The VLDB Journal (2024) 33:131–161
https://doi.org/10.1007/s00778-023-00800-5

REGULAR PAPER

Anytime bottom-up rule learning for large-scale knowledge graph completion

Christian Meilicke[1] · Melisachew Wudage Chekol[2] · Patrick Betz[1] · Manuel Fink[1] · Heiner Stuckeschmidt[1]

Received: 11 August 2022 / Accepted: 23 May 2023 / Published online: 16 June 2023

Abstract
Knowledge graph completion is the task of predicting correct facts that can be expressed by the vocabulary of a given knowledge graph, which are not explicitly stated in that graph. Broadly, there are two main approaches for solving the knowledge graph completion problem. Sub-symbolic approaches embed the nodes and/or edges of a given graph into a low-dimensional vector space and use a scoring function to determine the plausibility of a given fact. Symbolic approaches learn a model that remains within the primary representation of the given knowledge graph. Rule-based approaches are well-known examples. One such approach is AnyBURL. It works by sampling random paths, which are generalized into Horn rules. Previously published results show that the prediction quality of AnyBURL is close to current state of the art with the additional benefit of offering an explanation for a predicted fact. In this paper, we propose several improvements and extensions of AnyBURL. In particular, we focus on AnyBURL's capability to be successfully applied to large and very large datasets. Overall, we propose four separate extensions: (i) We add to each rule a set of pairwise inequality constraints which enforces that different variables cannot be grounded by the same entities, which results into more appropriate confidence estimations. (ii) We introduce reinforcement learning to guide path sampling in order to use available computational resources more efficiently. (iii) We propose an efficient sampling strategy to approximate the confidence of a rule instead of computing its exact value. (iv) We develop a new multithreaded AnyBURL, which incorporates all previously mentioned modifications. In an experimental study, we show that our approach outperforms both symbolic and sub-symbolic approaches in large-scale knowledge graph completion. It has a higher prediction quality and requires significantly less time and computational resources.

Keywords Knowledge graph completion · Link prediction · Rule learning

1 Introduction

A knowledge graph is a formal representation of a certain domain. In a knowledge graph, nodes represent entities, such as people, places, organizations, or abstract entities such as genders or nationalities. These entities are connected by labelled directed edges. A label corresponds to a binary predicate, which is also referred to as relation. A pair of entities (a, b) that is connected via an edge with label p corresponds to a fact $p(a, b)$ also called triple. Thus, a knowledge graph can be understood as a set of facts that are grounded binary predicates.

Knowledge graphs are widely employed in various domains. Some examples are Freebase [7], DBPedia [1], YAGO [54], Google Knowledge Graph, and Microsoft Satori [40]. These massive graphs can contain up to millions of entities and billions of facts. As pointed out in [17], knowledge graphs are often incomplete. The task of constructing missing triples in such a graph is known as *knowledge graph completion* or *link prediction*. This task can be solved with the help of external resources (e.g., text in web-pages) or by inferring new triples solely from the already existing triples in the given graph. We are concerned with the latter problem.

✉ Christian Meilicke
christian@informatik.uni-mannheim.de

Melisachew Wudage Chekol
m.w.chekol@uu.nl

Patrick Betz
patrick@informatik.uni-mannheim.de

Manuel Fink
manuel@informatik.uni-mannheim.de

Heiner Stuckeschmidt
heiner@informatik.uni-mannheim.de

[1] University Mannheim, Mannheim, Germany

[2] Utrecht University, Utrecht, Netherlands

An approach that does not use external information must rely on the statistics, patterns, distributions or any other kind of regularity that can be found in the graph. An intuitive choice for solving such a task is to learn and apply an explicit, symbolic representation of these patterns. While there is a long history of approaches that are concerned with learning symbolic representations, such as inductive logic programming [37] and relational association rule mining [15], today's research is following a different paradigm. The vast majority of methods that are developed nowadays learn a low-dimensional, sub-symbolic representation of entities and/or relations that appear in the given graph [47]. As a result, symbolic approaches are underrepresented in knowledge graph completion research.

As an exception to this general trend in [35] a symbolic approach has been proposed, which has been specifically tailored for the task of knowledge graph completion. This approach is called AnyBURL (Anytime Bottom-up Rule Learning) due to its anytime behavior and the fact that it is based on a sampling component that generalizes paths into rules. The results available in [35] as well as the results reported in an independent evaluation of the current state of the art [47] revealed that AnyBURL is not just a symbolic baseline, but performs on the same level as the best models proposed in the last five years.

Symbolic approaches are not necessarily restricted to rule-based methods only. Other symbolic approaches focus on the notion of paths [13] or on the concept of nearest neighbors [21]. One of the most important challenges that a rule-based approach needs to solve is to determine which types of rules are supported by the approach. We refer to the set of supported rules as the language bias of a rule-based approach. The language bias determines which regularities can be grasped and which regularities are ignored. Contrary to this, a latent approach does not need to list certain types of rules or regularities that can be detected by the approach. As a contribution of our work, we show that it is possible to explicitly define a language bias that can be efficiently used to achieve top results on very large knowledge graph completion problems.

The increased focus on sub-symbolic representations in the past decade, which was mainly driven by the flexibility and predictive quality achievable with end-to-end learning, somewhat conversely motivated the revival of the usage of symbolic methods due to an urgent need for *explainability*. Along these lines, it has been shown that AnyBURL can be utilized to explain predictions made by a latent model when restricting the types of learned rules [5]. Moreover, a symbolic model, when performing competitively in regard to predictive quality, represents a standalone alternative to a latent model as it is inherently explainable. For instance, every prediction obtained by a rule-based model can be backtracked to the respective symbolic rule that made the prediction and is therefore fully explainable. This is a huge advantage compared to approaches that are based on latent representations.

Another advantage of a rule-based approach is its applicability to both transductive and inductive scenarios. In an transductive setting, the set of entities, for which we want to derive new facts, is described in a given knowledge graph that we use to learn a model. In an inductive setting, we might also be interested to predict missing information for new entities not seen during the training phase. These entities might be described in a new set of observations that have emerged after the training phase. In such a setting, the rules that have been learned in the training phase can still be applied to make predictions, while, for example, standard knowledge graph completion approaches cannot be applied directly in such a setting [59].

In this paper, we further improve AnyBURL and report the impact of four major modifications.

Object identity While the previous version of AnyBURL does not utilize Object Identity [51], we take up the concept in Sect. 3.3 and report about experiments that illustrate its benefits w.r.t knowledge graph completion (Sect. 6.1). Our results show that it prevents learning a large number of quasi-redundant rules with misleading confidence scores. The basic idea of this extension is rather simple, as it adds to each rule a set of constraints, which requires that the groundings of different variables are different entities. We show that these constraints prevent the learning of problematic rules, which have an overrated confidence score. The removal of these rules increases the predictive quality and reduces the size of the learned rule set.

Confidence sampling We explain in Sect. 3.4 why it is especially important with respect to very large datasets to avoid a depth-first search (which has previously been used) when drawing samples for computing the confidence of a rule. The previously implemented method is problematic if the search tree that is constructed grounds a variable with a hub entity, e.g., `female`. In such a situation, it might happen that the chosen sampling bases its confidence on women only, while the rule captures a general regularity about any kind of person (independently of the gender of that person). Thus, we would learn a rule about persons, but use the statistics of the subgroup of women to compute its confidence. We propose a different sampling procedure that does not run into such problems.

Reinforcement learning-based path sampling We introduce reinforcement learning in Sect. 4 to guide the search during sampling paths. We argue and show that reinforcement learning is more robust, allows to leverage the characteristics of a given knowledge graph, and is less

negatively affected by choosing a wrong parameter setting. AMIE [22], for example, constructs and scores all rules of length n before it continues with rules of length $n+1$. The previous version of AnyBURL [35] uses a similar approach. Instead of mining all rules of a certain length, it learns nearly all rules based on a completeness estimation and a threshold which is set manually. Both approaches are problematic as too much time might be lost for rules of length n, while important longer rules of length $n+1$ have not yet been constructed. We propose instead to construct rules of different lengths in parallel guided by a reinforcement learning paradigm.

Multithreading In contrast to the previous AnyBURL version which runs single-threaded, we argue in this work that the rule mining process can be divided over multiple threads efficiently (Sect. 4.5). This is an important insight as it is a distinguishing characteristic compared to approaches that learn knowledge graph embeddings for which parallelization is non-trivial. Different parallelization techniques and their drawbacks have been discussed in [27]. The multithreaded implementation allows to run AnyBURL in acceptable runtimes on very large graphs, and we show, for example, that the new version achieves a higher predictive performance 20 times faster than the current state of the art.

Within this paper, we focus mainly on these improvements. However, we discuss also the language bias and the basic rule sampling procedure of AnyBURL, which has previously been introduced in [35].

As a key contribution of this paper, we evaluate our approach in an extensive experimental study on four large to very large datasets. The largest of these datasets is Freebase. It comprises more than 300 million triples that describe around 86 million entities. Our symbolic approach clearly outperforms the current state of the art on the two largest datasets in terms of predictive quality, runtimes, and CO_2 emission. Thus, our approach is, according to our experimental results, the best knowledge graph prediction approach for very large datasets.

2 Knowledge graph completion

In the following, we introduce the problem of knowledge graph completion and explain the standard evaluation method. This includes also the usual splitting of the evaluation datasets. We explain also why and how rules can be used to make a prediction for a given completion task following the intuitive approach that has already been used in [35]. This section does not contain a novel contribution but compiles some preliminary knowledge that helps to better understand and contextualize the content of the following sections.

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, $\mathbb{G} \subset \{p(a,b) \mid p \in \mathbb{R},\ a,b \in \mathbb{C}\}$ is a set of ground atoms or facts. Given a fact $p(a,b)$, we call a the subject and b the object of this fact. Sometimes we use the term triple to refer to a fact, which seems to be more common in the field of knowledge graph completion. A binary predicate is also called a relation. Besides, a constant (or what the constant refers to) is also called entity.

We call the problem of constructing missing triples in a given knowledge graph the knowledge graph completion problem. In real-world scenario, we might be aware that a graph $\mathbb{G}$ is incomplete and the task might be to find every missing triple or to list as many missing triples without making too many mistakes. We might also think of a more focused setting, where it is known that, for example, every person must have a nationality and the task is to specify the nationality of every person, for which a nationality is not yet known.

In a real-world setting, the proposals made by a knowledge graph completion method might be checked by a human expert, before some of them are added to $\mathbb{G}$ as new triples. Thus, it is important that the method can create candidate rankings such that a domain expert can look at the top-k part of the ranking only. If we ask for the nationality of a specific person a possible top-3 ranking might look like this: *(1) Nigerian [0.72], (2) French [0.68], (3) Dutch [0.23]*. The number in parentheses could be a confidence value, a probability, or any kind of score that has been used to determine the order of the ranking.

In an evaluation scenario, there is usually no domain expert or oracle available that tells us whether the prediction of a knowledge graph completion method is correct or incorrect. The performance of a knowledge graph completion method is approximated in an artificial and controlled setting that mimics the scenario described above. A given knowledge graph $\mathbb{G}$, which is used as an evaluation dataset, is split into training $\mathbb{G}_t$, validation $\mathbb{G}_v$ and test (or evaluation) set $\mathbb{G}_e$. This partition is a disjoint split in terms of triples; however, the split is not disjoint with respect to constants and relations, i.e., we have $\mathbb{G}_t \subset \{p(a,b) \mid p \in \mathbb{R}_t,\ a,b \in \mathbb{C}_t\}$ and $\mathbb{G}_e \subset \{p(a,b) \mid p \in \mathbb{R}_e,\ a,b \in \mathbb{C}_e\}$ with $\mathbb{G}_t \cap \mathbb{G}_e = \emptyset$, $\mathbb{C}_e \subseteq \mathbb{C}_t$ and $\mathbb{R}_e \subseteq \mathbb{R}_t$. The disjointness relation also holds between validation and training set.

A knowledge graph completion method, that is evaluated in a research context, uses the training set to learn a representation of (some aspects of) the dataset. This representation is sometimes called a model. For a rule-based approach, this will be a set of rules. These rules reflect the regularities and distributions of the training set. Due to the shared vocabulary, more precisely due to $\mathbb{R}_e \subseteq \mathbb{R}_t$ and $\mathbb{C}_e \subseteq \mathbb{C}_t$ these rules can make predictions for the test cases that are defined in the test set.

We call a triple in the test set a test triple, and such a test triple results into two test cases. Suppose we have a test triple *profession*(*mozart*, *composer*). This test triple results into the following two test queries Q_1 and Q_2, which ask for the profession of *mozart* and for people that have *composer* as profession.

(Q_1) *profession*(*mozart*, ?)
(Q_2) *profession*(?, *composer*)

Such a query is answered in terms of a ranking of possible candidates. As the test case is derived from a known triple, we know in the evaluation context the correct answer, which is for the first test case *composer* and for the second test case *mozart*.

The common evaluation metrics, which we use in our experiments in Sects. 5 and 6, are filtered hits@k and filtered mean reciprocal rank (MRR). For a test triple $p(s, o)$ let $\mathbf{rk}(o|s, p)$ be the position of the target candidate o in a ranking of candidates for the query $p(s, ?)$ and likewise let $\mathbf{rk}(s|p, o)$ be the position of target candidate s in the ranking for $p(?, o)$. Note that for the filtered metrics, in the respective rankings, a non-target candidate is suppressed if the resulting triple exists in $\mathbb{G}_t$, $\mathbb{G}_v$, or $\mathbb{G}_e$. MRR and Hits@k are defined as

$$MRR = \frac{1}{2|\mathbb{G}_e|} \sum_{p(s,o)\in\mathbb{G}_e} \left(\frac{1}{\mathbf{rk}(o|s,\, p)} + \frac{1}{\mathbf{rk}(s|p,\, o)} \right),$$

$$hits@k = \frac{1}{2|\mathbb{G}_e|} \sum_{p(s,o)\in\mathbb{G}_e} \Big(\mathbb{1}\{\mathbf{rk}(o|s,\, p) \le k\} + \mathbb{1}\{\mathbf{rk}(s|p,\, o) \le k\} \Big),$$

with $\mathbb{1}\{cond\}$ being an indicator that is one if the condition inside {.} is true and zero otherwise. The hits@k scores measure how often the correct answer, which is defined by the triple that the test case originates from, is among the top-k ranked entities, whereas the MRR takes into account all positions. In the following, we always refer to the filtered scores, as defined in [9], without explicitly stating it.

So far, we explained the purpose of the training set $\mathbb{G}_t$ and the test set $\mathbb{G}_e$. The validation set $\mathbb{G}_v$ has a specific purpose. It is required only for the class of methods for which performance depends to a high degree on the chosen hyperparameter setting. Such methods test hyperparameter settings systematically against the validation set. This process is called hyperparameter search. The hyperparameter setting that performs best is then used to train a model on the training set, which is finally used to predict rankings for the test queries, which are defined by the triples in the test set. The hyperparameter search requires often significantly more time than running the best setting that has been finally found. A rule-based approach uses typically a fixed hyperparameter setting and does not make any use of the validation set aside from filtering the results, which is part of the standard evaluation protocol. However, as many models require a validation set, the standard evaluation protocol is always defined on a fixed split into training, validation and test set. Every approach that wants to compare its results to other approaches needs to adhere to this split.

We now explain how rules are used to create a ranking for a test query, which can then be evaluated in terms of MRR and hits@k as explained above. The rules that we are concerned with are strict Horn rules. Given a rule $r = r_h \leftarrow r_b$, we call r_b the body of the rule and r_h the head of the rule. The head of the rule is a single atom; the body of a rule is a conjunction of atoms, which we separate by the symbol $\wedge$ or by comma. An atom has the form $r(t_1, t_2)$ where r is a relation with $r \in \mathbb{R}_t$. t_1 and t_2 are terms that are either constants from $\mathbb{C}_t$ or variables. We use lower-case letters for constants and upper-case letters for variables. We define the types of rules that are supported by our approach more precisely in the next section.

We use the symbol θ to refer to a substitution, which replaces a variable by a constant not explicitly specified. A substitution $\theta_{X=c}$ refers to a substitution that replaces X in a rule or in a part of a rule by a constant c. Sometimes we call such a substitution a grounding of X. Given a rule r, an expression such as $r\theta_{X=c}$ refers to a (partially) grounded rule where all occurrences of X are replaced by c. Note that we will often omit the constant, especially when we talk about all possible groundings or about the existence of a grounding for a certain variable.

We can now easily explain how we compute the answer to a query, like for example query Q_1. Suppose now that we have the following rule r given in (1), where *inf* refers to a relation which describes that a subject influenced an object.

$$profession(X, Y) \leftarrow inf(X, A), profession(A, Y) \qquad (1)$$

To compute an answer to the given query, we look at the partially grounded rule $r\theta_{X=mozart}$. The answer to our query is the set $\{y \mid r_b\theta_{X=mozart, A=a, Y=y} \in \mathbb{G}_t\}$ with $a, y \in \mathbb{C}$. Note that we do not need to iterate over all possible a and y values, but conduct a depth-first search starting from *mozart*, where each step in the tree takes constant time due to the index structures that AnyBURL created during preprocessing. For our specific example, the result set will probably contain several professions such as, for example, *musician*, *artist*, and hopefully also *composer*. Each of these candidates is annotated with the confidence of the rule, which will be introduced later as a quality measure of the rule.

Usually, there will be many rules that yield non-empty results sets for a given query. Moreover, these result sets might overlap. In the case of overlapping result sets, the can-

didate is annotated with the confidence of the rule that has the highest confidence among the rules that yield this candidate. The final ranking is defined by the order implied by the confidence scores of all candidates. To aggregate rules based on their max ranking is the most common practice which we already used successfully in [35]. While there are also other approaches to aggregate rules [44], which have shown to perform slightly better, we use the standard maximum-based approach, which seems to make more sense in a large-scale setting. It has also the advantage that we can, for each completion query, start with the application of the rule with highest confidence. We continue with the second best, and so on, until we collected the desired number of candidates and until potential ties in a ranking are resolved by comparing the respective candidates by their second strongest rule. In our experiments, we set this number to 100 candidates. Thus, our hits@k score is correct up to k=100, and our MRR would be slightly better, if we would construct longer rankings.[1]

The prediction phase is not a main topic in this paper. Instead, we are concerned with the construction of rules and the computation of their confidence scores, which happens both in the rule learning phase. More details on different strategies to create a ranking can be found, for example, in [6].

3 Bottom-up rule learning

In this section, we first introduce the type of rules that can be learned by AnyBURL before we describe how we create these rules from given sampled paths. Parts of this were already presented in a different form in [35]. As a novel contribution, we explain the concept of Object Identity that was introduced in [51] and argue why it is important for rule-based knowledge graph completion. Finally, we argue that the sampling technique used for approximating rule confidence used in the previous version of AnyBURL sometimes yields distorted confidence scores and we show how to fix this problem.

3.1 Language bias

We distinguish in the following between three types of rules that we call binary rules (**B**), unary rules ending with a dangling atom ($\mathbf{U_d}$) and unary rules ending with an atom that includes a constant ($\mathbf{U_c}$).[2] In the formulas listed below, we use h and b_i to refer to relations (binary predicates), we use an upper case letter with or without subscript to refer to variables, and c or c' to refer to constants, which are also called entities. When we discuss examples, we will also use words as *married* or *speaks* to refer to relations and proper names as *john* to refer to constants. We define an atom as a formula that uses a binary predicate to express the relationship between two terms, which can be a variable or constant. For example, $h(X, Y)$, $gender(X, female)$, and $married(jane, john)$ are atoms.

In our approach, we support the following three types of rules:

$$\mathbf{B} \qquad h(A_0, A_n) \leftarrow \bigwedge_{i=1}^{n} b_i(A_{i-1}, A_i)$$

$$\mathbf{U_d} \qquad h(A_0, c) \leftarrow \bigwedge_{i=1}^{n} b_i(A_{i-1}, A_i)$$

$$\mathbf{U_c} \qquad h(A_0, c) \leftarrow \left(\bigwedge_{i=1}^{n-1} b_i(A_{i-1}, A_i)\right) \wedge b_n(A_{n-1}, c')$$

In contrast to binary rules, the head atom $h(A_0, c)$ in unary rules contains a constant c and one variable A_0. Such an expression can also be understood as a complex way to write down a unary predicate, which is the reason for naming these rules as unary rules. Typical examples are head atoms such as $gender(X, female)$ or $citizen(X, spain)$.

We refer to the three rule types introduced above as path rules, because the body atoms $b_i(A_{i-1}, A_i)$ form a *path*. Note that our language bias also includes rule variations with flipped variables in the atoms: given a knowledge graph $\mathbb{G}$, a path of length n is a sequence of n triples $p_i(c_i, c_{i+1})$ with $p_i(c_i, c_{i+1}) \in \mathbb{G}$ or $p_i(c_{i+1}, c_i) \in \mathbb{G}$ for $0 \leq i \leq n$. The (abstract) rules shown above are said to have a length of n as their body can be instantiated to a path of length n. Instead of A_i, we will sometimes use A, B, C, and so on as names for the variables. Moreover, we will usually replace the variables that appear in the head by X for the subject and Y for the object.

Note also that the relations h and b_1 to b_n do not need to be different relations. Our definition includes also recursive rules, i.e., rules where the relation used in the head of the rule appears in one, several, or all body atoms. An example can be found in the transitive rule $contains(X, Y) \leftarrow contains(X, A), contains(A, Y)$.

B and $\mathbf{U_c}$ rules are a special form of closed connected rules. They can be learned by the rule mining system AMIE described in [22, 23]. $\mathbf{U_d}$ rules are not closed because A_n is a variable that appears only once. **B** rules are usually presented as typical examples of closed connected rules; however, the notion of a closed connected rule is wider as it is defined as a rule where each variable appears in at least two atoms. The

[1] In the worst case, we would lose ≈0.0099 MRR if we would assume that the correct candidate would always be on position #101, which is rather unrealistic. More realistic is an estimation of an MRR difference of less than 0.0005.

[2] In [35] binary rules have been called cyclic rules and unary rules have been called acyclic rules. This convention was slightly confusing, because a unary rule can also be sampled from a cyclic path.

rule $h(X, Y) \leftarrow b(X, Y), b'(X, A), b''(A, X)$ is an example of a closed connected rule that is not a **B**-rule.

In the following, we show several rules as examples for some of the rule types. We had to abbreviate some of the relation names and for some of them the meaning might be unclear. Thus, we briefly list and explain the relations.

- $hypernym(X, Y)$ - X is a hypernym of Y
- $hyponym(X, Y)$ - X is a hyponym of Y
- $prod(X, Y)$ - X is a movie produced by Y
- $sequel(X, Y)$ - X is a sequel of Y
- $g(X, Y)$ - X has gender Y
- $profession(X, Y)$ - X has Y as profession
- $actedIn(X, Y)$ - X acted in / starred in movie Y
- $lives(X, Y)$ - person X lives in country Y
- $speaks(X, Y)$ - person X speaks language Y
- $lang(X, Y)$ - X is the official language of country Y

Examples for binary rules, Rules (2) and (3), are shown below. They describe the relation between X and Y via an alternative path between X and Y. This path can contain a single relation or a chain of several relations. As mentioned before, we allow for recursive rules, i.e., the relation in the head can appear one or several times in the body as shown in Rule (3). Rule (4) is a $\mathbf{U_c}$ rule which states that a person is female, if she is married to a person that is male. A typical example for a $\mathbf{U_d}$ rule is Rule (5), which says that an actor is someone who acts (in a film).

$$hypernym(X, Y) \leftarrow hyponym(Y, X) \quad (2)$$

$$prod(X, Y) \leftarrow prod(X, A), sequel(A, Y) \quad (3)$$

$$g(X, female) \leftarrow married(X, A), g(A, male) \quad (4)$$

$$profession(X, actor) \leftarrow actedin(X, A) \quad (5)$$

All considered rules are probabilistic, which means they are annotated with confidence scores that represent the probability of predicting a correct fact with this rule. The *confidence* of a rule is calculated as the fraction of body groundings that result in a correct head grounding based on the training data (see Sect. 3.4 for a definition of confidence and how to compute it). It is important to understand the relation between the three rule types. It is particularly interesting in the context of probabilistic rules. For that purpose, consider the following set of rules (fictitious confidence scores added in square brackets).

$$speaks(X, Y) \leftarrow lives(X, A), lang(Y, A) \ [0.8] \quad (6)$$

$$speaks(X, english) \leftarrow lives(X, A) \ [0.62] \quad (7)$$

$$speaks(X, french) \leftarrow lives(X, france) \ [0.88] \quad (8)$$

$$speaks(X, german) \leftarrow lives(X, germany) \ [0.95] \quad (9)$$

Rule (6) states that X speaks a certain language Y, if X lives in a country A where Y is the official language. Rule (7) says that an entity speaks English if it lives somewhere, which is just an indirect way to describe that entity as a person. The remaining rules are specializations of Rule (7) as they inform about the probability that a person living in a specific country speaks a specific language.

The interesting aspect of this rule set is the fact that Rule (7) can be generated from Rule (6) by removing the second atom in the body and replacing Y in the head with the constant *english*. Likewise, Rules (8) and (9) can be constructed by additionally replacing A by a constant. It seems that we do not need these specialized rule variants, if we already have a more *general* rule. This is wrong for two reasons: (i) it might be the case that the given knowledge graph does not contain information about the official languages of France or Germany; and (ii) the confidences of the specific rule (7)–(9) differ from the confidences of the more general rule. The confidence of a general rule is closely related to the (weighted) average over the specific confidences (e.g., by aggregating over all countries and languages). For that reason, it is necessary to generate both types of rules, even though they might carry partially redundant information. Suppose, for example, that the given graph contains the triples *lives(fritz, germany)* and *lang(germany, german)* and we are interested in the query $speaks(fritz, ?)$. Without Rule (9) we would assign, based on Rule (6), a confidence score of 0.8. Taking all rules into account, we would assign—due to the maximum aggregation strategy mentioned in the previous section—a confidence score of 0.95. This difference might result in a different position in the ranking of all candidates.

3.2 Constructing rules from sampled paths

In the following, we use the notion of a bottom rule to refer to a rule that contains no variables. Such a rule is not useful for making any new predictions; however, it can be used as a basis for deriving more general rules by replacing some of the constants by variables. We adopt a special variant of the basic bottom-up approach for learning rules from *bottom rules* as proposed in [35]. It is divided into the following steps:

1. Sample a path from a given knowledge graph.
2. Construct a bottom rule from the sampled path.
3. Build a generalization lattice rooted in the bottom rule.
4. Store all useful rules that appear in the lattice.

The main focus of this section will be the steps 2–4. Note that we will replace the computation of the lattice mentioned in the third bullet point by directly instantiating certain rule patterns. We will show below that the outcome of our

approach is the same as if we would compute the whole lattice. However, it is simpler and more efficient to directly instantiate these patterns.

The above sketch of our approach is related to the algorithm implemented in Aleph [53]. However, Aleph uses the bottom rule to define the boundaries of a top-down search. It begins with the most general rule and uses the atoms that appear in the bottom rule to create a specialization lattice. A specialization lattice is a directed graph where each node is a rule and an edge from r to r' denotes that r' is more special than r. While in a specialization hierarchy each rule r' has only one more general r rule as parent node, in a lattice r' might have several more general rules that are specialized by r'.

Similarly, AMIE [22, 23] also does a top-down search, which in contrast to Aleph is complete because it does not limit which atoms to use to specialize a rule. Our approach differs fundamentally from both algorithms because we instantiate a set of rule patterns that results exactly in the beneficial subset of those rules that we would collect by creating a generalization lattice beginning from the bottom rule. In a generalization lattice, every child rule is more general than the parent rule. We argue in the following that all relevant rules within the generalization lattice instantiate one of the rule types defined in the previous section. Based on this insight, we can directly instantiate these rule types without the need to create the complete lattice.

To find rules for a fixed relation, AnyBURL samples triples of that relation from the training set and creates rules from them. Even though we do not know all the details of the overall algorithm, which will be available at the end of Sect. 4, we can already conclude that AnyBURL's search is obviously not complete. AMIE, on the other hand, will generate all rules that fulfill the quality criteria defined in the chosen settings. Thus, it can be described as a complete search. The incompleteness of our approach seems to be a significant drawback; however, we will later argue that it helps to detect the most important rules quickly at the beginning of the search. AMIE, on the other hand, might, for very large datasets, not be able finish at all, as it needs to construct all possible rules systematically.

Figure 1 shows a small subset of a knowledge graph $\mathbb{G}$. We use it to demonstrate how rules for the relation *speaks* would be learned from it. We construct bottom rules of length n, beginning from $speaks(ed, d)$, short for (Ed speaks Dutch), which will be the head of the rules. To do this, we randomly walk n steps in the graph, starting either from ed or d. Together with the head triple, the result is a path of length $n + 1$. In Fig. 1, we have marked three paths that could be found for $n = 2$ or $n = 1$, respectively. The green and blue paths are acyclic, while the red path, including $speaks(ed, d)$, is cyclic. We convert these paths into the

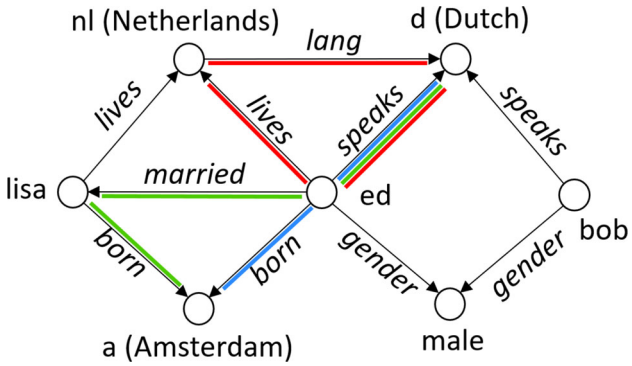

Fig. 1 A knowledge graph $\mathbb{G}$ used for sampling paths. We marked the path that corresponds to Rule (10) blue, Rule (11) green, and Rule (12) red

bottom rules (10), (11), and (12) given below.

$$speaks(ed, d) \leftarrow born(ed, a) \quad (10)$$

$$speaks(ed, d) \leftarrow married(ed, lisa), born(lisa, a) \quad (11)$$

$$speaks(ed, d) \leftarrow lives(ed, nl), lang(nl, d) \quad (12)$$

We argue that any generalization of a path of length $n+1$ will be a $\mathbf{B}$, $\mathbf{U_c}$ or $\mathbf{U_d}$ rule of length n or a shorter rule, which can be constructed from a shorter path, or a rule that is not useful for making a prediction. We elaborate this point by analyzing the generalization lattice rooted in Rule (11), depicted in the lower part of Fig. 2.

Each edge in the lattice transition stems from one of the following two generalization operations: (i) replace all occurrences of a constant by a fresh variable and (ii) drop one of the atoms in the body. Note that we have only depicted those rules in the lattice that have at least one variable in the head. If this would not be the case, the rule would only predict a triple that is already stated in the knowledge graph, which is useless for completion. Not all rules that appear in the lattice are useful in the context of the overall algorithm. We highlighted the beneficial rules in Fig. 2 by printing the border of their enclosing rectangles in bold. The remaining rules, which are not instantiated by our algorithm, belong to one of the categories described in the following paragraph. We have associated the symbols †, ∗, and ⋄ to these categories and used them to mark the nodes in Fig. 2.

Ambiguous prediction† The rule has an unconnected variable in the head, which does not appear in the body of the rule. Such a rule makes a prediction that something exists without exactly specifying what it is. Thus, it cannot be used to create a ranking of candidates. An example would be a rule such as $gender(X, Y) \leftarrow lives(X, usa)$, which states that an entity that is born in USA has a gender. If we have a query as $gender(john, ?)$ the rule is not helpful at all, no matter where john lives.

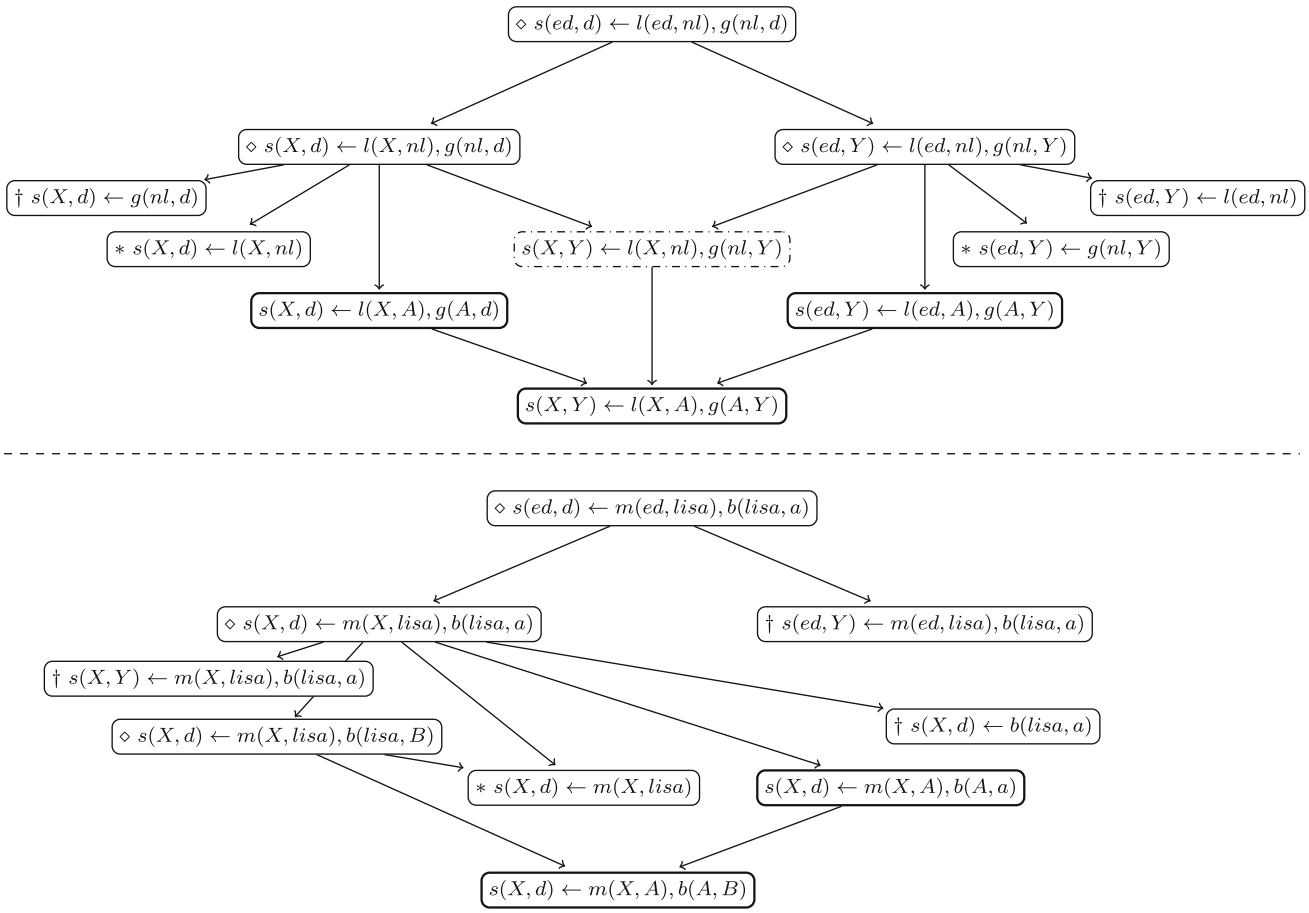

Fig. 2 In the upper half, we depicted the generalization lattice of the cyclic path $(s(ed, d), l(ed, nl), g(nl, d))$, in the lower part the generalization lattice of the acyclic path $(s(ed, d), m(ed, lisa), born(lisa, a))$. For legibility, we use the abbreviations $s = speaks$, $m = married$, $b = born$, $l = lives$ and $g = lang$

Shorter bottom rule* The rule might be useful, but it would also appear in the lattice of a bottom rule which originates from a shorter path. This point is detailed in Sects. 4.3 and 4.4, where we introduce the notion of a path profile which determines the length of the bottom rules. We will see that each path profile has an associated reward in the context of the overall algorithm, which is computed by summing up the reward of each rule that stems from this profile. By suppressing shorter rules, we enforce that each rule can be uniquely assigned to its path profile and will only be created from shorter paths.

Useless atom$^{\diamond}$ The body contains an atom without variables or an atom with a constant and a variable that appears in none of the other body atoms and also not in the head atom. Such atoms will always be true in the knowledge graph from which they were sampled and therefore do not affect the truth value of the body. These rules need to be generalized further. This can be done by dropping the specific atom, which results directly into a shorter rule, or by replacing a constant by a variable.

Note that a rule in the lattice marked with a † or * does not need to be generalized any further, because any resulting rule will be marked again with the same symbol.

When we apply this annotation scheme to the lattice (Fig. 2, lower half) that originates from the green acyclic path (in Fig. 1), only two rules are highlighted with a bold rectangle. These two rules are of type $\mathbf{U_d}$ and $\mathbf{U_c}$. One can easily argue that this will always be the result when we generalize a bottom rule that originates from an acyclic path. Thus, we do not need to search over the generalization lattice but can directly create these two rules from a given acyclic path.

We can observe a similar pattern when we generalize a cyclic path. The corresponding lattice is shown in the upper half of Fig. 2. It results in three rules that we can leverage for a prediction; one $\mathbf{B}$ rule and two $\mathbf{U_c}$ rules, where the head constant (subject/object) appears again in one of the body atoms. It is not so easy to see why the $\mathbf{U_c}$ rules are really required, because every time that one of these rules fire, the more general $\mathbf{B}$ rule will also fire. However, and this is a point that we already mentioned above, the confidences of the general binary rule and the more specific $\mathbf{U_c}$ rules might

be different. For that reason, it makes sense to store these rules to increase the quality of the resulting ranking.

In the upper half of Fig. 2, there is a non-marked inner node remaining that refers to $speaks(X, Y) \leftarrow lives(X, nl), lang(nl, Y)$. It is highlighted by the use of a dashed line. This rule might indeed be beneficial for a prediction. It is a conjunction of two $\mathbf{U_c}$ rule bodies where one is a constraint on the X groundings and the other is a constraint on the Y groundings. This type of rule is the only rule type that appears in the lattice, which is not supported by the language bias of AnyBURL. However, anytime that this rule fires the corresponding $\mathbf{B}$ rule $s(X, Y) \leftarrow l(X, A), g(A, Y)$ will also fire. Moreover, it is possible to create a complete list of $\mathbf{U_c}$ rules which cover the same cases that are covered by that rule. For this reason, it is less important to cover this rule type. Aside from this exception AnyBURL supports all types of rules that would appear in the lattice, i.e., that can be generalized from a simple cyclic or acyclic path.

3.3 Object identity

Object identity (OI) refers to an entailment framework that interprets every rule under the additional assumption that two different terms (variables or constants) that appear in a rule must refer to different entities. This means that each rule is extended by a pairwise complete set of inequality constraints. OI was first introduced in [51] and later it is used to propose refinement operators for the original framework [19]. In this work, we do not focus on its theoretic properties but on its impact on correcting the confidence scores of the learned rules.

In the context of our approach, the most important property of OI is its capability to suppress redundant rules that negatively affect performance. We illustrate the effect with the following two rules (h and b are two arbitrary relations).

$$h(X, Y) \leftarrow h(X, Y) \tag{13}$$

$$h(X, Y) \leftarrow b(X, A), b(B, A), h(B, Y) \tag{14}$$

Interpreting rules under OI can be done by adding additional constraints to the rules. For instance, the body of Rule (14) would need to be extended with the inequality constraints (15).

$$X \neq A, X \neq B, X \neq Y, A \neq B, A \neq Y, B \neq Y \tag{15}$$

We modified AnyBURL to interpret each rule under OI. Note that these inequality constraints are not explicitly shown whenever a rule is displayed or stored in a file generated by the new version of AnyBURL.

Rule (13) is obviously a tautology that will never generate any new facts. This is only partially true for Rule (14). The groundings of its body can be divided into the grounding θ with $B = X$, and the grounding θ' with $B \neq X$. In contrast to a θ' grounding, a θ grounding does not predict new facts and is also more likely to result in a true body because both atoms of relation b can be grounded to the same fact. This means that, without OI, the confidence score of Rule (14) overestimates its quality as it will always be used to predict unknown facts. Adding the inequality constraints will suppress the θ groundings and result in a more realistic confidence score for the task.

It is important to understand that it is not just variations of tautology rules that have this problem. For example, if there are rules with high confidence such as $m(X, Y) \leftarrow spouse(Y, X)$ (m = married), rules like the following are also affected.

$$m(X, Y) \leftarrow son(X, A), son(B, A), spouse(B, Y) \tag{16}$$

The confidence score of such a rule drastically (and rightfully) decreases under OI once we ignore groundings in which X and B are ground to the same son.

While OI helps us to avoid a blow-up of the rule base, a given rule is harder to evaluate under OI (see also §5.1.1 in [14]). This holds both for the confidence computation and for the application of the rule in the context of predicting new knowledge. If we ignore the inequality constraints, all possible (X, Y) groundings for Rule (14) can be computed with two join operations. As a result of the first join, we get the groundings for (X, B) which can be used to compute the (X, Y) groundings via a second join. However, when performing the second join the constraint $A \neq Y$ requires to know the variable bindings of A that we used for the first join to ensure that the constraint is not violated. Keeping track of all variable bindings makes it more complex to compute body groundings under OI.

3.4 Sampling confidences

Confidence and support are standard metrics for measuring the quality of a rule. Confidence can be understood as an estimation of the probability that a rule makes a correct prediction under the closed world assumption, i.e., the assumption that everything not stated in a given knowledge graph is incorrect. However, there are different ways to define confidence (e.g., [23, 58]). All of them are based on the idea of dividing the number of groundings for the conjunction of body and head by the number of body groundings. This informal definition leaves room for different interpretations that are based on different ways of counting. In [22, 23], where the authors introduce AMIE, support and (standard) confidence of a rule are defined as follows. We show the formula for a rule r that uses variables X and Y in the head and X, Y, and Z in the body. If more variables are used in the body, Z has to be replaced in all occurrences by all variables that

appear in the body only.

$$support(r) = |\{\theta_{XY} \mid \exists \theta_Z \; r_b\theta_{XYZ} \wedge r_h\theta_{XY}\}| \tag{17}$$

$$conf(r) = \frac{|\{\theta_{XY} \mid \exists \theta_Z \; r_b\theta_{XYZ} \wedge r_h\theta_{XY}\}|}{|\{\theta_{XY} \mid \exists \theta_Z \; r_b\theta_{XYZ}\}|} \tag{18}$$

In the above formulas, θ_{XY} refers to a grounding for the variables X and Y, which are the variables that appear in the head r_h of r. θ_Z is a grounding for the variables that appear in the body r_b of r that are different from X and Y. θ_{XYZ} refers to the union of θ_{XY} and θ_Z. According to this definition confidence is the fraction of body groundings projected to $\langle X, Y\rangle$ pairs that are correct predictions made by the rule, while support just counts the number of correct predictions.

This definition is an estimation for the probability that a randomly chosen body grounding results into a correct prediction. While we use the closed world assumption to compute the confidence score, we know that this assumption is not valid when we make a prediction. Otherwise we would never predict any new triple. For that reason, the confidence score is a lower bound for the real probability.

It is expensive to precisely compute the confidence of a rule. Starting from the first atom in the body, any further body atom requires to compute a join on the variable that this atom shares with the previous atom. Moreover, since we are using the principle of object identity, we cannot ignore to which entities the join variables were bound, because these entities are not allowed as possible values for the other variables. For that reason, it makes sense to approximate the confidence measure by drawing a sample of all body instantiations.

The most straightforward way to do this is to apply a depth-first search (DFS) over body groundings. This strategy has been applied in the previous version of AnyBURL [35]. Given a **B** rule, it picks a random grounding of a head variable and follows via a DFS the chain of relations in the body collecting all groundings of the other head variable. This is done until a sufficient number of body groundings have been sampled.

An example of applying this approach to a problematic constellation is shown in Fig. 3. Each circle represents a grounding for a variable. The numbers in the circles denote the possible node expansion order. In this example, the entity that is chosen in the second step as grounding for A yields 50 different groundings of the whole body. Entities like this will appear quite often in a knowledge graph that represents aspects of the real world. Typical examples for such entities are the gender female, as in $hasGender(..., female)$ or an important country or city, as in $bornIn(..., paris)$. Such "hub" entities are usually related to many other entities via the same relation. If such an entity appears in one of the sampled groundings at the beginning of the search, it can happen that only those body groundings are constructed that contain

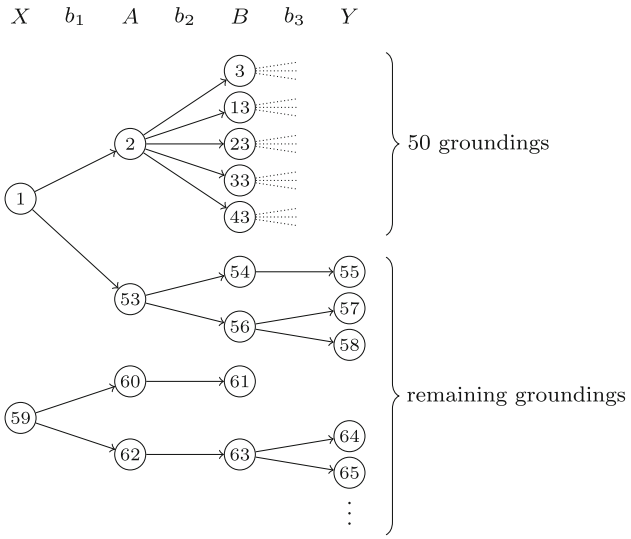

Fig. 3 Problematic example for a DFS-based confidence approximation of $h(X, Y) \leftarrow b_1(X, A), b_2(A, B), b_3(B, Y)$

that specific grounding. This would happen in Fig. 3 if we would draw a sample of 50 groundings. In that case, we are not drawing a sample that is representative for the general rule (19), but instead we are drawing a sample for a more specific rule (20).

$$h(X, Y) \leftarrow b_1(X, A), b_2(A, B), b_3(B, Y) \tag{19}$$

$$h(X, Y) \leftarrow b_1(X, usa), b_2(usa, B), b_3(B, Y) \tag{20}$$

While an increase of the sample size can circumvent the problem in some cases, this will not solve the problem in general. Notice also that large datasets have usually more hub entities that are related to a number of entities that might be higher than the sampling size. This makes the DFS-based sampling prone to distorted confidence scores especially when we apply it to large datasets.

Thus, we implemented a different sampling technique shown in Algorithm 1. This algorithm describes the procedure that we apply to approximate the confidence of **B**-rules. We explain below briefly how to apply the method to $\mathbf{U_c}$ and $\mathbf{U_d}$-rules. We assume that the input to Algorithm 1 is a rule of the form $h(A_0, A_n) \leftarrow b_1(A_0, A_1), ..., b_n(A_{n-1}, A_n)$. The actual algorithm is a bit more complicated as it has to deal with flipped positions of A_{i-1} and A_i, which can be handled with several additional case distinctions. We divided the algorithm into two functions. The first function determines the starting points for the search, while the second function uses these starting points to search for a full grounding of the chain of variables A_0 to A_n. The starting points of the search are groundings of variable A_0. They are collected in the set $\mathbb{X}$. As we use a set, each possible value appears only once. This means that an entity that appears quite often at

Algorithm 1 Sampling groundings

Require: $r = h(A_0, A_n) \leftarrow b_1(A_0, A_1), ..., b_n(A_{n-1}, A_n)$
1: **function** SAMPLEBODYGROUNDINGS(r)
2: $\alpha \leftarrow 0$
3: $\beta \leftarrow 0$
4: $\Phi \leftarrow \{\}$
5: $\mathbb{X} \leftarrow \{\theta_{A_0} \mid \exists\theta_{A_1} b_1(A_0, A_1)\}$
6: **while** $(\alpha < max_\alpha) \wedge (|\Phi| < max_\Phi) \wedge (\beta < max_\beta)$ **do**
7: $\alpha \leftarrow \alpha + 1$
8: $x \leftarrow$ random element from $\mathbb{X}$
9: $y \leftarrow$ BEAM$(1, x, r, \{\})$
10: **if** $y \neq$ **NIL then**
11: **if** $\langle x, y\rangle \in \Phi$ **then**
12: $\beta \leftarrow \beta + 1$
13: **else**
14: add $\langle x, y\rangle$ to Φ
15: $\beta \leftarrow 0$
16: **end if**
17: **end if**
18: **end while**
19: **return** Φ
20: **end function**
21:
22: **function** BEAM$(i, v, r, \mathbb{P})$
23: **if** $v \in \mathbb{P}$ **then** ▷ checks the OI constraint
24: **return NIL**
25: **else**
26: add v to $\mathbb{P}$
27: **end if**
28: **if** $i > n$ **then** ▷ n is the number of body atoms
29: **return** v
30: **else**
31: $\mathbb{V} \leftarrow \{\theta_{A_i} \mid b_i(v, A_i)\}$
32: **if** $\mathbb{V} = \emptyset$ **then**
33: **return NIL**
34: **else**
35: $v \leftarrow$ random element from $\mathbb{V}$
36: **return** BEAM$(i{+}1, v, r, \mathbb{P})$
37: **end if**
38: **end if**
39: **end function**

the subject position of b_1 is not preferred over an entity that appears rarely in that position.

The algorithm uses elements x from $\mathbb{X}$ repeatedly to find pairs $\langle x, y\rangle$ that are the endpoints of a path that correspond to a body grounding of the given rule. There are several conditions and parameters that define when to stop this process. Parameter max_α determines the overall numbers of attempts to find a $\langle x, y\rangle$ pair no matter if this attempt has been successful or not. Another stopping criteria is related to the number of groundings found so far. If this number reaches the boundary max_Φ, the sampling process stops. The third criteria is an additional condition that allows to stop the sampling process early if repeatedly a grounding is sampled that we found already previously. We count how often this happens in β, which is reset to 0 whenever a new grounding has been found. This criterion ensures that we stop early if the number of groundings is rather low instead of drawing many times the same sample. The default values for max_β, max_Φ, and max_α are 5, 1000 and 100000, respectively.

The beam-function (lines 22–39) searches for a path that leads from x, which is a grounding of A_0, over the chain of body relations to a grounding of A_n. It can be understood as an extreme form of a beam search, which selects candidates completely randomly and retains in each step only one candidate. Lines 23 to 27 are responsible for checking the OI constraints. As no variable appears twice in the subject position of the body atoms, we can simply check if the current entity stored in v is amongst one of the previously visited v-values. We use this function repeatedly until we constructed a set of pairs Φ as output of the algorithm. Now we have to check for each of these pairs $\langle x, y\rangle \in \Phi$ if it is also a grounding for the head of r. In particular, we approximate the confidence of the rule as the fraction of pairs for which $h(x, y) \in \mathbb{G}_t$ holds following Definition 18. We replace the set that appears in the denominator by Φ and the set that appears in the numerator by $\{h(x, y) \mid h(x, y) \in \mathbb{G}_t \wedge h(x, y) \in \Phi\}$

The algorithm can be easily modified to be applicable to $\mathbf{U_c}$ and $\mathbf{U_d}$-rules. Each of these rules uses only one variable in the head. Thus, we need to store a set of single values in Φ instead of pairs. The beam-function needs to be modified to return a truth value that is true if a body grounding for a specific x value has been constructed and false otherwise. If it was possible to construct a body grounding, the x value is stored in Φ. Finally, Φ is again used to approximate the confidence according to Definition 18.

4 Search strategy

So far we have explained how to create rules from a given path. We have not yet discussed how to draw such a path and what policy we should apply when drawing a sequence of paths. We have already explained why it makes a difference whether the path is acyclic or cyclic. This difference is taken up again in Sect. 4.1, where we also explain how we sample cyclic and acyclic paths. In Sect. 4.2, we explain that AnyBURL uses a canonical rule representation to detect efficiently whether a new rule has been found previously. In Sect. 4.3, we briefly recall the previously implemented approach, which was based on the concept of saturation. In Sect. 4.4, we present the new policy, that uses reinforced learning to leverage the available computational resources in a better way. Finally, in Sect. 4.5, we explain why and how it is possible to distribute the rule mining process over multiple threads without any loss.

4.1 Path sampling

In the paths that we sample for building bottom rules, each triple on a path is called a step. The steps can be made in the direction of a stated triple or in reverse direction. A step in reversed direction causes flipped terms in the corresponding atom of the resulting rule. Let c_0 to c_n be the entities on a path. We call such a path an *acyclic path* if it does not visit the same entity twice, i.e., $c_i \neq c_j$ for each $i \neq j$. A path is a *closed path* if $c_0 = c_n$ and $c_i \neq c_j$ for each of the remaining pairs of nodes. Such a path forms a cycle, as it ends where it began, and it does not contain any inner cycles.

A closed path results into a binary **B** rule and a special form of a $\mathbf{U_c}$ rule where the constant in the head and body of the rule is the same. An acyclic path results into a $\mathbf{U_c}$ rule (with different constants in head and body) and a $\mathbf{U_d}$ rule. Our method to sample a path is to choose a random entity as a starting point of a random walk. This approach differs from randomly selecting a starting triple $r(c_0, c_1)$, which would favor entities that are used more often. Within the random walk, we randomly select one of the edges that the current node is involved in, i.e., we consider both in- and outgoing edges. Each of these edges is selected with the same probability. Then, we follow the edge to the node that is connected to the current node and continue from that node. If the walk arrives at an entity that has been visited before (prior to the last step), the procedure can be restarted until an acyclic or closed path has been found. It can be expected that the majority of sampled paths will be acyclic. Especially for longer paths it will not often be the case that $c_0 = c_n$. This means that a pure random walk strategy will generate only few binary rules. This can be a problem for the resulting rule sets. According to the results presented in [35, 36], we know that a large fraction of correct predictions can be made with **B** rules.

Thus, it makes sense to design a specific strategy to search for closed paths. We have slightly modified the random walk strategy by explicitly looking for a fact that connects c_{n-1} and $c_0 = c_n$ in the last step. With an appropriate index, it is possible to check the existence of a relation p with $p(c_i, c_j)$ in constant time for any pair of constants. If we find such a triple, we use this as a final step in the constructed path. If we find several such triples, we pick randomly one of them. With this modification, we are able to find more closed paths in the same time span compared to the standard random walk. We are aware that there are more sophisticated methods for finding a closed path of length n, see for example [45].

4.2 Canonical rule representation

Although it is possible that the procedure for path sampling explained in Sect. 4.1 samples a path repeatedly, this rarely occurs in practice, especially for large graphs. It may happen more frequently, however, that AnyBURL samples a previously unseen path from which a rule is constructed that has been already constructed before. Such a behavior is typical, and we will in Sect. 4.4 introduce an approach that uses reinforcement learning which takes into account repeated rule construction by the reward and policy design. However, first we need a method that allows to check efficiently if a rule has already been seen before or if a new rule has been constructed. Such a check is in general non-trivial as the same rule can be represented in two completely different ways.

In fact, the problem of checking rule equivalence can be considered as the problem of testing graph isomorphism. Isomorphism of graphs can be checked in quasipolynomial time in the worst case. However, this problem can be solved in polynomial time for graphs of bounded degree [2]. Because of our language bias the graphs constructed from the learned rules are bounded (in such graphs, the maximum degree of a node is 2). Moreover, since we have a procedure for canonical rule representation, we can formalize the problem as: given two rules r_1 and r_2, they are isomorphic iff $C(r_1) = C(r_2)$ where $C(r)$ is the canonical representation of a rule r [2, 34]. Here is an example of two equivalent **B**-rules:

$$h(X, Y) \leftarrow b_1(X, A), b_2(A, B), b_3(B, Y) \quad (21)$$

$$h(A_0, A_3) \leftarrow b_3(A_2, A_3), b_1(A_0, A_1), b_2(A_1, A_2) \quad (22)$$

Fortunately, our language bias is so restrictive that it is easy to define a canonical representation for each rule, which enforces that there is a bijective mapping between each set of equivalent rules and their canonical representation.

Rule 21 is a rule in its canonical representation. For a **B**-rule, AnyBURL uses always X and Y as the variables in subject and object position in the head. Then, the body atom using the X variable is used as first atom and the body atom using the Y variable is used as last atom in the rule body. The atoms in between are listed in the order that is implied by the path that leads from X to Y. AnyBURL uses the variable names A, B, C, and so on, in the atoms that occur between the first and the last atoms. A is used as variable that appears in the same atom as X, B is the variable that appears in the atom that uses A, and so on.

A similar canonical representation is used for $\mathbf{U_c}$- and $\mathbf{U_d}$-rules. These rules contain at some positions constants instead of variables. As there is no specific identity relation, we can assume that different constants denote different entities. Moreover, our formalism does not support functions nor do functions belong to the vocabulary used in standard knowledge graphs. Thus, each term in an atom is a constant or variable. This means that we can define a canonical representation without special treatment for terms that are not variables.

AnyBURL hashes the rules based on their canonical representation. Moreover, each rule is directly constructed and

represented in its canonical representation. This allows to check the existence of a rule that has been found previously with a constant-time performance. In the following section, we measure, for example, the overlap between two rule sets. As explained, each atomic check in such an operation requires constant time.

4.3 Saturation-based search

A detailed description of the search policy that was implemented in a previous version of AnyBURL can be found in [35]. According to that policy, termed saturation-based search, the learning process is conducted in a sequence of time spans of fixed length (e.g., one second). Within a time span, the algorithm learns as many rules as possible using paths sampled from a specific path profile. A path profile describes path length and whether the path is cyclic or acylic. When a time span is over, the rules found within this span are evaluated. Let $\mathbb{S}$ refer to the rules that have been learned in the previous time spans, let $\mathbb{S}_t$ refer to the rules found in the current time span, and let $\mathbb{S}'_t = \mathbb{S}_t \cap \mathbb{S}$ refer to the rules found in the current time span that have also been found in one of the previous iterations. If $|\mathbb{S}'_t|/|\mathbb{S}_t|$ is above a saturation boundary, which needs to be defined as a parameter, the path length of the profile is increased by one. Initially, the algorithm starts with paths of length 2 resulting in rules of length 1. The higher the path length, the more time spans are usually required to reach the saturation boundary. The difference between cyclic and acyclic paths is taken into account by flipping every time span between the cyclic and acyclic profiles. The rule counts generated by cyclic and acyclic rules are independent.

It is an advantage of the saturation-based approach that it does not require to mine all cyclic (or acyclic) rules of length n before the algorithm looks at cyclic (or acyclic) rules of length $n+1$. Instead of that, the rule length is increased if a sufficient saturation has been reached. However, it is unclear how to set the required saturation boundary. The default is 0.99 which is motivated by the idea that such a boundary can be passed, if and only if less than one percent of the rules found in the current time span have not been seen before. However, it is hard to estimate whether this is a good choice. If this boundary is set too high, the algorithm spends a lot of time sampling paths resulting into rules that have already been generated previously. Moreover, a threshold that might work well for a small dataset might be too high for a large dataset that has a high number of relations. In such a situation, there is usually a high number of less important rules which hinder the algorithm to exceed the threshold. This means that for a (very) long time the algorithm focuses on short rules even though there might by some more important rules that have an additional body atom. If the value is set too low, important rules might be missed and cannot be found any more. Another disadvantage of the algorithm is that the ad hoc setting to spend exactly half of the time to search for cyclic paths and half for acyclic paths. To overcome these shortcomings, we propose a reinforced approach presented in the following section.

4.4 Reinforced search

In the following, we consider the path sampling problem as a special kind of multiarmed bandit problem [26]. This means that we have to decide about the policy that guides the overall process and how to quantify the reward associated with a learned rule.

4.4.1 Reward

In each time span, we have to decide how much effort to spend on which path profile. A path profile in our scenario corresponds to an arm of a bandit in the classical reinforcement learning setting. Each arm (or slot machine) in the bandit problem gives a reward when pulling that arm. The reward of pulling an arm corresponds in our scenario to the reward of creating rules from the paths that belong to a certain profile.

In the following, we develop three different reward strategies. They are based on the notion of measuring the reward paid out by a profile in terms of the explanatory quality of the rules that were created by that profile. The explanatory quality of a rule set can be measured in terms of the number of triples of a given knowledge graph that can be reconstructed with the help of the rules from the set. Thus, summing up the support of the rules seems to be a well-suited metric. We refer to this as reward strategy R_s.

$$R_s(\mathbb{S}) = \sum_{r \in \mathbb{S}} support(r) \tag{23}$$

where $\mathbb{S}$ is a set of rules and $support(r)$ is the support of a rule r as defined above.

As we are especially interested in rules that make many correct predictions with high confidence, which might result into top ranked candidates, we propose a second reward strategy $R_{s\times c}$ that multiplies the number of correct predictions by their confidence:

$$R_{s\times c}(\mathbb{S}) = \sum_{r \in \mathbb{S}} support(r) \times conf(r) \tag{24}$$

where $\mathbb{S}$ is a set of rules, $support(r)$ is the support and $conf(r)$ is the confidence of a rule r. A third reward strategy takes rule length $l(r)$ of a rule r into account. Remember that we defined the length of a rule as the number of its body

atoms.

$$R_{s\times c/2^l}(\mathbb{S}) = \sum_{r\in\mathbb{S}} support(r) \times conf(r)/2^{l(r)} \tag{25}$$

This reward strategy is a variant of $R_{s\times c}$ that favors shorter over longer rules. It enforces a constraint that assigns at the beginning of the search more computational effort to short rules. Thus, the search constraint has some similarities with a softened saturation-based search as long as we are only concerned with rule length.

All metrics are based on the capability of a rule set to reconstruct parts of a given knowledge graph in terms of the training set. An alternative approach would have been to compute the same or similar scores with respect to the prediction of the validation set. As we focus on the training set, we can directly reuse the scores that we already computed. Additional computational effort is not required.

4.4.2 Policy

All three reward strategies can be combined with each of the following two policies. The first policy is a well-known policy referred to as ϵ-greedy policy [57]. The parameter ϵ is usually set to a relatively small positive value, for example $\epsilon = 0.1$. Every time a decision needs to be made, that decision is a random decision with a probability ϵ and a greedy decision with a probability $1-\epsilon$. When we talk about decisions, we mean the allocation of CPU cores to path profiles. In the ϵ-greedy policy, a small number of decisions is randomized to reserve a small fraction of the available resources for exploration compared to an approach that would focus completely on exploitation.

In our context, a greedy decision assigns all cores, which have not been assigned randomly, to a path profile that generated the rule set that yielded the highest reward the last time it has been selected. Formally, for ϵ-greedy policy, a path profile pf^*, with $1-\epsilon$ probability, is chosen for time span t_k according to the following equations

$$pf^* = \underset{pf\in\mathbb{F}}{\text{argmax}}\, Q(pf, last(pf, t_k)), \tag{26}$$

$$Q(pf, t_i) = \frac{1}{N_{t_i}(pf)}\mathbf{R}(S(pf, t_i) \setminus \bigcup_{j=1}^{i-1} S(pf, t_j)) \tag{27}$$

where $last(pf, t_k)$ refers to the last time span t_i prior to t_k (i.e., with $i < k$) where path profile pf has been used, $Q(pf, t_i)$ is the value of the path profile pf for time span t_i, $\mathbb{F}$ is the set of path profiles, $\mathbf{R}$ denotes a reward strategy R_s, $R_{s\times c}$ or $R_{s\times c/2^l}$, and $S(pf, t_i)$ refers to the set of rules that have been mined by the use of path profile pf during t_i. The expression $S(pf, t_i)\setminus\bigcup_{j=1}^{i-1} S(pf, t_j)$ refers to the set of new rules that have been mined in t_i but not in one of the previous time spans. We quantify $N_{t_i}(pf)$ in terms of the number of cores that are assigned to pf during t_i. This means that the reward is normalized with the number of allocated cores.

The above definitions can only be applied if there is a previous time span for each profile pf where that profile has achieved a reward, i.e., if $last(pf, t_k)$ is defined for each profile pf and for each time span t_i with $i \geq 1$. This is obviously not the case in the first time span t_1. For that reason we set $last(pf, t_0) = \infty$ for each $pf \in \mathbb{F}$. This results into a random selection in t_1 and ensures that each profile has been chosen once after the first $|\mathbb{F}|$ time spans have passed.

Note that our scenario differs from the classical multi-armed bandit setting in the sense that the expected reward of a certain profile will decrease any time we use this profile for generating rules. The more often we use that profile, the more probably it is to draw a path that results into a previously learned rule, which was created from the same or from a different path. For that reason, we do not base our decision on the average of all previous time spans, but look at the last time span that this profile has been used. The reward of a profile is shrinking continuously, with random ups and downs that are caused by drawing only a limited number of path samples. This results into flips between different profiles which are not caused by knowing more (exploration) but by the impact of exhausting profiles over time.

The ϵ-greedy policy might not be a good choice if some profile pf creates higher rewards than another profile pf'; however, pf' would also generate relatively good rules that could make correct predictions. Suppose further that both profiles are relatively stable, i.e., their reward decreases only slightly when they are used for generating rules. In such a setting, we might prefer to draw rules not only from pf but also from pf'. For that reason, we propose a second policy where we distribute the available computational resources to all profiles proportional to the reward that has been observed the last time they have been used. We refer to this policy as weighted policy. For each CPU core with a probability ϵ, we take a random decision and with a probability $1-\epsilon$ we proceed as follows. For each profile $pf \in \mathbb{F}$, we compute the probability of resource allocation $P(pf, t_k)$ at time span t_k, given by the following formula:

$$P(pf, t_k) = \frac{Q(pf, last(pf, t_k))}{\sum_{pf'\in\mathbb{F}} Q(pf, last(pf', t_k))} \tag{28}$$

where Q and $last$ are introduced above. For each core that is not yet assigned to a profile due to the random assignment in the $1-\epsilon$ case, we assign one of the path profiles $pf \in \mathbb{F}$ with probability $P(pf, t_k)$.

To better understand the impact of combining different reward strategies and policies, AnyBURL can be run with a completely random policy where each profile has always the same probability. This can be achieved by setting $\epsilon = 1$.

This setting is not necessarily bad. If there are K different profiles, in the worst case one of these profiles would generate many useful rules and none of the other profiles would generate such rules. An algorithm that makes perfect decisions would arrive K times faster at the same result than the random policy. However, this is an extreme setting and useful rules are usually scattered over different path profiles. This means that an approach that draws paths from each profile with the same probability is the simplest approach to collect these scattered rules. At the same time, we can assume—and the results published in [35] support this assumption—that the most beneficial rules are often mined first. This means that running the random policy and the weighted policy for the same time span will not yield results that are K times worse. The random policy might even outperform the previous, saturation-based implementation of AnyBURL. This will be the case if the saturation threshold is chosen too low or too high.

4.5 Multithreading

In this section, we explain roughly how and why it is possible to distribute the rule mining process to different threads. Contrary to this, approaches that rely on knowledge graph embeddings cannot be parallelized that easily (see [27] for an overview on different parallelization techniques and their challenges). They require to modify the shared representation in each basic operation, whereas the rule mining process is additive. Its basic operation is to add rules to a set of rules.

We call threads responsible for mining rules *worker threads* and the thread that assigns different path profiles to different workers *controller thread*. As explained above, each rule is generated from a sampled path. This means that each worker needs to have read access to the knowledge graph which is stored in RAM together with index structures that allow to answer basic queries in constant time (e.g., retrieve all groundings for X with $r(c, X)$). For each time step, the worker is informed by the controller about the path profile that the worker has to use within this time step. Each worker samples paths from this profile, converts them to rules, and computes the confidence scores of these rules by drawing samples as described above. This is done until the time span is over. We set this time span to two seconds in our experiments.

Once a worker realizes that the end of a time span is reached, the thread computes the reward and sends this information to the controller. As soon as this information is available for all worker threads, the controller aggregates this information to decide for the next time step which path profiles should be used to which degree. The controller assigns again a path profile to each worker and the next time step is started. This procedure continues until the overall user-defined learning time runs out and finally all rules that have been learned are stored in a file.

Each rule that fulfills a set of predefined quality criteria, e.g., making a certain number of correct predictions against the training set, has a certain score that is added to the reward that the path profile achieved in the time span. However, the rule needs to be new in the sense that it has not been found by this worker thread or any other worker thread previously. All threads use a shared hash table to store new rules and to check whether the rule has already been found. Storing a rule in this hash table is the only critical operation because it requires to modify a synchronized data structure. Nevertheless, such a basic update operation happens significantly less often than the basic operations that need to be executed when sampling and scoring a rule. This means that the overall process will not slow down as long as the number of threads used will not be close to the number of basic operations required for computing the score of a rule. In our experiments, we made tests with up to 50 threads and we observed only a minor slowdown at the beginning of the mining process where nearly each rule is a new rule that needs to be stored. After a few seconds, the number of additions to the shared set of learned rules decreases significantly as a large fraction of sampled rules is already known. Thus, the synchronized access is no longer a bottleneck for the overall performance.

5 Comparative analysis

In this section, we present experiments that compare the improved version of AnyBURL against other approaches on four large datasets with respect to predictive quality, total runtime, and energy consumption. We introduce the datasets in Sect. 5.1 and provide a detailed discussion about the used settings, the compared work, and runtime calculations in Sect. 5.2. We compare the results of our approach against other techniques that have so far been applied to these datasets in terms of MRR and hits@k in Sect. 5.3. Moreover, we discuss runtimes and electricity consumption in Sect. 5.4. Both aspects are neglected in most of the research papers. Finally, we provide a brief discussion about memory consumption in Sect. 5.5. In the comparisons, we focus mainly on the current state of the art, which is given by knowledge graph embedding models. However, we also compare our results against the results of the previous version of AnyBURL [35] and include the latest version of AMIE, AMIE 3 [30], as another representative of a rule-based approach. Note that we discuss other rule-based approaches including some of their experimental results in Sect. 7.

Table 1 Dataset characteristics. The first three lines refer to number of entities, relations and triples in the training set. The last line refers to the number of triples in the test set

	FB15k	Yago03-10	Wikidata5M	Freebase
Entities	15k	123k	4,594k	86,054k
Relations	1,345	37	822	15,000
Training set	483k	1,079k	20,625k	338,586k
Test set	59k	5k	5.3k	10k

5.1 Datasets

In the previous years, a large fraction of research on knowledge graph completion techniques was focused on two evaluation datasets known as FB15k-237 and WN18RR. They are subsets of FB15k (a subset of Freebase) and WN18 (a subset of WordNet). Both datasets have been proposed in [9], and it has been shown that AnyBURL is competitive.[3] However, they are rather small-sized and the goal of this work is to extend symbolic rule learning to larger knowledge graphs, i.e., to make AnyBURL applicable to more realistic scenarios.

Therefore, we focus on the datasets used in [27], an experimental study about different methods for parallel training of knowledge graph embedding models. In particular, the datasets are FB15k, Wikidata5M (WD5M), Yago03-10, and Freebase (FB). The characteristics of these datasets are described in Table 1. Especially for the two largest datasets, WD5M and Freebase, it is hard or at least extremely costly to learn embeddings in non-parallelized settings, for instance, as model parameters might not fit on a single GPU [27]. They are thus well-suited for assessing if our improved rule-based approach can deal with challenging large-scale settings.

In regard to the FB15k dataset, the best performing approaches achieve a hits@1 score that is lower than 85%, suggesting that there is still room for improvement. The dataset Yago03-10 is described in [33] and has first been used in the context of knowledge graph completion in [16]. It is the subset of YAGO3 that consists of entities which are described by at least 10 triples. It is two times larger in number of triples than FB15k, even though it uses only 37 different relations. WD5M [61] is based on the July 2019 dump of Wikidata. Freebase is the largest dataset that we use and it is simply the full version of Freebase. We use the version that has been used in [27, 31]. To make our results comparable to the results presented in [27], we use the same subset of the test set for evaluation and we use the full test set, validation set and training set for filtering when we compute the filtered MRR and hits@k scores.

[3] For a detailed treatment, we refer the reader to [47], where an independent evaluation study comparing different families of knowledge graph completion approaches is presented. One of these approaches is the version of AnyBURL that is close to the version that we describe in this paper. The results shown in [47] indicate that the approach is among the best methods with respect to both predictive quality and runtime efficiency.

5.2 Experimental settings

The main results of Sect. 5 are presented in Table 2. In the following, we describe the experimental details in regard to AnyBURL, introduce prior work against which we compare, and explain settings and some dimensions of Table 2 with a special focus on runtime.

5.2.1 AnyBURL settings

Throughout all experiments we use, with only one exception, the default AnyBURL setting. As described in [35], we use max aggregation to generate predictions from a rule set. We learn rules up to length three from cyclic paths and restrict the length of rules learned from acyclic paths to one. Previous experiments have shown that **B** rules of length ≥ 4 can improve the results only to a rather limited degree, while using them for prediction is rather costly in terms of runtime. It is similar to $\mathbf{U_d}$ and $\mathbf{U_c}$ rules of length ≥ 2. Moreover, the search space for $\mathbf{U_c}$ rules with one body atom is already quite large, as they use one constant in the head and one constant in the body. If we would allow a second body atom, this would increase the number of mined rules by several orders of magnitude. This rule set would not fit into main memory given a high number of entities and relations.

Confidences of rules are approximated by sampling and evaluating groundings on the training set followed by a Laplace smoothing with parameter $p_c = 5$ (see Section 4 in [35] for details). For all datasets, with the exception of Freebase, we keep all rules with a support of at least two. For Freebase, we increase this parameter from two to five to avoid rule sets which might become too large. If not stated otherwise, we use within this section the weighted reinforced policy together with the reward strategy $R_{s \times c/2^l}$. We compare different policies and reward strategies later in Sect. 6.

We are running the AnyBURL experiments with the datasets FB15k, Yago03-10, and WD5M on a CPU sever with 786 GB RAM and two Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz cores which are virtualized as 40 cores. For Freebase, we used a CPU sever with 1024 GB RAM and an AMD EPYC 7413 24-Core Processor with 96 virtualized cores. We set the number of worker threads in all experiments to 32.

Note that we do not use the validation set as the same AnyBURL setting is used for all datasets. The validation set is only required for computing the filtered scores which allows to make our experiments comparable to the results of other models. For the two largest datasets, we present the

results based on learning rules for 100, 1000 and 10000 s. For FB15k and Yago03-10, we present only the results for 100 s and for 100 and 1000 s, respectively, as longer learning times do not provide additional benefits.[4]

While a knowledge graph embedding model requires to calculate a complete ranking as triple scores are not known beforehand, when sorting the rules of AnyBURL before the prediction stage according to their confidences, it is possible to efficiently create top-k rankings. For all the experiments, we calculate the top-100 rankings with AnyBURL and every candidate not within these top-100 is assigned a score of 0. Tie handling is performed with the random strategy as proposed in [56].

5.2.2 Current state of the art

In [27], the authors used two standard knowledge graph embedding models (ComplEx [60] and RotateE [55]) via different parallelization techniques across multiple GPUs or machines to perform knowledge graph completion on large datasets. We compare against the best results of this work. After an initial small non-parallelized hyperparameter search with 30 trials and 20 epochs for every dataset, the best configurations (measured with the help of the validation set) have been used in various parallelized settings. From these settings (compare Table 5 in [27]), we have chosen the approaches that resulted in the best MRR and the approach that turned out to be the fastest. We marked these approaches with b (best) and f (fastest), and added them to our results Table.

Note that parallelization techniques and approaches for large-scale knowledge graph completion have also been proposed in [31, 66]. However, these papers use a sample-based evaluation. In [27], the authors have argued that this evaluation technique generates misleading results. Moreover, they implemented most of the methods proposed in [31, 66] under a common framework and included them in their experiments.

While [27] has a focus on runtimes and scalability, we also discuss the most recent state-of-the-art results in regard to predictive quality on the respective datasets. For WD5M, to our knowledge, the best results in terms of MRR have been achieved by a model called KGT5 [50], an ensemble combining an encoder–decoder transformer model and ComplEx. For the two smallest datasets, we pick additionally the best and the second best results from the meta study [47], where sixteen different methods have been evaluated. Finally, for Freebase we include the results of *GRASH* [28] which is the first algorithm that successfully performed a hyperparameter search on this dataset and achieved state of the art with respect to predictive quality.

In addition to these state-of-the-art embedding-based approaches, we also included AMIE 3 [30] in our experiments. AMIE 3 is the latest version of the rule learner AMIE [23]. It is specifically designed to be applicable to large datasets. Results of AMIE are not available for the prediction tasks and datasets used in our experiments. Thus, we had to run AMIE on our own. We report about results for three settings. We used the default setting, which is rather restrictive. It does not allow any constants; moreover, only rules with one or two body atoms are constructed and only those rules are generated that have a support of more than 100. In addition to the default setting, we report about the results for two relaxed settings where we decrease the support threshold to two (referred to as $s \geq 2$ in Table 2) and increase the rule length from two to three (referred to as $l \leq 3$ in Table 2). We do not report about some initial experiments where we also activated rules with constants, as AMIE did not terminate within 24 h for three out of four datasets. Note that we had to use AnyBURLs rule application module to apply the learned rules as AMIE does not support the functionality to solve link predictions tasks based on the learned rules.

We report also about the results of the old version of AnyBURL [35], which we refer to as AnyBURL-2019 in Table 2. We do not expect that this version yields results that are on par with currents state of the art. We included it to illustrate the benefits of the improvements and extension that we described in the paper as a whole.

5.2.3 Runtime calculation

The runtimes of AnyBURL and AMIE can be divided into three phases. First the dataset is loaded and indexed (*Index* column of Table 2). Then rules are learned for one of the above-defined time spans (*Learn* column). These rules are used to make predictions in the final phase (*Predict* column). For knowledge graph embedding models, the final runtimes are composed differently, which will be explained in the following paragraphs.

The runtimes of latent models based on continuous learning can be divided into a phase called hyperparameter search (*HS* column of Table 2) and into the learning/training phase (*Learn* column). It is obvious that these models also need to load the data. We could not find any numbers related to this point; therefore, we assumed in our calculations that no time is required.

Loading the data will likely take the same amount of time for both approaches. However, in regard to prediction times KGE models are usually faster than AnyBURL when they are operating on GPUs. Due to vectorized operations, many candidates can be predicted simultaneously for obtaining rankings. We were not able to find precise prediction

[4] The current version of AnyBURL, together with all configurations files and datasets required to rerun our experiments, is available at https://web.informatik.uni-mannheim.de/AnyBURL/

Table 2 Results of AnyBURL compared to the models and parallelization techniques analyzed in [27]. Runtimes of knowledge graph embedding approaches have been computed based on the numbers available in the respective publications. Italic type indicates that exact runtimes are not available in a paper but have been estimated from a diagram or have been retrieved by contacting the authors of the publication

Dataset / Model		Runtimes					Rules	Predictive Quality		
		Index	Learn	Predict	HS	Σ		hits@1	hits@10	MRR
FB15k	AnyBURL	9 s	100 s	512 s	–	10.3 m	2243k	**0.820**	0.897	0.847
	AnyBURL-2019 [35]	2 s	10000 s	5260 s	–	254 m	10240k	0.797	0.888	0.825
	AMIE, default	2 s	58 s	11 s	–	1.2 m	37k	0.773	0.824	0.791
	AMIE, $s \geq 2$	2 s	79 s	16 s	–	1.6 m	92k	0.803	0.855	0.821
	AMIE, $l \leq 3$	2 s	34740 s	782 s	–	592 m	1872k	0.775	0.833	0.795
	ComplEx (f)		104 s		26 s – 780 s	2.2 m – 14.7 m			0.858	0.766
	ComplEx (b)		2360 s		118 s – 3540 s	41.3 m – 98.3 m			0.862	0.779
	RotatE (b&f)		1104 s		92 s – 2760 s	19.9 m – 64.4 m			0.835	0.725
	1st in [47], ComplEx					*>10 h*		0.816	**0.905**	**0.848**
	2nd in [47], HolE					*>2 h*		0.759	0.869	0.800
Yago03-10	AnyBURL	8 s	100 s	186 s	–	4.9 m	738k	0.495	0.683	0.561
		8 s	1000 s	221 s	–	20.5 m	4079k	0.497	0.689	0.565
	AnyBURL-2019 [35]	5 s	10000 s	5133 s	–	252 m	3903k	0.428	0.640	0.492
	AMIE, default	7 s	23 s	1 s	–	0.5 m	224	0.303	0.517	0.375
	AMIE, $s \geq 2$	7 s	38 s	1 s	–	0.8 m	519	0.305	0.519	0.377
	AMIE, $l \leq 3$	7 s	4440 s	2 s	–	74 m	2561	0.327	0.557	0.405
	ComplEx (b)		8627 s		486 s – 14580 s	151.9 m – 386.8 m			0.675	0.542
	ComplEx (f)		7600 s		380 s – 11400 s	133 m – 316.7 m			0.669	0.538
	RotatE (b)		29640 s		1482 s – 44460 s	518.7 m – 1235 m			0.637	0.451
	RotatE (f)		13260 s		816 s – 24480 s	234.6 m – 629 m			0.607	0.438
	1st in [47], ComplEx					*>20 h*		**0.505**	**0.704**	**0.576**
	2nd in [47], TuckER					*>10 h*		0.466	0.681	0.544
Wikidata5M		122 s	100 s	1285 s	–	25.1 m	85k	0.275	0.378	0.310
	AnyBURL	122 s	1000 s	2591 s	–	61.8 m	642k	0.300	0.413	0.338
		122 s	10000 s	7009 s	–	285.5 m	4294k	**0.312**	**0.433**	**0.353**
	AnyBURL-2019 [35]	82 s	10000 s	1964 s	–	207.7 m	459k	0.261	0.356	0.293
	AMIE, default	104 s	1443 s	4 s	–	25.8 m	3.7k	0.178	0.204	0.188
	AMIE, $s \geq 2$	104 s	1942 s	6 s	–	34.2 m	27.5k	0.190	0.221	0.202
	AMIE, $l \leq 3$	104 s	>24 h	–	–	>24 h	–	–	–	–
	ComplEx (b&f)		63840 s		4560 s – 136800 s	0.8 days – 2.3 days			0.398	0.308
	RotatE (b&f)		35003 s		9334 s – 280020 s	0.50 days – 3.6 days			0.344	0.264
	KGT5					*≈7 days*		0.267	0.365	0.300
	KGT5+ComplEx					*≈10 days*		0.286	0.426	0.336
Freebase		1150 s	100 s	264 s	–	25.2 m	113k	0.690	0.714	0.699
	AnyBURL	1150 s	1000 s	412 s	–	42.7 m	505k	0.698	0.725	0.707
		1150 s	10000 s	971 s	–	202 m	3245k	**0.702**	**0.728**	**0.711**
	AnyBURL-2019 [35]	539 s	10000 s	414 s*	–	182.5 m	254k	0.648	0.665	0.655
	AMIE, default	886 s	>24 h	–	–	>24 h	–	–	–	–
	ComplEx (b)		7046 s		118 s – 3540 s	119.4 m – 176.4 m			0.529	0.426
	ComplEx (f)		5916 s		26 s – 780 s	99.0 m – 111.6 m			0.523	0.421
	RotatE (b)		64957 s		92 s – 2760 s	1084.15 m – 1128.6 m			0.627	0.566
	RotatE (f)		9383 s		92 s – 2760 s	157.9 m – 202.4 m			0.621	0.562
	ComplEx [28]				32274 s	>537.9 m				0.678

times regarding the full datasets and the compared models. Nevertheless, in [47] average prediction times for individual queries in milliseconds on Yago03-10 and FB15k are given for, e.g., ComplEx, RotatE and AnyBURL. AnyBURL is roughly on average slower by a factor of 10-1000. As we do not have better estimates, we will suppress the prediction times of the KGE models and set them to zero.

We will now highlight the runtime calculations that we performed for the comparison against [27]. As explained above, the optimal hyperparameter setting has been identified via a straightforward sequential search using 30 search configurations (trials) trained for 20 epochs. As the work investigates the parallelization of the training process instead of the hyperparameter search, we have to consider that the search conceptually can easily be parallelized as search trials can be run independently. For a lower bound, we assume that all trials were run in parallel. Note that this requires to use 30 GPUs in parallel. As an upper bound we simply assume the search was run sequentially. For the upper bound, this results in $20 \times 30 = 600$ epochs used for the search and for the lower bound it is $20 \times 1 = 20$ epochs. The epoch numbers are multiplied with the epoch runtimes which are provided in the work. These calculations lead to the *HS* column of Table 2. Please note that the energy consumption with respect to GPU usage is identical in both cases. For the *Learn* column we multiply the number of epochs used for training the final model using the best hyperparameter setting, which were provided by the authors, with the epoch runtimes. For the Freebase dataset, the authors trained constantly for 10 epochs and did not perform a hyperparameter search instead the respective configurations of FB15K were used.

In regard to the runtimes of *GRASH* [28], which is designed to perform an efficient hyperparameter search on large datasets, we obtain the overall time for conducting the search from the paper. We could not find the training time for the final model after the search. Hence, we exclude it to not make an unfair assumption.

Total runtimes for ComplEx, HolE, and TuckER as used in [47] are directly obtained from Figures 5 and 6 in this paper. As these runtimes are only shown in a diagram with a logarithmic scale, we had to estimate a lower bound. Note that these numbers do not include the hyperparameter search. As no runtimes were specified for KGT5 in [50], we contacted the authors and they provided a rough estimation, which we added to our results table (Table 2).

5.3 Results

In this section, we discuss the results in Table 2 with respect to prediction quality and briefly mention runtime considerations. As explained above, the entries for ComplEx and RotatE marked with (b) and (f) correspond to the best and fastest configuration in [27], the first and second best models from [47] are added as well as the current state of the art on WD5M [50] and Freebase [27]. Further details can be found in the experimental settings section (§5.2). For AnyBURL and AMIE we calculated filtered hits@1, hits@10 and MRR. For the remaining models, we reused scores available in the literature. In the Σ column, we summed up the runtimes of all operations required to learn a model and to use that model to solve the prediction tasks of the test sets.

AnyBURL outperforms each combination of knowledge graph embedding model and parallelized setting described in [27] for each of the four datasets when we look at the results based on the rule sets that have been learned after 100 s. If we increase the learning time to 10000 s, this holds also for the ensemble of KGT5+ComplEx on the WD5M dataset, which has been described as the current state of the art in [50].[5] Moreover, AnyBURL outperforms the previous best results [28] on Freebase with respect to MRR and hits@10.

Our results for FB15k and Yago03-10 outperform ComplEx and RotatE results when using different parallelization techniques [27], however, several publications reported slightly better results on these datasets. The best model from around sixteen models that have been analyzed in [47], achieved 0.848 on FB15k. In [29] the authors reported an MRR of 0.86. With an MRR of 0.847 we are close to these numbers. With respect to Yago03-10 its similar. While we achieve a score of 0.565, in [29], the authors report an MRR of 0.58 and in [47] an MRR of 0.576. Runtimes have not been stated in [29], even though it's known that the authors use a very large embedding dimension which may lead to high runtimes. The results reported in [47] require a training time (not including the hyperparameter search) of more than 20 h for Yago03-10 while AnyBURL require only slightly more than 20 min. These runtime comparisons might indicate what effort is required to achieve these MRR scores with an embedding approach; more detailed discussions of runtimes will be provided in the next section. We conclude that for datasets limited in size it is possible to find models that slightly outperform AnyBURL in terms of predictive quality. However, the results in [47] indicate that in a realistic scenario we might not know which model performs well beforehand, e.g., the choice for the best model and hyperparameters depends on the dataset, and AnyBURL achieves good results out of the box on each of the datasets with a fixed universal configuration.

The datasets WD5M and Freebase are significantly larger than the two other datasets. This means that an exten-

[5] There are some works that use instead a sampled MRR [31, 66], where the correct entity is not ranked against all other entities but against a relatively small sampled subset. We refer to the considerations in [27], that clarify why this variant of the MRR should not be used, yields distorted results, and the respective models perform worse than the models/techniques used in [27], which are again clearly outperformed by our version of AnyBURL.

sive hyperparameter search is very costly. Moreover, these datasets have not been used for several years in an evaluation context. This means that experience about hyperparameter settings (or hyperparameter search spaces) for knowledge graph embedding models are not yet available. This is a situation that resembles a realistic evaluation scenario. Indeed, our symbolic method clearly outperforms the previous state of the art results in regard to prediction quality while being faster with respect to total runtime. On WD5M AnyBURL performs 1.7 percentage points better in terms of MRR than KGT5+ComplEx, which has an overall running time of roughly 10 days while AnyBURL needs less than 4 h. On Freebase, the largest dataset, AnyBURL achieves an MRR that is 3.3 percentage points higher that the MRR that has been achieved by a ComplEX model using a hyperparameter setting found by an efficient hyperparameter search tailored for large datasets [28].

Our results show also that the modifications and extensions presented in this paper improved the previous version of AnyBURL significantly. In our experiments, we ran the old version for 10000 s. Nevertheless, when comparing the MRR results from smallest to largest dataset against using the results of running the current version for 100 s, then the current version is 2.2, 6.9, 1.7 and 4.4 percentage points better. Note also that we had to use the rule application module of the new version to generate the predictions for Freebase in less than 24 h. For that reason, we marked that runtime entry in Table 2 with *.

We also conducted experiments with the latest version of AMIE, called AMIE 3. The default setting of AMIE is rather restrictive. Thus, we added two other settings by decreasing the support threshold and by increasing the supported rule length. If we compare the size of the ruleset learned by AMIE in all of these settings with the rule sets learned by AnyBURL, we observe a significant difference. This difference is mainly caused the large number of rules with constants that are learned by AnyBURL. As mentioned above, AMIE did not terminate within 24 h on three out of four datasets when activating constants.

In contrast to AnyBURL, AMIE is not an anytime algorithm. This means that the parameter setting determines runtime. For some datasets/settings, AMIE did not finish within 24 h. We discuss the reasons and AMIE runtimes in general in the next section. Whenever AMIE generated a result within 24 h it performed clearly worse than the results obtained by the 100 s learning time run of AnyBURL. The best results have been achieved on the smallest dataset, FB15k, in the setting that uses a very low support threshold. AMIE achieves an MRR of 0.822 in only 1.6 min, which outperforms the embedding-based approaches from [27].

5.4 Runtimes and carbon footprint

In this section, we present a detailed comparison in regard to total runtimes and energy consumption with respect to the results in [27, 28, 47]. A comparison between our approach and a knowledge graph embedding model (and probably any deep-learning model), with respect to runtime and usage of computational resources, is not trivial. The required computation for training a knowledge graph embedding model is usually conducted on one or several GPUs. AnyBURL requires no access to a GPU but is multithreaded and runs on several (virtual) CPU cores. Thus, we report runtimes together with relevant characteristics of the used computing devices. Both the experiments reported in [27] and our experiments have been conducted on a CPU server with the same characteristics (mentioned above in Sect. 5.2). An exception is our experiments on Freebase where we had to use a computing environment with a bigger RAM. The models in [27] have additionally used up to 8 GPUs (GeForce RTX 2080 Ti). In addition to a discussion of runtimes, we compare the energy consumption at the end of this section. This allows us to get a rough idea about required resources within the same measure.

If we compare, for instance, AnyBURL in the 1000 s learning setting against the fastest way to run ComplEx in [27], which always performs worse in regard to predictive quality, we can see that runtimes are slightly worse on FB15k (the smallest dataset), while AnyBURL is significantly faster on other datasets: 6-15 times faster on Yago03-10, more than 18-54 times faster on WD5M, and more than 2 times faster on Freebase. The time for training the model once (no hyperparameter search) for only 10 epochs is already 2 times higher. These are significant differences. In [27], a specific method for conducting an efficient hyperparameter search has been proposed. Applying this method to Freebase results in a runtime that is 21 times longer than learning rules with AnyBURL for 100 s. However, the results of AnyBURL are nevertheless 2.1 percentage points better.

In regard to the results in [47] and FB15k, AnyBURL is 58 times faster than the best-performing model while achieving a similar predictive quality. On Yago03-10 AnyBURL performs slightly worse considering the MRR but is 7 times faster (58 times in the 1000 s learning setting) than the best-performing model. AnyBURL is 50 times faster and 1.7 percentage points better in terms of MRR on WD5M compared to the previous state-of-the-art model KGT5+ComplEx [50]. Finally, for Freebase AnyBURL is 2.5 to 20 times faster than the previous state of the art presented in [28] while outperforming this approach in regard to MRR by more than three percentage points.

The runtimes we measured for AMIE illustrate an important difference between the rule mining method of AMIE and AnyBURL. AMIE's algorithmic core is a depth-first search

that constructs all possible rules of a certain type, where a search step corresponds to adding an atom to a rule body. This basic approach is improved by using several sophisticated pruning strategies and optimizations. The authors of AMIE argue that, due to these improvements, they do not have to resort to approximations or sampling, but are able to compute the exact confidence and support of each rule. Our results show that this claim is not valid if we apply AMIE to very large datasets. This might be caused by the simple fact that the potential number of rules is increased by the number of predicates with each additional body atom. Moreover, some of these rules have billions of different groundings, which makes the confidence computation extremely costly. While pruning techniques might allow to reduce the branching factor, we can still observe a significant increase in runtimes if we compare the default setting, where rules can have only 2 body atoms, to the $l \leq 3$ setting, where we allow 3 body atoms. For FB15k runtime increases by a factor of 600, for Yago03-10 runtime increases by a factor of 193. For WD5M AMIE requires already 25 min in the default setting. If we increase the number of body atoms by one, AMIE does not finish with 24 h. For Freebase, a dataset that has a significantly higher number of predicates, AMIE does not even terminate within 24 h in the default setting.

The experiments in [27] have been conducted on a compute server with RTX 2080 Ti GPUs. These GPUs have a maximal performance of 250 Watt (Thermal Design Power). As their computational power is probably not always fully exploited, we estimate an average demand of 200 Watt for each GPU. Furthermore, we assume that the demand of the CPU can be neglected in the knowledge graph embedding setting. Both assumptions are in favor of the embedding-based approaches. Most runs of AnyBURL are executed on a CPU server with two Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz cores which are virtually divided into 40 virtual cores. Each core demands 90 Watt (Thermal Design Power); however, we estimate a demand of 100 Watt as there might be some peak loads. Thus, if our approach makes full usage of all CPU cores we estimate this as $2 \times 100 = 200$ Watt. We conducted the Freebase experiments on a more modern compute server that has a more modern CPU architecture. As we restricted the number of worker threads to 32, only one of the cores of one CPU has been used which results into a maximal electricity demand of 180 Watt. This means that AnyBURL has approximately the same electricity demand as running a knowledge graph embedding model on one GPU. We are aware that these numbers are a rough estimation.

Let us take a closer look at the two largest datasets. For Freebase, we ignore in the following the hyperparameter search, which, in this specific case, has been conducted on FB15k. The fastest setting, which performs well, is given by RotatE. In this setting, the training process has been distributed over 8 GPUs. Although training the model needs 78% of the time required by AnyBURL in the 10000 s learning setting, the power consumption is ≈6 times higher than the power consumption of AnyBURL, while RotatE's MRR results are 15 percentage points worse. Even when we consider the 100 s learning setting of AnyBURL, there is still a difference of 13 percentage points. In that setting, the power consumption of the knowledge graph embedding model is 57 times higher than that of AnyBURL.

On WD5M the differences are even more significant. Here the best results have been achieved by a setting using two GPUs for training. The hyperparameter search has been conducted with one GPU. Thus, we have for the ComplEx model that achieves an MRR of 0.308 an estimated power consumption of 63840 s × 400Watt + 136800 s × 200Watt = 52,896,000Ws = 52,896 kWs = 14.7 kWh. This amounts roughly to 1.5 days energy consumption of a 2-person average household in Germany.[6] By contrast, AnyBURL requires in the 10000 s learning setting 17031 s × 200Watt = 3406200 Ws = 3406 kWs = 0.95 kWh (100 s rule learning yields in an overall demand of 0.08 kWh). This is a significant difference, given that AnyBURL generates a result that is 4.5 percentage points better in terms of MRR (the 100 s rule learning achieves an MRR that is still 0.2 percentage points better).

We have not yet included any indirect costs, which are higher for a compute server with several GPUs compared to a compute server that runs only CPUs. This includes the costs of (and the electricity required to run) a better air-conditioning system and the additional space required in the computing center. We are aware that these costs and the carbon footprint of a knowledge graph embedding-based approach might still be justified if the results would be better compared to the results of using an efficient symbolic approach. However, our experimental results show that this is not the case. We have also not included in our calculations the fact that we picked a posteriori the combination of model and parallelization technique that worked best for the specific datasets. This is usually not known without any previously conducted experiments.

5.5 Memory consumption

We provide a brief discussion about the memory consumption of the different approaches in this section. One has to distinguish consumed disk space, RAM memory, and GPU memory with the latter being the most expensive memory type.

[6] Our calculation is based on the assumption that an average 2-person household requires 9 kWh per day. This number is based on the data available at https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Umwelt/UGR/private-haushalte/Tabellen/stromverbrauch-haushalte.html

Although embedding models and symbolic models are conceptually different, in regard to disk space complexity, it is possible to use the sizes of the learned rule set and embeddings for an approximate disk space usage comparison of model sizes. We estimate model sizes for the latent models in [28] and [27] by calculating *#embeddings_stored* $\times$ *embedding_dim* $\times$ 4 bytes. According to the authors, the embeddings are stored in FloatTensors which take 4 bytes per number stored. On the Freebase dataset, this results in 176.27 GB for ComplEx [28] and 44.07 GB for RotatE and ComplEx [27]. The rule set for AnyBURL in the largest setting (10000 s) has a size of 0.17 GB only. For WD5M, RotatE and ComplEx in [27] use 2.35 GB each whereas the rule set in the largest setting requires 0.253 GB. For Yago03-10, RotatE and ComplEX in [27] both take 0.063 GB compared to the rule set size 1.5 GB for AnyBURL. Finally, for FB15k RotatE and ComplEX in [27] take 0.0084 GB each whereas the rule set of AnyBURL in the largest setting has a size of 6.6 GB. While the embeddings sizes are correlated with the size of the datasets, the same is not the case for the rule sets. Here we have the smallest rule set for the largest dataset. This counter-intuitive observation is caused by the fact that the confidence approximation is more expensive for larger datasets, which yields less mined rules in the same time compared to smaller datasets. However, the size of the rule sets is not critical at all and even relatively small rule sets perform already very well considering the results of the 100 s runs in Table 2 for the smaller datasets.

In regard to GPU memory, AnyBURL does not use this memory type, whereas a large portion of research on large-scale embedding models is centered around the problem that the model parameters do not fit on a single GPU jointly. We therefore refer to the respective works for more details [27, 28].

AnyBURL does not need to fit its model on the GPU but runs on the cores of one or several CPUs instead. It requires RAM memory to load the dataset and to build up index structures, which allows to construct body groundings of rule bodies quickly. The internal representation of the rule set requires, compared to this, an insignificant amount of memory. To process FB15k and Yago, AnyBURL requires less than 16GB, for WD5M around 80GB are required. For the largest dataset, Freebase, AnyBURL requires around 900GB. In regard to main memory consumption, we are unfortunately not able to provide estimates for the knowledge graph embedding models; however, a multiple of the model sizes might be needed as, e.g., optimizers such as Adagrad [18] need to store the parameter values from previous iterations of the optimization procedure.

6 Ablation study

In the following, we first analyze the impact of the OI constraint. We focus on the impact on predictive quality and on the size of the mined rule sets. Then, we study the impact of different policies and reward strategies to better understand how far our approach is capable to adapt itself to the specifics of the datasets. Subsequently we analyze the differences of the approximated confidences and exact confidence computation in regard to performance on the datasets where the exact computation does terminate. Finally, we report about an experiment which indicates that rule learning and rule application scales nearly linear with respect to the available number of cores.

6.1 Object identity

For the purpose of analyzing the impact of the Object Identity (OI) constraints, we run AnyBURL under OI constraints learning rules for 100, 1000, and 10000 s. Then, we repeat the experiments without OI constraints. The results of our experiments with Yago03-10, WD5M and Freebase are shown in Table 3. In the first two data columns, we show MRR scores and the number of learned **B**-rules under OI constraints. The left section of the Table reflects the default setting of AnyBURL. In the right section of the table, we illustrate what happens if we deactivate OI constraints both in terms of learned **B**-rules and MRR. In particular, we show the change of the MRR in terms of the difference between the MRR with activated and deactivated OI-constraints. A negative score means that predictive quality decreased when deactivating the constraints.

The last column informs about the number of **B**-rules learned without OI constrains compared to the number of **B**-rules learned with OI constraint. A value of +33% means, for example, that additionally 33% more **B**-rules have been learned without OI constraints. We focus here on **B**-rules because $\mathbf{U_c}$ and $\mathbf{U_d}$-rules of length one are not affected by OI constraints. This means that the impact on the number of learned rules can only be measured in terms of **B**-rules. Moreover, learning an increased number of **B**-rules requires more computational resources compared to learning $\mathbf{U_c}$ and $\mathbf{U_d}$-rules. This results sometimes in the counter-intuitive outcome that the overall number of rules decreases when we deactivate OI constraints.

We observe that deactivating OI constraints results in a drop in MRR for all datasets and learning times. There is only one exception, which is given by a minor improvement with respect the 100 s learning time for WD5M. However, the longer we learn the more beneficial are the OI constraints. By deactivating OI constraints, we lose between 1.5 and 4 percentage points. This means that the predictive quality is clearly affected by OI constraints. At the same time, we

Table 3 Comparing the impact of deactivating the OI constraints in terms of MRR and size of the learned **B**-rule subsets

	Learn	With OI		Without OI		
		MRR	**#B**	ΔMRR	**#B**	**ΔB**
Yago	100 s	0.564	2182	−0.020	2620	+20%
	1000 s	0.565	4814	−0.020	5867	+22%
WD5M	1000 s	0.338	8835	−0.005	11037	+25%
	10000 s	0.353	31190	−0.015	41606	+33%
FB	1000s	0.707	7238	−0.035	7295	+1%
	10000 s	0.711	22463	−0.041	23890	+6%

observe that up to 33% more **B**-rules are learned. Again, we have one exception which is again related to the smallest learning time. The difference on Yago03-10 is relatively small, which can be explained by the small number of different relations used in the dataset. This results in a relatively small number of potential binary rules. The OI constraints are only affecting this type of rules. On FB15k and WD5M, the impact is much stronger. We observe an increase of 10% to 55%.

Note that the number of rules can become critical for very large datasets as all rules have to be kept in RAM together with some index structures during the prediction phase. This is also the reason why we omitted an entry for Freebase, as we were running out of memory with deactivated OI constraints. We conclude that the OI constraints have a small positive impact on the predictive quality and reduce the risk of running out of memory for very large datasets.

6.2 Impact of reward and policy

In the following, we compare the random policy against all possible combinations of policies and reward strategies. Before we start with the discussion of the results, we have to emphasize that our approach allocates computational resources between four different path profiles (cyclic paths of length one, two and three, as well as acyclic paths of length one). The random approach distributes computational effort equally between these four profiles. We will see that the random approach works surprisingly well. This is partially caused by the fact that within a profile the most important rules are found first, as these rules occur more frequently. At the same time, we have to understand that we distinguish between only four path profiles. This means that a perfect adaption to an extreme dataset, in which regularities belong to only one profile, will converge at most four times faster than the random approach. Taking both aspects into account, we can expect that an approach guided by a specific reward might yield results in the same time span that are only slightly better and the same results can be achieved by running the random baseline for a longer time.

The results of our experiments are shown in Table 4. We evaluated each setting five times and report the resulting average MRR scores. We conducted the experiments for Yago03-10, WD5M and Freebase. As Yago03-10 is, compared to the other two datasets, relatively small, we observed some variance in the results. We computed the average over five runs for that reason. We omitted the smallest datasets FB15k as results converge so quickly to the top scores presented in Table 2 that it is hard to measure differences between any of the policies. We compare each non-random reinforcement setting against the random policy for different learning times. We show, in particular, the difference of the MRR scores. A positive number means that the non-random policy performs better than the random selection of a path profile.

The weighted policy outperforms the random baseline consistently independently of the chosen reward strategy. According to our results, the best combination is the weighted policy together with the reward strategy $R_{s\times c}$. For WD5M the MRR scores is after learning rules for 10,000 s more than two percentage points higher than the random approach. For Freebase we observe constantly a performance that is between 0.7 and 0.8 percentage points better. The other two reward strategies perform worse, however, they are still superior to the random baseline.

The results of the greedy strategy are inconclusive. It performs clearly worse than the weighted policy, while it performs on Freebase a bit better than the random baseline and on WD5M slightly worse. We believe that a bad performance of the greedy approach happens especially if one path profile is in fact more important while the other path profiles (or at least some of them) yield also beneficial rules. In such a situation, the greedy approach will focus on the outstanding profile until most of its rules have been constructed. During this time span, the other rules are completely ignored aside from those rules that are mined randomly from some path profile with probability ϵ.

Figure 4 shows in detail how greedy (colored bar at the top) and weighted policy (line plot below) adapt to the specifics of the Yago03-10 dataset. The first phase (0-50 s) is dominated by mining rules that are generated from the cyclic path profile of length 1. When 100 s have passed, and after an intermediate period where mixed profiles are used, most of the cores are used to mine rules from acyclic paths of length 1 resulting in $\mathbf{U_d}$ and $\mathbf{U_c}$ rules. We can also see that cyclic paths of length two receive a low priority throughout the whole process. Thus, our approach is capable to put the focus on the most important path profile and changes its focus once the most important rules from this profile have been mined.

Table 4 Comparing policies and reward strategies against the random policy (Rand) in terms of MRR

	Learn	Rand	Greedy			Weighted		
			R_s	$R_{s\times c}$	$R_{s\times c/2^l}$	R_s	$R_{s\times c}$	$R_{s\times c/2^l}$
Yago	100 s	0.555	+0.005	+0.006	+0.007	+0.006	+0.005	+0.006
	1000s	0.564	+0.002	+0.002	+0.002	+0.002	+0.002	+0.002
WD5M	100	0.308	−0.005	−0.002	−0.002	+0.001	+0.002	+0.006
	1000	0.322	−0.002	±0	±0	+0.003	+0.016	+0.005
	10000	0.332	−0.001	±0	±0	+0.002	+0.021	+0.004
Freebase	100	0.692	+0.010	+0.002	+0.004	+0.004	+0.007	+0.004
	1000	0.700	+0.005	±0	+0.001	+0.002	+0.007	+0.001
	10000	0.703	+0.003	−0.001	±0	+0.001	+0.008	±0

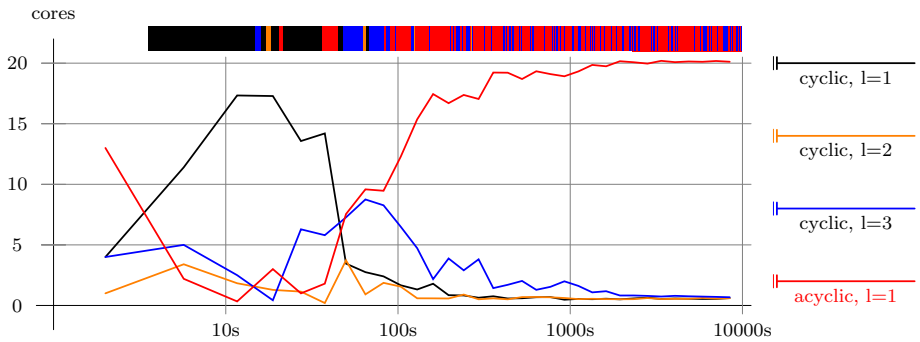

Fig. 4 Greedy (top bar) and weighted policy (lines) with reward strategy $R_{s\times c}$ applied on Yago03-10. Note that the x-axis uses a logarithmic scale

6.3 Confidence sampling

The efficiency of our approach is partially based on estimating confidences by sampling rule bodies instead of correctly computing the confidence of a rule. It is clear that this has a significant impact on runtimes, especially when we apply our method to very large datasets. Instead of enumerating up to billions of different body groundings, AnyBURL stops, in its default setting, after 100,000 attempts of constructing a body grounding. This is also backed by our experimental results, where we observed that AMIE, which computes exact confidences, does not terminate within 24 h on Freebase in its default setting. The same happens when we apply AMIE on WD5M with a setting where we increased the rule length to the default rule length of AnyBURL.

However, we do not yet know if the sampling-based approach has a negative impact on the quality of the generated rankings. According to the results presented in Table 2, it is already clear that such a negative impact must be rather limited, because otherwise it would not be possible to achieve state-of-the-art results on the two largest datasets and good results on the other two datasets. Nevertheless, we do not know if the quality of the results would be further increased if we would exchange the sampling-based confidence estimation by an exact computation.

To answer this question, we used a rule set learned by AnyBURL and replaced its approximated confidence scores with exact confidences. We computed these scores by using a complete variant of the DFS-based method that was previously used in AnyBURL. Then, we compared the predictive quality of both rule sets to understand the difference between using approximated and correct confidences on the same rule set. We conducted these experiments for the rule sets that have been learned after 1000s and for the subset that consists of **B**-rules only. If there is a systematic deviation between the exact computation and the approximation it can be expected that this happens for long **B**-rules more frequently.

The results of our experiments are depicted in Table 5. As the exact computation does not terminate within 24 h for the two largest datasets, we can present only results for FB15k and Yago03-10. For both datasets, there are nearly no differences between using exact and approximated confidences if we look at the complete rule sets. If we look at the subset, that consists only of **B**-rules, there are nearly no differences for Yago03-10. However, the exact rule computation is around half a percentage point better on FB15k. This difference is probably not noise but might be caused by our sampling technique. However, the minor negative influence of the approximated confidence scores seems to vanish if we use the full rule set. Adding $\mathbf{U_c}$ and $\mathbf{U_d}$-rules mitigates the negative impact. These results justify the use of our approximation method, which are a key element to make a rule based knowledge base completion approach applicable to very large datasets.

Table 5 Comparing the impact of using exact confidences vs. approximated confidences

		Sampled			Δ-Exact		
		hits@1	hits@10	MRR	hits@1	hits@10	MRR
FB15k	All rules	0.820	0.897	0.847	−0.002	+0.002	−0.001
	B-rules	0.805	0.877	0.829	+0.005	+0.006	+0.006
Yago	All rules	0.497	0.689	0.565	±0	±0	±0
	B-rules	0.396	0.618	0.474	−0.001	+0.001	±0

6.4 Multithreading

In the following, we report about two experiments that illustrate in how far rule learning and rule application scales nearly linear with respect to the number of available cores. First, we start with analyzing the impact on rule learning. Due to the fact that many good rules can be mined at the beginning, we cannot run AnyBURL with different numbers of threads for a fixed time comparing the number of learned rules. Instead, we propose the following procedure. We run AnyBURL with one thread until a certain number of rules have been found. Then, we repeat the experiment using 2 threads, 4, 8, 16 and 32 threads. We conducted these experiments for YAGO and for WD5M. For YAGO we stopped after 500k have been found, for WD5M we stopped after 50k rules have been found.

The results in Table 6 show that the approach scales nearly linear. By using twice as much cores, we reduce the required time to mine the same number of rules to a range from 50% to 62%. This holds for each datasets and each pair of setting where we doubled the number of worker threads. If we compare the runtimes of using one thread against the setting where we use 32 threads, the reduction of the time required to learn the specified number is 1/16 instead of 1/32. The loss is probably caused by single threads that start to compute the confidence scores of a large rule at the end of a time span which requires the remaining threads to wait. The more worker threads we use, the higher is the probability that this happens.

In the rule application phase, each worker thread picks repeatedly a completion task from a queue of all queries and generates the top-k ranking for this task until all tasks are solved. We applied the Yago and WD5M rule set to predict the candidates for the test sets once with one threads and once with 32 threads. For Yago it took 23.4 times longer in the single thread setting, for WD5M it was a factor of 22.1. An optimal factor would have been 32. While these numbers show that there is still some loss when distributing the work over several threads, the rule application scales well enough to benefit significantly from an increased number of available cores.

7 Related work

The methods AMIE [22], RuleN [36] and the previous version of AnyBURL [35] have been applied to solve knowledge graph completion tasks as first representatives from the family of rule mining approaches. RuleN was designed as a baseline to better understand which share of test queries can be solved with rather simple rules. It does not support $\mathbf{U_c}$ and $\mathbf{U_d}$ rules. AMIE was developed as a generic rule learner and has only been used in the RuleN paper as alternative rule-based baseline.

AMIE supports also $\mathbf{U_c}$ rules; however, as a complete search is conducted it is not possible to activate this rule type for larger datasets. This can be seen by the fact that for more restrictive settings the rule learning already did not terminate within 24 h for some datasets as shown in the experimental section. We also conducted experiments, not reported in Sect. 5, including rules with constants where again the mining process did not terminate within 24 h for the larger datasets. We conclude that AMIE 3 is applicable to very large datasets only in rather restricted settings which does not yield competitive results in regard to knowledge graph completion.

The surprisingly good results of RuleN and AMIE, compared to the currents state of the art back in 2018, motivated the development of AnyBURL, which supported $\mathbf{U_c}$ and

Table 6 Impact of multithreading on rule learning. Time elapsed (seconds) until 500k rules for Yago03-10 and 50k rules for WD5M have been learned using different numbers of worker threads. Runtimes for loading and indexing the dataset are excluded

Number of threads	1	2	4	8	16	32
Yago03-10, learn 500k rules	521 s	294 s	181 s	102 s	53 s	32 s
WD5M, learn 50k rules	436 s	229 s	134 s	85 s	53 s	33 s

$\mathbf{U_d}$ rules and achieved good results on the standard benchmarks [35]. However, this previous version of AnyBURL was not feasible for large datasets such as WD5M and Freebase due to conceptual and implementation issues discussed in the previous sections. In this work, we present the required adjustments for scaling the rule learner to these large datasets. Additionally, the improvements also affect predictive quality on small-/medium-sized graphs positively as shown in Sect. 5. AnyBURL's competitiveness in regard to sixteen other models is also demonstrated in [47], where the authors used the version of AnyBURL that we describe in the current paper.

The rule aggregation technique of AnyBURL has been improved in an approach called SAFRAN [44]. SAFRAN computes clusters of redundant rules and applies a Noisy-OR aggregation of these clusters to compute the final score assigned to a prediction. In [6], the authors explored in how far it is possible to improve rule aggregation by embedding the rules themselves into a multidimensional space. Both approaches improve the quality of the predictions by learning a function that aggregates rule confidences to determine the final ranking. However, both methods are also less efficient compared to the simple and robust rule application model of AnyBURL. Thus, we did not change the default maximum aggregation of the previous AnyBURL version with respect to our large scale setting.

Several rule mining approaches applicable to large knowledge graphs have been proposed in the previous years [12, 20, 43, 46]. Most of them have not been applied to knowledge graph completion tasks. Instead the quality of these approaches is mostly measured in the numbers of mined rules or by estimating how many high-quality rules (= rules with support and/or confidence above a certain threshold) have been found within a given time span. We believe that these evaluations do not help to understand in how far the overall approach is well suited for knowledge graph completion. This is also backed by our observations related to object identity, where we found that smaller rule sets generate better results.

One of the few exception is the approach called RARL (Relatedness-Aware Rule Learning), which has also been evaluated on standard knowledge graph completion tasks [46]. RARL has been inspired by an approach called Ontological Path Finding [11]. It uses TBox information, i.e., a schema that describes the relations used in the given knowledge graph, to restrict the set of possible rule candidates. The type of rules that are supported by RARL are restricted to **B** rules. Results are available for the datasets WN18RR, FB15K-237 and YAGO. The current version of AnyBURL performs on WN18RR more than 10 percentage points better and on FB15k-237 and Yago03-10 quite similar compared to RARL. A crucial point about the approach is the availability of a schema. Moreover, if the schema is rather generic and does not distinguish between different types of entities, the algorithm fails to generate rules that are beneficial for knowledge graph completion. This becomes visible in the WN18RR results.

Knowledge graph embedding models are based on latent representations of the entities and relations, i.e., the instances are assigned to elements in a low-dimensional vector space with the goal of capturing semantically relevant features. Each model is defined by a specific scoring function, and the latent representations are learned during the training of the models. RESCAL [39], DistMult [63], and ComplEx [60] are characterized by bilinear scoring functions with ComplEx augmenting DistMult to calculations in the complex vector space. RotatE [55], on the other hand, substitutes the vector product of the previous models with rotations in the complex vector space to calculate the triple scores. TransE [8] is a seminal model for the translation-based approaches, TuckEr [4] relies on tensor factorizations and HolE [38] combines circular correlations with matrix multiplications. Please note that the selection of knowledge graph embedding models in our comparative experiments is based on the best and fastest performing models in regard to large-scale evaluation in previous work. Knowledge graph embedding models are also heavily used in downstream applications such as visual relationship detection [3] and entity alignment [10]. In [52], the embeddings are utilized in a combined approach for entity and relation alignment demonstrating that these tasks also have benefits for the completion problem.

There are also several approaches that try to combine embeddings and rules. An example is the system Ruge [25], which learns rules, materializes these rules, and injects the inferred triples as new training examples with soft labels into the process of learning the embedding. The authors report results on FB15k which are worse than the results achieved by AnyBURL after 100 s. In [41, 42] the authors proposed a method to learn rules more efficiently from the embeddings space instead of learning them directly from the symbolic representation. The approach is called Rule Learning via Learning Representation (RLvLR). With respect to the task of knowledge graph completion RLvLR results are available for three datasets (FB15k-237, FB75k, and a dataset based on Wikidata). For two of the dataset we were able to run and evaluate AnyBURL in the same way as described in [41] to allow a fair comparison. The numbers of the third dataset are based on a random selection of 50 target relations, which does not allow a fair comparison. On FB15k-237, RLvLR achieved an MRR of 0.24, whereas AnyBURL achieved an MRR of 0.326. For FB75k, a subset of Freebase which has more entities than FB15k but less training triples, the RLvLR MRR is 0.337, where AnyBURL has an MRR of 0.421. Moreover, AnyBURL has been $\approx$8 times faster in our experiments compared to the runtimes of RLvLR as reported in [41]. These results so far available do not support the overall idea that

a tight integration of rule learning and knowledge graph embeddings has advantages over a pure symbolic approach.

Another interesting line of research is differentiable rule learning. Examples for this line of research are Neural-LP [64] and DRUM [49]. Instead of searching and scoring the rules in the symbolic space, rules are derived from the embedding of a given knowledge graph. This allows to convert the discrete search problem into a differentiable problem. It might be an underlying assumption that this conversion is more efficient compared to learning rules directly as it is based on differentiable operations. However, to our knowledge there exists no differentiable approach that has been applied to large datasets. An exception is given by the results for Neural-LP on FB15k, which is the smallest dataset in our experiments. While we achieve an MRR of 0.84 within 10 min, the authors report an MRR of 0.76 without specifying any runtimes. For DRUM we found only a remark that 1.2 min are required for the kinship dataset which describes around 100 entities in 10000 triples. FB15k is already more than 100 times larger in number of entities and 50 times larger in number of triples. DRUM has not yet been applied to this dataset.

Reinforcement learning has already been used for the task of query answering in [13, 32, 62]. These approaches have also been applied to knowledge graph completion. Similar to AnyBURL, they provide explanations; however, they rely on vector representations and not on symbols. Moreover, they formulate the knowledge graph completion problem as a reinforcement learning problem by learning a policy that starts at a given query variable (entity to be predicted) and using the relation (in the query) and the path history until the answer node is reached. The goal is to maximize the reward (reaching the answer node) by following an optimal path from a query to an answer node [13]. While these approaches use a reward strategy for paths that lead to answer nodes, AnyBURL uses reward strategies for path profiles that provide paths which lead to rules with high predictive power. None of these approaches has been applied to large datasets. Moreover, an evaluation protocol has been used that deviates from the standard by evaluating the predictions for tail entities only. We have adopted this protocol and computed in a 1000s run the MRR for FB15k-237, which is the largest dataset to which Minerva [13] and Multihop [32] have been applied. We measured an MRR of 0.42 for our approach, Minerva has an MRR of 0.293 and Multihop reports 0.393 or 0.407 depending on the embeddings method that has been used.

In our experiments, the main focus lies on a comparison with the results reported in [27]. This paper is motivated by the idea that large-scale knowledge graph completion, if based on standard knowledge graph embedding models, is barely possible without methods that parallelize the training process. The authors argue that parallel training methods can handle large-scale knowledge graphs and provide reasonable training times, without a (strong) negative impact on the resulting predictive quality. Note that none of the methods discussed in [27] is designed to reduce the required electricity demand but tries to reduce training times while keeping the loss of predictive quality as small as possible. We have shown that our approach is in most settings faster (compared to settings that use up to 8 GPUs in parallel), demands significantly less energy and yields better predictions.

When knowledge graph embedding models are employed in large-scale settings, a crucial cost determining factor is given by the dataset-specific hyperparameter search. The most basic search strategy (grid search) would require to fully train the selected model for every distinct configuration of a specified search space. This is mitigated on smaller graphs, for instance in [48] where Bayesian optimization is utilized to only move along promising paths in the search space. However, in cases where one single training run already poses a challenge such as on Freebase (compare Sect. 5), this is hardly applicable. Recently, the hyperparameter optimization engine $GRASH$ was introduced in [28]. Based on successive halving, $GRASH$ combines graph and epoch reduction techniques to enable hyperparameter searches for knowledge graph embedding models in large scale settings. To the best of our knowledge, $GRASH$ is the first algorithm that successfully was applied to perform a hyperparameter search on Freebase and achieved state-of-the-art results in regard to predictive quality [27]. Nevertheless, the improved version of AnyBURL presented in this work outperforms $GRASH$ on Freebase with regard to predictive quality, overall runtime, and CO2 consumption as demonstrated in the experimental section.

Another line of research is based on different classes of graph neural networks (GNNs). In [24], a knowledge graph embedding model based on a modified graph attention network architecture is proposed that accompanies a classification model designed for finding erroneous facts in a knowledge graph. The combined pipeline is trained in an active learning framework and is also used for repairing the detected false triples. Recently a branch of GNNs achieved state-of-the-art results on the knowledge graph completion task, outperforming prior work by a significant margin [65, 67]. In contrast to vanilla GNN architectures, in these approaches message passing follows a progressive propagation. In the Neural-Bellman-Ford Network [67], in short NBFNet, message passing is query dependent, that is, messages are propagated by starting from the source entity of a given query via each in- or outgoing relation. The efficiency of this approach is improved in [65] by only allowing a sampled subset of nodes to send messages in each iteration, i.e., introducing an adaptive propagation path. Nevertheless, these models are expensive to train and still need to utilize the whole training graph at inference time for message passing. For instance, the authors of NBFNet report training times of

264 min on the small WN18RR dataset, which contains less than 90k triples, when using 4 Tesla V100 GPUs under basic data parallelism. This is a larger training time than the overall time needed by AnyBURL on the largest dataset used in this work, which contains more than 330 million triples (see Table 1) and is therefore more than 3.5 thousand times larger than WN18RR. Please additionally note that a hyperparameter search would add a multiple of the training time to the overall cost of the GNN. Therefore, it can be doubted that these models are able to generate high-quality predictions in a large scale setting.

8 Conclusion

The problem of knowledge graph completion has been dominated over the last decade by approaches that are based on the use of embeddings. Even though symbolic approaches have a long history rooted in inductive logic programming, purely symbolic methods are an outsider compared to the majority of methods proposed in the context of knowledge graph completion. Against this trend, a competitive symbolic rule mining system called AnyBURL has been proposed in [35]. In this paper, we described a new version of AnyBURL. In particular, we replaced a core component, the saturation-based search procedure, by a method that uses reinforcement learning. We modified the sampling technique for estimating the confidence of a rule. We changed the semantics of the rules by interpreting them under object identify constraints. Finally, we implemented the new AnyBURL version as a multithreaded rule mining process. These modifications improve AnyBURL in general, but show their strongest benefits in the context of large-scale knowledge graph completion.

We applied the new version of AnyBURL to four large datasets. The largest of these datasets is Freebase. It consist of more than 300 million triples, that describe more than 80 million entities. It requires parallelization methods, as discussed in [27], to train a model that embeds the entities and relations of the graph into a multidimensional vector space. Even if we assume that this problem is solved, the runtime of the hyperparameter search remains a critical issue and specific search methods have to be designed to speed up the search process [28]. While large datasets are challenging for embedding based methods, these problems do not affect our symbolic approach.

We conducted an extensive evaluation to support this claim. Our results show that the new version of AnyBURL is better in terms of predictive quality, runtimes, and energy consumption compared to current state of the art for the two largest datasets we used in our experiments. Moreover, within a time span of only 100 s AnyBURL can learn rules that achieve a predictive quality that is close or even better than the best models, which require up to 10 days (the ensemble combining an encoder–decoder transformer model and ComplEx on WD5M) or 9 h (ComplEx on Freebase). These differences get even stronger if we compare the energy consumption. While we designed the new version of AnyBURL to fully exploit the capabilities of running multithreaded on a CPU server, knowledge graph embedding techniques are usually run on a GPU, or, if parallelized, on several GPUs. Depending on which settings we focus on AnyBURL requires between 6 to 180 times less electricity on the two largest datasets to achieve better or similar results than the results of the best embedding based models known so far.

The differences in the predictive quality measured in terms of MRR are partially caused by the fact that both datasets are relatively new with respect to their usage as evaluation datasets for knowledge graph completion. Thus, a good hyperparameter setting or a good hyperparameter search space is not known in advance and an extensive search is extremely expensive in terms of runtimes. As a consequence, first attempts reported in [27] achieve an MRR that is more than 15 percentage points below the MRR that we achieved in the first run. By designing a more efficient hyperparameter search it was possible to learn a good (and probably not perfect) hyperparameter setting to increase the MRR in [28]. However, this result is still several percentage points worse than the result we achieved with AnyBURL. It can be expected that at some point in time we might have a ComplEx model or another embedding-based model that achieves a higher MRR than AnyBURL. However, these results will partially be based on the dataset specific knowledge that has been accumulated via previous research effort and publications. In a realistic application scenario, we have to deal with problems and datasets that have not been in the focus of research for years and a good hyperparameter search space is usually not known. The results of AnyBURL are based on a setting which is dataset independent. It works well for the four datasets used in our experiments, which have very different characteristics. It can be expected that it will work well or at least decent out of the box for a novel large dataset.

There is still room for improving the runtime performance of AnyBURL. In this paper, we have mainly focused on the rule mining process. It will probably not be easy to speed up this part of the overall process. However, runtimes for larger datasets are also heavily affected by the prepossessing time of loading and indexing the dataset. For Freebase, we were able to achieve results better than previous state-of-the-art results within 25.2 min. Seventy-six percentage of the overall runtime were dedicated to loading and indexing the dataset. This means that a reduction of loading times would have a significant impact on the overall runtimes.

One way to further increase the predictive quality is related to the rule application model. The results published in [44] and [6] have shown that the rankings generated by AnyBURL can be improved by learning an aggregation function dif-

ferent from ranking the predictions based on the rule with highest confidence that generated the prediction. However, both approaches have a clearly increased rule application time compared to the simple model of AnyBURL. It is an open challenge how to search and apply better aggregation functions in an efficient way.

It might also be possible to improve the predictive quality by supporting other rule types than the rule types that are covered so far. We have already conducted initial experiments with negative rules, i.e., rules that have a negated head. These rules prevent or decrease the probability of certain predictions. First results on WN18RR, a small dataset which is well suited for exploring the impact of new concepts, showed that a specific type of these rules can improve the MRR by up to 2.5 percentage points. The integration of these rules is currently implemented as post-processing step. A tight integration into the rule application model of AnyBURL is missing and also challenging for the reasons mentioned in the previous paragraph.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. In: The semantic web, pp. 722–735. Springer (2007)
2. Babai, L.: Graph isomorphism in quasipolynomial time. In: Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, pp. 684–697 (2016)
3. Baier, S., Ma, Y., Tresp, V.: Improving visual relationship detection using semantic modeling of scene descriptions. In: International Semantic Web Conference, pp. 53–68. Springer (2017)
4. Balazevic, I., Allen, C., Hospedales, T.: TuckER: Tensor factorization for knowledge graph completion. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pp. 5185–5194. Association for Computational Linguistics (2019)
5. Betz, P., Meilicke, C., Stuckenschmidt, H.: Adversarial explanations for knowledge graph embedding models. In: Proceedings of the 31th International Joint Conference on Artificial Intelligence, pp. 2820–2826. Ijcai.org (2022)
6. Betz, P., Meilicke, C., Stuckenschmidt, H.: Supervised knowledge aggregation for knowledge graph completion. In: European Semantic Web Conference, pp. 74–92. Springer (2022)
7. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 1247–1250. ACM (2008)
8. Bordes, A., Glorot, X., Weston, J., Bengio, Y.: A semantic matching energy function for learning with multi-relational data. In: Machine Learning, vol. 94, pp. 233–259. Springer (2014)
9. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in neural information processing systems, pp. 2787–2795 (2013)
10. Chen, M., Tian, Y., Yang, M., Zaniolo, C.: Multilingual knowledge graph embeddings for cross-lingual knowledge alignment. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, pp. 1511–1517. Ijcai.org (2017)
11. Chen, Y., Goldberg, S., Wang, D.Z., Johri, S.S.: Ontological pathfinding. In: Proceedings of the 2016 International Conference on Management of Data, pp. 835–846. ACM, Association for Computational Linguistics (2016)
12. Chen, Y., Wang, D.Z., Goldberg, S.: Scalekb: scalable learning and inference over large knowledge bases. The VLDB J. **25**(6), 893–918 (2016)
13. Das, R., Dhuliawala, S., Zaheer, M., Vilnis, L., Durugkar, I., Krishnamurthy, A., Smola, A., McCallum, A.: Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In: Sixth International Conference on Learning Representations (2018)
14. De Raedt, L.: Logical and relational learning. Springer Science & Business Media (2008)
15. Dehaspe, L., Toivonen, H.: Discovery of relational association rules. In: Relational data mining, pp. 189–212. Springer (2001)
16. Dettmers, T., Minervini, P., Stenetorp, P., Riedel, S.: Convolutional 2d knowledge graph embeddings. In: Thirty-Second AAAI Conference on Artificial Intelligence, pp. 1811–1818. AAAI Press (2018)
17. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 601–610 (2014)
18. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research **12**(7) (2011)
19. Esposito, F., Laterza, A., Malerba, D., Semeraro, G.: Refinement of datalog programs. In: Proceedings of the MLnet familiarization workshop on data mining with inductive logic programming, pp. 73–94 (1996)
20. Fan, W., Fu, W., Jin, R., Lu, P., Tian, C.: Discovering association rules from big graphs. Proceed. VLDB Endowment **15**(7), 1479–1492 (2022)
21. Ferré, S.: Link prediction in knowledge graphs with concepts of nearest neighbours. In: The Semantic Web: 16th International Conference, ESWC 2019, Portorož, Slovenia, June 2–6, 2019, Proceedings 16, pp. 84–100. Springer (2019)
22. Galárraga, L., Teflioudi, C., Hose, K., Suchanek, F.M.: Fast rule mining in ontological knowledge bases with AMIE+. The VLDB J. **24**(6), 707–730 (2015)
23. Galárraga, L.A., Teflioudi, C., Hose, K., Suchanek, F.: Amie: association rule mining under incomplete evidence in ontological knowledge bases. In: Proceedings of the 22nd international conference on World Wide Web, pp. 413–422. International World Wide Web Conferences Steering Committee (2013)

24. Ge, C., Gao, Y., Weng, H., Zhang, C., Miao, X., Zheng, B.: Kgclean: An embedding powered knowledge graph cleaning framework. arXiv preprint arXiv:2004.14478 (2020)
25. Guo, S., Wang, Q., Wang, L., Wang, B., Guo, L.: Knowledge graph embedding with iterative guidance from soft rules. In: Thirty-Second AAAI Conference on Artificial Intelligence, pp. 4816–4823. AAAI Press (2018)
26. Katehakis, M.N., Veinott, A.F., Jr.: The multi-armed bandit problem: decomposition and computation. Math. Operat. Res. **12**(2), 262–268 (1987)
27. Kochsiek, A., Gemulla, R.: Parallel training of knowledge graph embedding models: a comparison of techniques. Proceed. VLDB Endowment **15**(3), 633–645 (2021)
28. Kochsiek, A., Niesel, F., Gemulla, R.: Start small, think big: On hyperparameter optimization for large-scale knowledge graph embeddings. In: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (2022)
29. Lacroix, T., Usunier, N., Obozinski, G.: Canonical tensor decomposition for knowledge base completion. In: Proceedings of the 35th International Conference on Machine Learning, pp. 2869–2878. PMLR (2018)
30. Lajus, J., Galárraga, L., Suchanek, F.: Fast and exact rule mining with amie 3. In: European Semantic Web Conference, pp. 36–52. Springer (2020)
31. Lerer, A., Wu, L., Shen, J., Lacroix, T., Wehrstedt, L., Bose, A., Peysakhovich, A.: Pytorch-biggraph: a large scale graph embedding system. Proceed. Mach. Learn. Syst. **1**, 120–131 (2019)
32. Lin, X.V., Socher, R., Xiong, C.: Multi-hop knowledge graph reasoning with reward shaping. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pp. 3243–3253. Association for Computational Linguistics (2018)
33. Mahdisoltani, F., Biega, J., Suchanek, F.M.: Yago3: A knowledge base from multilingual wikipedias. In: Seventh Biennial Conference on Innovative Data Systems Research. Cidrdb.org (2015)
34. McKay, B.D., Piperno, A.: Practical graph isomorphism, ii. J. Symb. Comput. **60**, 94–112 (2014)
35. Meilicke, C., Chekol, M.W., Ruffinelli, D., Stuckenschmidt, H.: Anytime bottom-up rule learning for knowledge graph completion. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, pp. 3137–3143. Ijcai.org (2019)
36. Meilicke, C., Fink, M., Wang, Y., Ruffinelli, D., Gemulla, R., Stuckenschmidt, H.: Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion. In: International Semantic Web Conference, pp. 3–20. Springer (2018)
37. Muggleton, S., De Raedt, L.: Inductive logic programming: theory and methods. J. Logic Program. **19**, 629–679 (1994)
38. Nickel, M., Rosasco, L., Poggio, T.: Holographic embeddings of knowledge graphs. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 1955–1961. AAAI Press (2016)
39. Nickel, M., Tresp, V., Kriegel, H.P.: A three-way model for collective learning on multi-relational data. In: Proceedings of the 28th International Conference on Machine Learning, vol. 11, pp. 809–816. Omnipress (2011)
40. Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., Taylor, J.: Industry-scale knowledge graphs: lessons and challenges: five diverse technology companies show how it's done. Queue **17**(2), 48–75 (2019)
41. Omran, P.G., Wang, K., Wang, Z.: Scalable rule learning via learning representation. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, pp. 2149–2155. Ijcai.org (2018)
42. Omran, P.G., Wang, K., Wang, Z.: An embedding-based approach to rule learning in knowledge graphs. Trans. Knowl. Data Eng. **33**(4), 1348–1359 (2019)
43. Ortona, S., Meduri, V.V., Papotti, P.: Robust discovery of positive and negative rules in knowledge bases. In: 34th International Conference on Data Engineering, pp. 1168–1179. IEEE (2018)
44. Ott, S., Meilicke, C., Samwald, M.: SAFRAN: An interpretable, rule-based link prediction method outperforming embedding models. In: 3rd Conference on Automated Knowledge Base Construction (2021)
45. Pallottino, S.: Shortest-path methods: complexity, interrelations and new propositions. Networks **14**(2), 257–267 (1984)
46. Pirrò, G.: Relatedness and tbox-driven rule learning in large knowledge bases. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 2975–2982. AAAI Press (2020)
47. Rossi, A., Firmani, D., Matinata, A., Merialdo, P., Barbosa, D.: Knowledge graph embedding for link prediction: A comparative analysis. ACM Transactions on Knowledge Discovery from Data **15**(2), 14:1–14:49 (2020)
48. Ruffinelli, D., Broscheit, S., Gemulla, R.: You CAN teach an old dog new tricks! on training knowledge graph embeddings. In: 8th International Conference on Learning Representations (2020)
49. Sadeghian, A., Armandpour, M., Ding, P., Wang, D.Z.: Drum: End-to-end differentiable rule mining on knowledge graphs. In: Advances in Neural Information Processing Systems, pp. 15,321–15,331 (2019)
50. Saxena, A., Kochsiek, A., Gemulla, R.: Sequence-to-sequence knowledge graph completion and question answering. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, pp. 2814–2828. Association for Computational Linguistics (2022)
51. Semeraro, G., Esposito, F., Malerba, D., Brunk, C., Pazzani, M.: Avoiding non-termination when learning logic programs: A case study with foil and focl. In: Logic Program Synthesis and Transformation-Meta-Programming in Logic, pp. 183–198. Springer (1994)
52. Singh, H., Jain, P., Chakrabarti, S., et al.: Multilingual knowledge graph completion with joint relation and entity alignment. 3rd Conference on Automated Knowledge Base Construction (2021)
53. Srinivasan, A.: The aleph manual(techical report). Computing Laboratory, Oxford University, Tech. rep. (2000)
54. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: Proceedings of the 16th international conference on World Wide Web, pp. 697–706. ACM (2007)
55. Sun, Z., Deng, Z.H., Nie, J.Y., Tang, J.: Rotate: Knowledge graph embedding by relational rotation in complex space. In: 7th International Conference on Learning Representations (2019)
56. Sun, Z., Vashishth, S., Sanyal, S., Talukdar, P., Yang, Y.: A re-evaluation of knowledge graph completion methods. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 5516–5522. Association for Computational Linguistics (2020)
57. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, 2 edn. MIT press (2018)
58. Tanon, T.P., Stepanova, D., Razniewski, S., Mirza, P., Weikum, G.: Completeness-aware rule learning from knowledge graphs. In: International Joint Conference on Artificial Intelligence, pp. 507–525. Ijcai.org (2017)
59. Teru, K., Denis, E., Hamilton, W.: Inductive relation prediction by subgraph reasoning. In: Proceedings of the 37th International Conference on Machine Learning, pp. 9448–9457. PMLR (2020)
60. Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., Bouchard, G.: Complex embeddings for simple link prediction. In: International Conference on Machine Learning, pp. 2071–2080. PMLR (2016)
61. Wang, X., Gao, T., Zhu, Z., Zhang, Z., Liu, Z., Li, J., Tang, J.: Kepler: A unified model for knowledge embedding and pre-trained language representation. Trans. Assoc. Comput. Linguist. **9**, 176–194 (2021)

62. Xiong, W., Hoang, T., Wang, W.Y.: Deeppath: A reinforcement learning method for knowledge graph reasoning. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pp. 564–573. Association for Computational Linguistics (2017)
63. Yang, B., Yih, W.t., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. In: 3rd International Conference on Learning Representations (2015)
64. Yang, F., Yang, Z., Cohen, W.W.: Differentiable learning of logical rules for knowledge base reasoning. In: Advances in Neural Information Processing Systems, pp. 2319–2328 (2017)
65. Zhang, Y., Zhou, Z., Yao, Q., Chu, X., Han, B.: Learning adaptive propagation for knowledge graph reasoning. arXiv preprint arXiv:2205.15319 (2022)
66. Zheng, D., Song, X., Ma, C., Tan, Z., Ye, Z., Dong, J., Xiong, H., Zhang, Z., Karypis, G.: DGL-KE: Training knowledge graph embeddings at scale. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 739–748 (2020)
67. Zhu, Z., Zhang, Z., Xhonneux, L.P., Tang, J.: Neural bellman-ford networks: a general graph neural network framework for link prediction. Adv. Neural Inform. Process. Syst. **34**, 29476–29490 (2021)