules. While the sampling technique of AnyBURL is inspired by this approach, AnyBURL can learn more expressive rules than PRA. In [Lao *et al.*, 2015] the authors proposed an extension, that can also deal with instances (constants). However, the rules that are supported by this extension are similar to the conjunction of $\mathbf{AC_1}$ rules of length 1 and $\mathbf{C}$ rules of arbitrary length, while $\mathbf{AC_1}$ and $\mathbf{AC_2}$ on their own are, upon our understanding, not supported. We have argued above that such a conjunction would result in a significantly larger search space, which we try to avoid with a strict language bias.

Another interesting approach is Gaifman models [Niepert, 2016], where neighbourhoods are sampled instead of paths. While this allows in principle to learn every possible rule (if the neighborhood is large enough), the application that the authors report about is based on a rather restrictive language bias that covers only a small fraction of the rules that we learn. We introduced a language bias and an efficient sampling technique that is tightly coupled, while sampling in the context of Gaifman models is a means to create examples that fit (in principle) against any language bias.

The majority of existing approaches for knowledge graph completion are based on the concept of embeddings. There is also a family of approaches that try to combine embeddings and rules (see [Wang *et al.*, 2017] for a survey). An example is RLvLR [Omran *et al.*, 2018], for which we added some results above, and the system Ruge [Guo *et al.*, 2018], which uses learned rules to inject new training examples with soft labels into the process of learning the embedding. The authors report about results for FB15, which are worse than the AnyBURL 100 seconds learning results. We believe that the benefits of combining rules and embeddings can only be understood, if we know first how far one can get with each method on its own. With our work, we show that rules on their own perform surprisingly well, which should not be neglected in further work on combining embeddings and rules.

7 Conclusion

We have proposed an anytime algorithm for learning rules from knowledge graphs by following the bottom-up paradigm. We applied our approach, called AnyBURL, to five different and frequently used datasets. AnyBURL generates in short time results that are as good and better than many recent state of the art techniques, while using only limited resources (both in terms of memory and computational power). Due to sampling, the algorithm learns within the chosen language bias rules with high support first. Opposed to this, a top down approach as AMIE has to learn all rules of a certain type to ensure that these rules are covered. Compared to latent representation models AnyBURL has several advantages in common with other rule based approaches:

- The candidate ranking can be explained in terms of the rules that generated this ranking. Such an explanation is easy to understand and can even be related to statistical evidence, e.g., *Ed speaks Dutch, because he is married to someone who is born in Amsterdam, and people married to someone born in Amsterdam speak Dutch with a confidence of 82%. Dutch is the highest ranked candidate for the language that Ed speaks. English is on the second rank with a confidence of 67%, because ...*
- The generated model (= rule set) can be partially reused for a dataset using the same predicates and an overlapping set of constants. Rules without constants can be definitely reused, and if (frequently used) constants like *london* or *female* appear also in the new dataset, rules that contain such constants can also be reused.
- A rule based approach does not require to learn dataset specific hyper parameters. We know that AnyBURL has also several parameters that have an impact on its behaviour. However, the impact of these parameters is transparent and there are less dependencies with dataset specific characteristics. Note that we run AnyBURL with the same parameter setting on all five datasets.

While these advantages are well known, their recognition had only a limited effect in motivating current research to develop symbolic approaches for knowledge graph completion. One reason might have been the prejudice that rule learning cannot be applied to larger datasets efficiently. With our results we have shown that the opposite is the case.

References

[Bordes *et al.*, 2013] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[Dettmers *et al.*, 2018] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[Galárraga *et al.*, 2013] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*, pages 413–422. ACM, 2013.

[Galárraga *et al.*, 2015] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. Fast rule mining in ontological knowledge bases with AMIE+. *The VLDB Journal—The International Journal on Very Large Data Bases*, 24(6):707–730, 2015.

[Guo *et al.*, 2018] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Knowledge graph embedding with iterative guidance from soft rules. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[Kazemi and Poole, 2018] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *Advances in Neural Information Processing Systems*, pages 4289–4300, 2018.

[Lacroix *et al.*, 2018] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In *ICML*, pages 2869–2878, 2018.

[Lao *et al.*, 2011] Ni Lao, Tom Mitchell, and William W Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 529–539. Association for Computational Linguistics, 2011.

[Lao *et al.*, 2015] Ni Lao, Einat Minkov, and William Cohen. Learning relational features with backward random walks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, volume 1, pages 666–675, 2015.

[Meilicke *et al.*, 2018] Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. Fine-grained evaluation of rule- and embedding-based systems for knowledge graph completion. In *Proceedings of the International Semantic Web Conference*, pages 3–20. Springer International Publishing, 2018.

[Muggleton and Feng, 1990] Stephen H. Muggleton and Cao Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, pages 368–381, 1990.

[Niepert, 2016] Mathias Niepert. Discriminative gaifman models. In *Advances in Neural Information Processing Systems*, pages 3405–3413, 2016.

[Omran *et al.*, 2018] Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. Scalable rule learning via learning representation. In *IJCAI*, pages 2149–2155, 2018.

[Ortona *et al.*, 2018] Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. Robust discovery of positive and negative rules in knowledge bases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1168–1179. IEEE, 2018.

[Schlichtkrull *et al.*, 2018] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[Srinivasan, 2000] Ashwin Srinivasan. The aleph manual(techical report). Technical report, Computing Laboratory, Oxford University, 2000.

[Tanon *et al.*, 2017] Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. Completeness-aware rule learning from knowledge graphs. In *International Joint Conference on Artificial Intelligence*, pages 507–525. Springer, 2017.

[Toutanova and Chen, 2015] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, 2015.

[Trouillon *et al.*, 2016] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pages 2071–2080, 2016.

[Wang *et al.*, 2017] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[Zhang *et al.*, 2019] Wen Zhang, Bibek Paudel, Wei Zhang, Abraham Bernstein, and Huajun Chen. Interaction embeddings for prediction and explanation in knowledge graphs. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 96–104. ACM, 2019.