Rule-Based Knowledge Graph Completion

Patrick Betz
University of Mannheim, Germany

Christian Meilicke
University of Mannheim, Germany

Heiner Stuckenschmidt
University of Mannheim, Germany

Abstract

The field of knowledge graph completion is concerned with augmenting knowledge graphs with missing information. Symbolic rule-based approaches are not only efficient and interpretable but also competitive with embedding-based methods in regard to predictive quality. Rule-based knowledge graph completion can be separated into two stages, the learning stage and the application stage, which are both individually challenging. In the learning stage, horn rules are mined from a given knowledge graph. Given the vast size of the space of all possible rules, the mining approach must select relevant rules effectively. In the application stage, the mined rules are used to make new predictions which are assigned with plausibility scores. These scores need to be set by aggregating individual confidence values of rules that have the same consequence. This tutorial covers the fundamental aspects required to build a symbolic rule-based approach for knowledge graph completion. It will discuss the different rule types, mining strategies, and how to effectively apply the rules in different scenarios. Finally, we discuss practical examples for rule application by using the Python-based PyClause library.

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning

Keywords and phrases Knowledge Graph Completion, Rule Learning, Symbolic AI

Digital Object Identifier 10.4230/OASIcs.RW.2024/2025.1

Category Invited Paper

Related Version "Symbolic Rule-Based Knowledge Graph Completion". Patrick Betz. PhD-Thesis. 2025.
Full Version: `https://link.springer.com/article/10.1007/s00778-023-00800-5`
Full Version: `https://2022.eswc-conferences.org/wp-content/uploads/2022/05/paper_67_Betz_et_al.pdf`
Full Version: `https://arxiv.org/abs/2309.00306`
Full Version: `https://dl.acm.org/doi/10.1145/3583780.3615042`

Supplementary Material *Software*: `https://github.com/symbolic-kg/PyClause`
archived at `swh:1:dir:5478fe65310fa02ca44ffcd2569f4507037a3db1`

1 Introduction

Knowledge graphs describe structured information (knowledge) about certain domains. They are composed of facts or triples that express true statements, such as $prof(Obama, Politician)$, which says that the profession of Barack Obama is politician where $prof$ is short for $profession$. Knowledge graphs are employed in various fields. Some of them are of general purpose and describe people, cities, countries, movies, organisations, and other heterogeneous topics. Examples are given by Freebase [9], DBPedia [2] or YAGO [50, 43]. Some of these graphs contain up to millions of entities and billions of facts. Other knowledge graphs describe more specific domains as, for example, the biomedical domain [1].

As the evaluation of all possible facts is costly, most of the knowledge graphs provide only an incomplete description of the respective domain. For example, a knowledge graph might describe that Barack Obama has the profession politician, but it might not contain the information that he is also a lawyer. The task of constructing missing facts in these cases is referred to as knowledge graph completion or link prediction. One possibility to find missing facts is to include external resources, for example, information written on web-pages [54]. In this work, however, our focus is on inferring new facts exclusively by the help of the already existing facts in the knowledge graph. For example, we could look for the most frequent additional profession that politicians have in the graph, and infer possible additional professions for Obama.

An approach that does not use external information for finding new facts relies on the statistics, patterns, distributions, or any other kind of regularity that is present in the graph. A natural choice is to learn and apply an explicit and symbolic representation of these patterns. There is a long history of approaches that are concerned with learning symbolic representations, such as inductive logic programming [34] and relational association rule mining [13]. Nevertheless, these approaches are underrepresented in the field of knowledge graph completion. A large proportion of methods learn a low dimensional and sub-symbolic representation of the graph elements [44] and often these are coupled with a complex neural architecture [14, 63].

While in the last years deep learning methods have experienced tremendous success in various fields, the raw data representation of a knowledge graph is symbolic. The problem itself is already given in a logical representation. Intuitively, it should take little effort to apply existing frameworks that perform learning based on symbolic representations. One advantage of these methods is, as we will see throughout this work, that their fact predictions are fully interpretable. The reason of why a prediction is made, is always given by a human-understandable symbolic formula (or rule). However, symbolic approaches in this context are often perceived as being inferior in regard to predictive performance. Moreover, performing logical inference does not scale well to large datasets.

In this work, we will describe how to build and use a rule-based approach for performing knowledge graph completion. In fact, we will show that employing such a symbolic approach presents substantial challenges. For each of these, we will discuss potential solutions and open questions. We will recall the main ideas and algorithms of our rule learner AnyBURL, which has been first introduced in [31] and later been improved in [30]. Subsequently, we will discuss how to apply the rules to make new predictions efficiently. This will lead to the confidence aggregation problem, which we discussed in [8, 39, 6]. In general, we will demonstrate that it is indeed possible to build a fully rule-based approach for knowledge graph completion, which is interpretable, competitive in regard to predictive performance, and scales well to very large datasets. Some of the discussed contents are also directly based on the PhD-Thesis of the first author [4].

Before we start with the the main content of this work, we introduce some preliminaries and background information in Section 2 and Section 3. In particular, we introduce knowledge graphs and the problem of knowledge graph completion formally and describe the standard evaluation protocol. We will then discuss the basic concept of using rules for knowledge graphs and introduce some technical notations that we need in the remainder of the paper. The main parts of the work are then structured as follows:

- We recall in Section 4 the main ideas and algorithms of the rule learner AnyBURL, which has been improved with the focus of scalability in [30]. It uses sampling techniques for both constructing rules and for computing their confidence scores. The resulting procedure is an anytime algorithm which can also be applied to very large datasets.

- In Section 5, we show how to apply previously learned rules to infer new facts. Here we also discuss the confidence aggregation problem: If several rules make the same prediction, we need to compute a final score based on the confidences of these rules. We also contrast the type of rule application that we use with the use of logical inference under uncertainty with ProbLog [12]. Moreover, we will explain why we do not use it for our setting. The contents of this section were discussed first in [7, 39, 6].
- In Section 6, we present and discuss experimental results. In Section 6.1, we are concerned with standard evaluation datasets and pay special attention to the impact of using different aggregation techniques. Our results indicate that a symbolic method can achieve a predictive quality that is close to current state-of-the-art. In Section 6.2, we focus on very large datasets, where we outperform current state-of-the-art on the two largest datasets in both runtimes and predictive quality. Most of the results were published first in [39] and [30].
- In Section 7, we present the rule application framework PyClause. The Python-based library allows with a wrapper to directly learn rules with AMIE [16] and AnyBURL from Python. The main functionality offers a rich toolkit of different rule application modes, such as query answering, fact predictions, explanations generation, and rule materialization. PyClause allows for an easy augmentaion of Python-based neural implementations with rule-based functionality on knowledge graphs.

2 Background

2.1 Knowledge Graphs

Knowledge Graphs (KGs) describe certain domains by using structured predicate-subject-object facts or triples of the form $p(s, o)$. Here, p is the predicate or relation and s, o are entities. We call s the subject and o the object entity of the triple. The terms predicate and relation will be used interchangeably throughout this work. Likewise for the terms triple and fact.

An entity in the KG can be anything that one can think of. This includes individual persons or objects, such as Barack Obama, the ISS, or the Eiffel Tower, as well as abstract concepts like love, human, and building. The decision of what constitutes an entity depends exclusively on the designer of the KG. Entities will be written in lowercase letters in the following. The predicates, on the other hand, are binary relations over the entities and therefore they describe a certain relationship between the entities. It also holds here that a predicate can be anything, as long as it serves to describe a meaningful relationship between two entities. Some possibilities matching the entities given above could be, for example, *has profession*, *likes*, or *contained in*. But also more abstract relations are possible such as *similar to*, *related to*, or *part of*.

The facts contained in the KG are true statements about the world or the domain that it describes. We assume that facts that are not included are, on the other hand, not necessarily false. This corresponds to the open-world assumption about KGs [35]. Formally, we define a KG as follows.

▶ **Definition 1** (Knowledge Graph). *A knowledge graph is a tuple $(\mathcal{E}, \mathcal{P}, \mathcal{G})$ where $\mathcal{E}$ is a set of entities, $\mathcal{P}$ is a set of relations and $\mathcal{G}$ is a set of facts. Facts are written in the form $p(s, o)$ or equivalently as triples (s, p, o) with $p \in \mathcal{P}$ and $s, o \in \mathcal{E}$. Let $\mathcal{E} \times \mathcal{P} \times \mathcal{E}$ denote all possible facts, then it holds that $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{P} \times \mathcal{E}$.*

We will usually directly refer to the set of facts $\mathcal{G}$ when mentioning a KG. A KG can be represented as a directed graph where the subject of a fact is the source of an edge, the object is the target, and the relations denote different edge types. Figure 1 depicts an example KG in graph form, and Table 1 shows the triple representation. The KG describes persons, cities and countries. For example, it states that the entity *lisa* lives in *london* and speaks the language *english*.

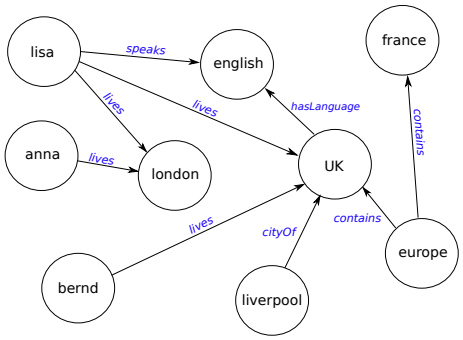

Figure 1 A subset of a knowledge graph.

Within this work we treat a KG as a flat collection of facts without an explicit underlying taxonomic structure. While a KG may contain triples such as $hasType(obama, person)$ or $subclassOf(politician, person)$, there is no conceptual distinction between, for instance, *obama* and *person*. In the context of an ontology, we might say *obama* is an instance and *person* and *politician* are classes. In the context of a KG, they are simply treated as being entities contained in the set $\mathcal{E}$. Hence, there is no explicit modelling of a schema.

Table 1 The KG serialized to a collection of facts.

Knowledge Graph		
subject	predicate	object
lisa	speaks	english
lisa	lives	UK
lisa	lives	london
bernd	lives	london
europe	contains	UK
...	...	...

2.2 Knowledge Graph Completion

The existing facts in a KG are considered to be true statements about the world. However, it is hardly possible to evaluate all possible facts from $\mathcal{E} \times \mathcal{P} \times \mathcal{E}$. Therefore, most of the existing KGs are incomplete. Formally, we can say a KG $\mathcal{G}$ is incomplete, if there exists a fact $t \in \mathcal{E} \times \mathcal{P} \times \mathcal{E}$ which is true but $t \notin \mathcal{G}$. For example, let us consider the KG given in Figure 1. We know that entity *lisa* lives in *london*. However, the graph does not contain the information that she also lives in the UK, which clearly must be true given our background knowledge. Likewise, there is no information about *bernd* and *anna* speaking any language.

The problem of Knowledge Graph Completion (KGC) aims to find missing facts given an incomplete KG. Additionally, for the setting that is considered in this work, there is no further external knowledge given. Hence, only the information already given in the original KG must be used to derive new information.

2.2.1 Ranking-Based Evaluation Protocol

There exist different evaluation protocols in the literature of how to measure the quality of a model in regard to its ability to perform KGC. Intuitively, one could simply define a classification problem in which positive facts need to be classified as true and negative facts as false. However, as discussed above, the KG does not contain negative examples. When sampling negative facts from all possible facts $\mathcal{E} \times \mathcal{P} \times \mathcal{E}$, a large number of samples would describe useless statements such as $speaks(obama, europe)$. It turns out that these are too easy for many models leading to partly meaningless evaluation results.

In this work, we will focus on ranking-based protocols instead. They can be employed by exclusively using true facts and are based on the idea that a model should rank correct facts on top of all other non-existing facts without making a particular truth statement about them. Yet, they come with an intricacy in that they are based on answering queries opposed to directly predicting the truth values of target facts. These queries are formed from known facts, as shown in the following example.

▶ **Example 2.** *We are given the true fact* $speaks(bernd, english)$*. From the fact we can form two queries. A tail query* $speaks(bernd, ?)$ *and a head query given by* $speaks(?, english)$*. For the tail query, a true answer is english and for the head query it is bernd.*

Clearly, a query does not necessarily have to be based on an existing fact. We will nevertheless stick with this convention as it aligns with the practical procedure of the protocol that will be introduced in the following paragraphs. To this end, we assume that two queries are formed from a true base fact, one in tail direction and one in head direction as shown in the example. While there may be multiple correct answers to each of the queries, the base fact provides one true answer for each direction. For a given query, we require an ordered list of answer proposals, i.e., a ranking of candidate entities from the KG.

▶ **Definition 3** (Ranking)**.** *Let* $\mathcal{E}$ *be a s set of entities,* $\mathcal{P}$ *be a set of relations and let* $p(s, o)$ *be a true fact. Let* q *denote a tail query* $p(s, ?)$ *or a head query* $p(?, o)$ *based on the fact. A* ***ranking*** *with respect to* q *is an ordered list of entities from* $\mathcal{E}$ *that serve as candidate proposals for the ? in the query.*

If the ranking is with respect to a head (tail) query, we call it a head (tail) ranking. We will equivalently also sometimes refer to the head direction or the tail direction of a ranking. Table 2 shows an example for a head and a tail ranking regarding queries formed from a base fact. We added some more languages in regard to the KG given in Figure 1. We have not yet discussed how a model or rule-based approach can create such a ranking and we will see in later sections that candidates are ranked by predicting scores for the respective triples. For the current discussion, it is enough to consider that a candidate that is ranked on top of another candidate is also more plausible. For example, in Table 2, the entity *anna* is predicted as the most plausible answer in the tail ranking.
The standard procedure for ranking-based protocols is to filter rankings with all the known facts (except of the known current answers). This is done to not penalize models erroneously.

Table 2 A tail (left) and head (right) ranking with respect to a query formed from a true base fact.

Rankings	
Base fact: *speaks*(*bernd*, *english*)	
Tail Query	**Head Query**
$speaks(bernd, ?)$	$speaks(?, english)$
Tail Ranking	**Head Ranking**
german	anna
french	lisa
italian	bernd
english	-

Table 3 Filtered tail and head rankings with respect to a query formed from a true base fact. We assume the facts $speaks(lisa, english)$ and $speaks(bernd, german)$ are known to be true. All known answers to the queries are filtered except of those given by the base fact.

Filtered Rankings	
Base fact: *speaks*(*bernd*, *english*)	
Tail Query	**Head Query**
$speaks(bernd, ?)$	$speaks(?, english)$
Tail Ranking	**Head Ranking**
~~german~~	anna
french	~~lisa~~
italian	bernd
english	-

▶ **Definition 4** (Filtered Ranking). *Let $\mathcal{E}$ be a s set of entities, $\mathcal{P}$ be a set of relations and let $t = p(s, o)$ be a true fact. Let $p(s, ?)$ be a tail query based on t and let $e \in \mathcal{E}$ be a candidate in a tail ranking with respect to the query. For the **filtered ranking**, we remove (filter) every $e \neq o$ from the ranking, if the fact $p(s, e)$ is known to be true. The definition holds likewise for a head ranking.*

Commonly the original set of facts is split into training, validation and testing subgraphs. Then, all the known facts are given by the union and therefore the filtering is performed by using the facts of all the three subgraphs. Table 3 shows the filtered rankings when the filtering is applied to Table 2. In this case, we assume that the facts $speaks(bernd, german)$ and $speaks(lisa, english)$ are known and therefore the answers *german* for the tail query and *lisa* for the head query are filtered out. We can observe that the position of the true answer *english* in the tail ranking moved from four to three and likewise in the head ranking the position of *bernd* moved from three to two.
For calculating ranking-based metrics for KGC, we need to collect the ranking positions of the true answers after filtering. Therefore, we will now formally define the ranking position of an entity in a ranking.

▶ **Definition 5** (Ranking Position). *Let $t = p(s, o)$ be a true fact. And let $q_t^o = p(?, o)$ denote the head query based on the fact and $q_t^s = p(s, ?)$ the tail query. We are given rankings for the queries and we define $\boldsymbol{rk}[e|q_t^o] \in \mathbb{N}^+$ to be the **ranking position** (the rank) of entity e in the ranking with respect to query q_t^o. The definition holds likewise for the tail query.*

With these definitions we can describe the final evaluation protocol that is used for estimating model quality in regard to KGC.

Evaluation Protocol. We assume that the facts of a KG are split into training, validation, and testing subgraphs. A given model or approach can learn its parameters or rules on the training graph possibly by using guidance from the validation graph (e.g., hyperparameter tuning). After this stage, the test facts are used for evaluation. From every fact of the test set, a head and tail query is created. The model has to propose candidate rankings for each query. The rankings are filtered with the facts from the training, validation and testing graphs. For every filtered ranking, the position of the correct candidate is tracked (e.g., the position of *english* for the tail query in Table 3 is two). Then, various metrics can be calculated from the tracked positions.

2.2.2 Metrics

Let $\mathcal{G}$ denote here the facts of a test KG. Let us assume that we are given the filtered rankings for all queries based on the facts of $\mathcal{G}$ and let the definitions from the previous section apply. Then we have,

$$\text{MR} = \frac{1}{2|\mathcal{G}|} \sum_{p(s,o)\in\mathcal{G}} \Big(\mathbf{rk}[o|q^s] + \mathbf{rk}[s|q^o] \Big),$$

$$\text{MRR} = \frac{1}{2|\mathcal{G}|} \sum_{p(s,o)\in\mathcal{G}} \left(\frac{1}{\mathbf{rk}[o|q^s]} + \frac{1}{\mathbf{rk}[s|q^o]} \right),$$

$$\text{Hits@k} = \frac{1}{2|\mathcal{G}|} \sum_{p(s,o)\in\mathcal{G}} \Big(\texttt{Ind}\big\{\mathbf{rk}[o|q^s] \leq k\big\} + \texttt{Ind}\big\{\mathbf{rk}[s|q^o] \leq k\big\} \Big).$$

Where $\texttt{Ind}\{..\}$ is an indicator that evaluates to one if the condition in the braces is true and it evaluates to zero otherwise. In the definitions q^s (q^o) refers to the tail (head) query based on fact $p(s, o)$ where we dropped the reference to the fact for brevity.

The MR is simply the average of all the ranking positions of the correct answers. The MRR is closely related. It is between zero and one (higher is better) and it gives higher weights for better ranking positions. For instance, an improvement by one position has higher effect on the MRR if the improvement is from position two to one compared to an improvement from 100 to 99. The Hits@k metrics measures in how many percent of rankings the true answer was within the top-k candidates. The Hits@1 metric can also be interpreted as the multi-class accuracy. The section is concluded with an example regarding the defined metrics.

▶ **Example 6.** *We assume that the evaluation KG consists of only one fact given by* $speaks(bernd, english)$ *as in Table 3 and we consider the filtered rankings in the table. Then we have that,*

$$\text{MRR} = \frac{1}{2 \cdot 1}(\frac{1}{3} + \frac{1}{2}) = 0.42,$$
$$\text{MR} = \frac{1}{2 \cdot 1}(3 + 2) = 2.5,$$
$$\text{Hits@1} = 0,$$
$$\text{Hits@2} = 0.5 \quad .$$

3 Rules for Knowledge Graphs

A KG represents structured data by the use of symbolic elements that we defined as entities and relations. When considering the graph representation shown in Figure 1, we can *walk* from one entity to another via a path that is given by the relations. If there exist multiple paths in the KG with the same structure of relations but varying entities, we may call this loosely a regularity that is present in the graph. For instance, we might consider all cities of countries ($cityOf$ relation) that are contained in some continent (*contains* relation). Or we might collect all persons that live in a country that has an official language ($hasLanguage$ relation). Now we might observe for these persons that most of them speak a particular language. When we generalize this observation, we can conclude that a person speaks a language if it is the official language of the country in which they live. This brings us to symbolic rules as they are used within this work.

3.1 Rules

In this section, we will describe and define symbolic rules as they are understood in this work while in the subsequent section, we will bridge the gap to a complete description in first-order logic. In general, rules are composed of logical atoms.

▶ **Definition 7** (Atom). *An atom a is an expression of the form $a = p(\tau_1, ..., \tau_k)$ where p is a relation of arity k and $\tau_1, ..., \tau_k$ are terms where a term is either a constant or a variable.*

We exclude functions as parts of terms, as functions are usually not used within a KG. We will treat the entities $\mathcal{E}$ as logical constants and likewise the atom relations are taken from $\mathcal{P}$. Naturally, the facts of the KG are considered to be ground atoms, i.e., atoms containing two constants. Moreover, we will exclusively consider atoms $a = p(\tau_1, \tau_2)$ with relations of arity two as this aligns with the relations of a KG. Then, we can define a rule as follows.

▶ **Definition 8** (Rule). *Let $a_0, ..., a_n$ be atoms over binary relation. A **rule** is a formula of the form*

$$a_0 \leftarrow \bigwedge_{i=1}^{n} a_i \quad .$$

*We say a_0 is the **head** of the rule and $\bigwedge_{i=1}^{n} a_i$ is the **body** of the rule. The rule **length** n is the number of body atoms.*

There are a few things to consider regarding the definition. First, we do not allow negated atoms in this form. The rules that we consider are horn rules. They can equivalently be written as a disjunction of atoms. Here, all body atoms are negated whereas the head atom

is not negated. Second, for better readability, we write the head of the rule on the left-hand side. Third, when we write out specific rules, we will chain the body atoms simply with a comma instead of the conjunction symbol as shown in the following example:

▶ **Example 9.** *Consider the following rules where X, Y and A are variables and london and english are constants (entities),*

$$speaks(X, Y) \leftarrow lives(X, A), hasLanguage(A, Y)$$
$$speaks(X, english) \leftarrow lives(X, london)$$
$$contains(X, Y) \leftarrow contains(X, A), cityOf(Y, A) \quad .$$

The first rule says that a person speaks a language if they live at a place (country) which has the language as official language. The second rule says that somebody speaks english if they live in london. The third rule says that a place is contained in an area if that place is a city of another place that is already contained in the area.

We can observe that the variable order within the body atoms is arbitrary. For example, in the first rule A is positioned before Y and vice versa for the third rule. Changing the positions of two variables within a rule atom will nevertheless change the meaning of the rule.

We can now also see that rules are built with path templates over a KG. The body of the first rule describes a path template that takes one step via the *lives* relation and another step via the *hasLanguage* relation. In Figure 1 (shown in the previous section), we can find two paths that match the template: From *bernd* to UK to *english* and from *lisa* to UK to *english*. Also the head of the rule can be viewed as a path template of length one, e.g., from *lisa* to *english*. Together, the head and the body of the first rule form a template for a cycle in the graph. For instance, the cycle composed of the entities *lisa*, UK, and *english* where the edge direction does not play a role.

3.2 Rules and Logic

Let us consider the first rule of Example 9. In first-order logic, we could write the rule as

$$\forall x, y, z : \; lives(x, z) \wedge hasLanguage(z, y) \rightarrow speaks(x, y) \quad . \tag{1}$$

The head is written on the right-hand side. Additionally, the variable quantification is explicitly described. In fact, we will assume that throughout the work, all variables are implicitly universally quantified if not specified otherwise.

In expression 1, it is not forbidden that distinct variables will refer to the same entities. While this is common practice in logic programming, we make a strict assumptions and force distinct variables to be unequal. This is denoted *object identity* and becomes relevant when introducing substitutions in the next section. More details are also discussed in Section 4. In expression 1, we would need to add $x \neq y \wedge x \neq z \wedge y \neq z$ to the formula.

3.3 Rule Predictions

If we knew about a mechanism that can automatically learn good rules (whatever that means) from a KG, we could already learn something about the patterns in the KG by just looking at the rules. Alternatively, we might simply define rules based on our background knowledge about the KG, such as $friendOf(X, Y) \leftarrow friendOf(Y, X)$. However, in either of

these cases we might ultimately be interested in deriving new statements with the rules, i.e., making fact predictions. As we will see throughout many parts of this work, rule predictions are human understandable and therefore have an advantage regarding purely neural models.

Clearly, we could perform inference with rules by using logical entailment. However, using full logical entailment is not feasible in the context of large KGs and millions of rules (more details in Section 5) and we are therefore more interested in the relatively cheap one-time application of rules. We will in the following define the concepts needed for making fact predictions with rules.

▶ **Definition 10** (Substitution). *Given a collection of variables $X, Y, A...$, a (ground) substitution θ is a mapping from the variables to constants.*

As discussed previously, for the constants we can readily use the entities from a KG. Recall the concept of object identity introduced in the last section. In practical terms, we are only concerned with substitutions where distinct variables map to distinct constants. We proceed with an example in regard to the variables of a rule.

▶ **Example 11.** *Let X, Y, A be variables and $bernd, english$ and UK be entities from a KG. Consider the rule*

$$speaks(X, Y) \leftarrow lives(X, A), hasLanguage(A, Y).$$

One possible substitution θ for the variables is given by:

$$\{X \mapsto bernd,\ A \mapsto UK,\ Y \mapsto english\}.$$

In our case, we merely apply substitutions to the rule atoms. If we apply a substitution to an atom, we obtain a ground atom and in our case it will be a fact of a KG. We are in particular interested in the distinction of the head and the body of the rule.

▶ **Definition 12** (Grounding). *Let us assume we are given a rule r and a substitution θ. The **head grounding** of the rule is the fact that results when applying θ to the head of the rule. Similarly, we define the **body grounding** as the set of facts, that result when applying θ to the body atoms of the rule.*

We will continue with Example 11.

▶ **Example 13** (continued). *Applying the substitution θ from Example 11 to the rule results in the head grounding:*

$$speaks(bernd, english).$$

And it results in the body grounding:

$$\{lives(bernd, UK),\ hasLanguage(UK, english)\}.$$

When a rule contains entities from $\mathcal{E}$ already, they are left unchanged. With the help of body groundings, we can start defining what we mean with a rule prediction. A rule makes a prediction always with respect to a KG and we will also use the phrase 'the rule is applied to the KG'. Intuitively, a rule body causes a rule head and therefore we have to look for body groundings of the rule that are true with respect to the KG, i.e., body groundings that are contained in the KG.

▶ **Definition 14** (Rule prediction). *Let $\mathcal{G}$ be the facts of a KG. Let r be a rule and let t be a target fact. We say that t is a* ***prediction*** *of the rule with respect to $\mathcal{G}$ if there exists a substitution such that the resulting body grounding is contained in $\mathcal{G}$ and t is the head grounding.*

In terms of logic programming the prediction of a rule is also termed the immediate consequence of the rule with respect to some knowledge base. We continue with an example.

▶ **Example 15.** *Consider the rule $speaks(english, X) \leftarrow lives(X, london)$ and consider the KG shown in Figure 1. For the substitution $\{X \mapsto lisa\}$. We obtain the body grounding $\{lives(lisa, london)\}$ and the head grounding $speaks(lisa, english)$. Given that the body grounding is contained in the KG, $speaks(lisa, english)$ is a prediction of the rule.*

Usually we want to have a more fine-grained view on the predictions and we therefore separate them into true predictions and predictions with unknown truth value.

▶ **Definition 16** (True prediction). *Let $\mathcal{G}$ be a KG, let r be a rule and let t be a fact. We say that t is a true prediction of r with respect to $\mathcal{G}$ if it is a prediction and it holds that $t \in \mathcal{G}$.*

We can say that t is true if it is contained in the KG as we defined the facts of the KG as true statements. A prediction could also be true even if it is not contained in the KG. However, in this case we cannot be sure and rely on external knowledge. We conclude the section with an example that shows different predictions of a rule with regard to the KG shown in Figure 1.

▶ **Example 17.** *Consider the rule $speaks(english, X) \leftarrow lives(X, london)$ and consider the KG shown in Figure 1. Then it holds that $speaks(lisa, english)$ and $speaks(anna, english)$ are predictions of the rule with respect to the KG whereas only the former is a true prediction. The fact $speaks(bernd, english)$, for example, is not a prediction of the rule.*

The previous definitions show that we can derive new facts with rules by using a computationally manageable approach. In fact, we only need to perform a one-time application of every rule instead of using logical entailment which would rely on multiple reasoning steps.

3.4 Uncertainty and Rule Confidences

Thus far, our viewpoint on rules and their predictions was strict, with any given prediction being considered as either true or false. For a more fine-grained view, we need to be able to assess the quality of a rule and its predictions. We can do so by introducing an uncertainty measure associated with the rules.

▶ **Definition 18** (Support). *The support of a rule r with respect to a KG $\mathcal{G}$ is the number of true predictions of the rule.*

$$support(r) = |\{t \mid r \text{ predicts } t \text{ w.r.t. } \mathcal{G} \ \wedge \ t \in \mathcal{G}\}| \tag{2}$$

▶ **Definition 19** (Confidence). *The confidence of a rule r with respect to a KG $\mathcal{G}$ is the number of true predictions divided by the number of all predictions.*

$$conf(r) = \frac{support(r)}{|\{t \mid r \text{ predicts } t \text{ w.r.t. } \mathcal{G}\ \}|} \tag{3}$$

Equation 3 is the standard confidence definition described in many works [17, 30]. We can see that in cases where two different rule body groundings belong to the same prediction (same head but different body groundings) they are not counted multiple times. Only

unique predictions, i.e., unique head groundings, are counted. There also exists variations of this confidence formulation in which a constraint is imposed on what is counted in the numerator [17, 53].

The confidence serves as a quality measure for the rules. Intuitively, when we define rules for a KG or when we learn them automatically, we want to focus on the rules with high confidences. It can be interpreted as the likelihood that a prediction made by the rule is true. Therefore, for a given prediction t, if it is not contained in the base KG, we can assign it with a likelihood score given by the confidence of the rule which made the prediction. This leaves us with the question of which score to assign to a prediction t when different rules made this prediction which will be discussed in Section 5.

4 Rule Learning with AnyBURL

In this section, we describe the rule learning algorithm of AnyBURL, which has been state-of-the-art in rule based knowledge graph completion for the last five years. AnyBURL has first been introduced in an IJCAI paper in 2019 [31]. Some important aspects of the rule mining algorithm have been changed meanwhile. The most accurate description can be found in a VLDB Journal paper [30] published 2024. The following section is too a large extent taken from the description that can be found this paper. However, we modified structure and content at several places.

4.1 Supported Types of Rules

AnyBURL does not learn arbitrary Horn rules but is tailored to learn only few specific types of rules. However, empirical results have shown that these types of rules are sufficient to achieve very good results in the context of KGC. In this section, we introduce the supported rule types and explain their dependencies. We distinguish between three types of rules that we call binary rules ($\mathbf{B}$), unary rules ending with a dangling atom ($\mathbf{U_d}$), unary rules ending with an atom that includes a constant ($\mathbf{U_c}$), and unary rules with an empty body ($\mathbf{U_z}$). Rules of type $\mathbf{U_z}$ are unusual and deviate from the standard semantics. We explain this rule type briefly at the end of this section.

These are the types of rules supported by AnyBURL. We discuss more specific examples later.

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

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

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

$$\mathbf{U_z} \qquad h(A_0, c) \leftarrow$$

Where $h, b_i \in \mathcal{P}$ are relations, the A_i are variables for all i, and $c, c' \in \mathcal{E}$ are entities. We do allow that $c = c'$. In contrast to binary rules, the head atom $h(A_0, c)$ in unary rules contains a constant/entity 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 rules that belong to one of the four rule types introduced above as path rules, because the body atoms, starting from the head of the rule, form a path. Note that our language bias also includes rule variations with flipped variables in the atoms: given a knowledge graph $\mathcal{G}$, a path of length n is a sequence of n triples $p_i(c_i, c_{i+1})$ with $p_i(c_i, c_{i+1}) \in \mathcal{G}$ or $p_i(c_{i+1}, c_i) \in \mathcal{G}$ with $c_i \in \mathcal{E}$ 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 where the relation used in the head of the rule appears in one ore several body atoms. The rule $contains(X, Y) \leftarrow contains(X, A), contains(A, Y)$ is an example.

B and $\mathbf{U_c}$ rules are a special form of closed connected rules. Closed connected rules can be learned by the rule mining system AMIE described in [16, 17]. $\mathbf{U_d}$ 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 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. This means that AMIE and AnyBURL are rather similar with respect to the supported rule types, however, there are also some rules that can be learned by AMIE, which cannot be learned by AnyBURL and vice versa.

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 introduce some new relations compared to Section 3. We briefly list and explain the relations in the following:

- $hypernym(X, Y)$ - 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
- $hasLang(X, Y)$ - X is the official language of country Y

Examples for binary rules, Rules (4) and (5), 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 (5). Rule (6) 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 (7), which says that an actor is someone who acts (in a film).

$$hypernym(X, Y) \leftarrow hyponym(Y, X) \tag{4}$$

$$prod(X, Y) \leftarrow prod(X, A), sequel(A, Y) \tag{5}$$

$$g(X, female) \leftarrow married(X, A), g(A, male) \tag{6}$$

$$profession(X, actor) \leftarrow actedin(X, A) \tag{7}$$

As defined in the previous Section, the *confidence* of a rule is calculated as the fraction of body groundings that result in a correct head grounding based on a reference KG. That is, the number of true predictions, divided by the number of all predictions. It is important to understand the relation between the three rule types. For that purpose, consider the following set of rules (fictitious confidence scores added in square brackets).

$$speaks(X, Y) \leftarrow lives(X, A), hasLang(Y, A) \quad [0.8] \tag{8}$$

$$speaks(X, english) \leftarrow lives(X, A) \quad [0.62] \tag{9}$$

$$speaks(X, french) \leftarrow lives(X, france) \quad [0.88] \tag{10}$$

$$speaks(X, german) \leftarrow lives(X, germany) \quad [0.95] \tag{11}$$

Rule (8) states that X speaks a certain language Y, if X lives in a country A where Y is the official language. Rule (9) 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 (9) 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 (9) can be generated from Rule (8) by removing the second atom in the body and replacing Y in the head with the constant *english*. Likewise, Rules (10) and (11) 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 (9)–(11) differ from the confidences of the more general rule. The fact that some of these rules might allow to derive the same prediction, already points us to the necessity of aggregating their confidences to achieve a final prediction score. This will be discussed in Section 5.

So far, we have not been talking about the $\mathbf{U_z}$ rule type. Here are some examples for these rules.

$$speaks(X, english) \leftarrow \quad [0.623] \tag{12}$$

$$speaks(X, french) \leftarrow \quad [0.102] \tag{13}$$

$$speaks(X, german) \leftarrow \quad [0.089] \tag{14}$$

Strictly speaking, it is not correct to call these formulas rules. Their intended semantics does not coincide with our formal understanding of an empty rule body. If we understand an empty rule body as something that is always true, then Rule (12) would allow us to entail that everything speaks *english* with a probability of 62.3%. Instead of that, we define the semantics of these rules only in the context of a completion query. Suppose we have a tail query $speaks(anna, ?)$, then we say that Rule (12) always predicts for such a query *english* as candidate. But the rule is never activated for a head query as it would predict every possible entity. We will describe query answering with rules more closely in Section 5.1.

Thus, $\mathbf{U_z}$ rules as Rule (12), (13) and (14) reflect the prior probability of X speaking a specific language, given that X speaks some language, which is implied from the query.

In [30] we explained that all of these rules are explained under object identity [48]. This means that each distinct term, which is in our language bias a variable or a constant, is assumed to be unequal from any other term that appears in the same rule. This means that we have to omitted several inequalities when we on the previous pages For example, rule (5) to be written down as follows, if we make these inequalities explicit.

$$prod(X, Y) \leftarrow prod(X, A), sequel(A, Y), X \neq A, A \neq Y, X \neq Y \tag{15}$$

In the following we will also omit these inequalities for the sake of brevity. Object identity helps to avoid that trivial dependencies have an unintended influence on the confidence of a rules. This is explained in details in Section 3.3 in [30].

4.2 Rule Mining

In the following, we first give a sketch of the AnyBURL algorithm for mining rules, before we describe its four steps, which are repeated until a given time span passed, in detail. This requires us to introduce 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 is used as a basis for deriving more general rules by replacing some of the constants by variables. AnyBURLs algorithm is a loop over the following steps:

1. Sample a path from a given knowledge graph.
2. Construct a bottom rule from the sampled path.
3. Derive general rules from the bottom rule.
4. Compute confidences and store rules above a threshold.

Before we explain this approach in detail, we contrast it with two alternative rule mining techniques used in Aleph [49] and AMIE [17, 16].

4.2.1 Alternative Rule Mining Techniques

The above sketch of our approach looks similar to the algorithm implemented in Aleph [49]. 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 [17, 16] 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 yet, 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 within an acceptable time frame, as it needs to construct all possible rules systematically.

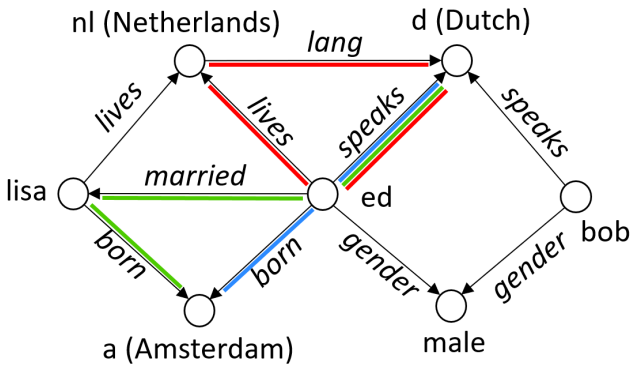

Figure 2 A knowledge graph $\mathcal{G}$ used for sampling paths. We marked the path that corresponds to Rule (16) blue, Rule (17) green, and Rule (18) red.

4.2.2 Path Sampling and Bottom Rule

Figure 2 shows a small subset of a knowledge graph $\mathcal{G}$. We use it to demonstrate how rules for the relation *speaks* can be learned. We construct a bottom rule of length n starting with the edge that corresponds to $speaks(ed, d)$. This triple will be the head of the rule. 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 Figure 2, 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 bottom rules (16), (17), and (18) given below.

$$speaks(ed, d) \leftarrow born(ed, a) \tag{16}$$

$$speaks(ed, d) \leftarrow married(ed, lisa), born(lisa, a) \tag{17}$$

$$speaks(ed, d) \leftarrow lives(ed, nl), lang(nl, d) \tag{18}$$

So far, we explained the sketch of the path sampling procedure with an example. The details will be discussed in the following. 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 $\mathbf{B}$ 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 [33, 31] 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 [41]. We might also compute paths up to a certain length using the Floyd-Warshall algorithm in a preprocessing step. However, for our purpose a simple modification of the random walk procedure is sufficient.

4.2.3 Generalization of Bottom Rules

We argue in the following that any generalization of a path of length $n+1$ will be a **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 analysing the generalization lattice rooted in Rule (17), depicted in the lower part of Figure 3. Note that AnyBURL does not compute the lattice, but directly instantiates that types of rules that we highlighted within the lattice using a yellow background color.

Each child node in the lattice results from one of two generalization operations.

1. Replace all occurrences of a constant by a fresh variable.
2. 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 Figure 3 with a yellow background color. 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 Figure 3.

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.

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 Section 4.2.5, 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.

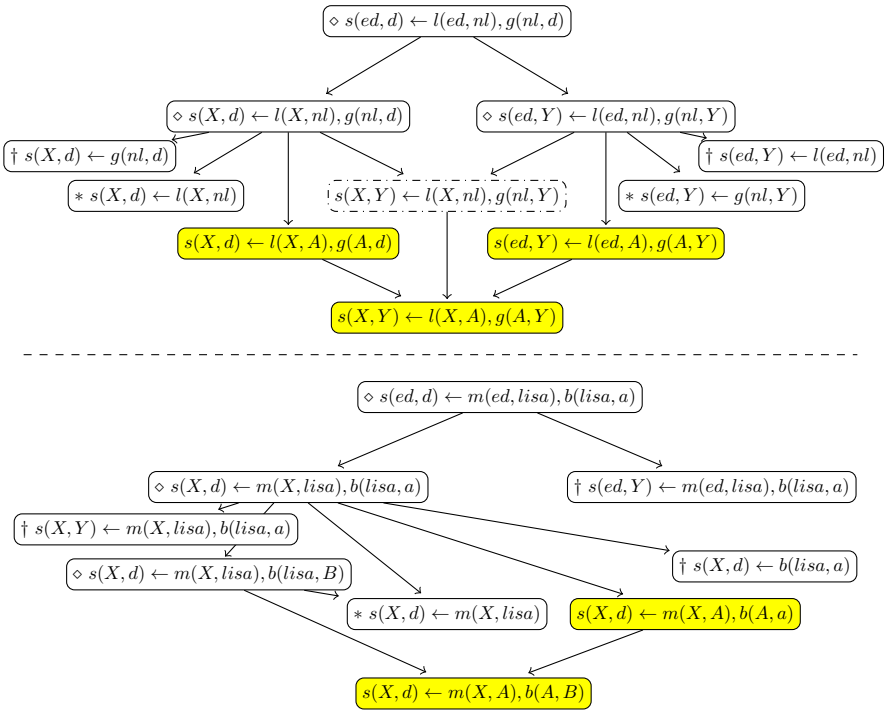

Figure 3 In the upper half we depicted the generalization lattice of the cyclic path $(s(ed,d), l(ed,nl), g(nl,d))$, in the lower part the generalization lattice of the acyclic path $(s(ed,d), m(ed,lisa), born(lisa,a))$. For legibility we use the abbreviations $s = speaks$, $m = married$, $b = born$, $l = lives$ and $g = lang$.

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. As already stated above, we depicted the relevant rules within the lattice to emphasise that the approach of AnyBURL is in that sense complete that it constructs all generalizations, which can be used to make a new prediction, by directly instantiating these rule types instead of setting up the full lattice.

4.2.4 Confidence Sampling

Confidence and support have been introduced in Section 3 as standard metrics for measuring the quality of a rule. By repeating the steps described above multiple times, AnyBURL constructs a high number of rule candidates. However, AnyBURL stores only those rules that have a confidence and support above or equal to a certain threshold. This means that we have to compute the confidence scores for a very large number of rules. With respect to our example these are the rules marked with a yellow background.

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 straight forward 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 [31]. 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. Meanwhile, the sampling procedure has been modified. A DFS based search suffers from the problem that, if a hub entity as *usa*, *female* or *berlin* is visited, all (or most) sampled paths might pass through this hub entity and the confidence of a rules as $h(X,Y) \leftarrow b_1(X,A), b_2(A,B), b_3(B,Y)$ might be estimated as the confidence of the more specific pattern $h(X,Y) \leftarrow b_1(X,usa), b_2(usa,B), b_3(B,Y)$. Therefore, the confidence would not inform about the general regularity.

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 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 easily with some 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 atoms that are chained via variables A_0 to A_n. The starting points of the search are possible substitutions 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 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 results in 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 pairs in Φ 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 pair is sampled that we found already previously. We count how often this happens in β, which is reset to 0 whenever a new pair has been found. 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 substitution of A_0, over the chain of body relations to a substitution 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 results in a grounding for the head of r. In particular, we approximate the confidence of the rule as the fraction of pairs in Φ for which $h(x,y) \in \mathcal{G}$ holds.

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

Algorithm 1 Sampling pairs that result into body groundings.

Require: $r = h(A_0, A_n) \leftarrow b_1(A_0, A_1), ..., b_n(A_{n-1}, A_n)$

function SAMPLEBODYGROUNDINGS(r)
  $\alpha \leftarrow 0$
  $\beta \leftarrow 0$
  $\Phi \leftarrow \{\}$
  $\mathbb{X} \leftarrow \{x \mid b_1(x, a_1)\}$
  while $(\alpha < max_\alpha) \wedge (|\Phi| < max_\Phi) \wedge (\beta < max_\beta)$ **do**
    $\alpha \leftarrow \alpha + 1$
    $x \leftarrow$ random element from $\mathbb{X}$
    $y \leftarrow$ BEAM(1, x, r, {})
    if $y \neq$ **NIL then**
      if $\langle x, y \rangle \in \Phi$ **then**
        $\beta \leftarrow \beta + 1$
      else
        add $\langle x, y \rangle$ to Φ
        $\beta \leftarrow 0$
      end if
    end if
  end while
  return Φ
end function

function BEAM(i, v, r, $\mathbb{P}$)
  if $v \in \mathbb{P}$ **then** ▷ checks the OI constraint
    return NIL
  else
    add v to $\mathbb{P}$
  end if
  if $i > n$ **then** ▷ n is the number of body atoms
    return v
  else
    $\mathbb{V} \leftarrow \{a_i \mid b_i(v, a_i)\}$
    if $\mathbb{V} = \emptyset$ **then**
      return NIL
    else
      $v \leftarrow$ random element from $\mathbb{V}$
      return BEAM(i+1, v, r, $\mathbb{P}$)
    end if
  end if
end function

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. Algorithm 1 makes it possible to approximate confidences values for very large knowledge bases were the exact computation of the confidence of a long **B** rule might require to construct an enormous amount of body groundings.

4.2.5 Path Sampling within a Reinforcement Learning Paradigm

Up to now, we know that the main loop of AnyBURL samples paths, converts them into rules, and esimates their confidences. Rules that have a confidence above a certain threshold are stored and can later be used to predict missing triples. However, there are different types of paths and, thus, when calling the path samplig procedure we need to specify two parameters. Paths can vary in their length and we can distinguish between cyclic and acyclic paths. In the following we use the notion of a path profile. A path profile determines the length of a path and whether it is a cyclic or acyclic path. In each iteration of the main loop, we have to decide from which profile to sample paths. AnyBURL updates the policy that selects the path profiles after a certain time span. In the default setting this happens every five seconds. Moreover, on most computing devices AnyBURL will run several threads in parallel. Thus, AnyBURL needs to decide for each time span how to distribute the different path profiles over the available threads.

In the following we consider the path sampling problem as a special kind of multi-armed bandit problem [22]. This means that we have to decide about the policy that guides the overall process and how to quantify the reward associated to a learned rule. 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 [30], we developed different reward strategies. All of them 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, for example, 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. Another reward strategy is based on the multiplication of the number of correct predictions by their confidence. In an extension of that reward strategy, the rule length can be taken into account to favour short rules over longer rules. This reward strategy is the default setting of AnyBURL. Experimental results showed that all of these strategies work well and there are only slight differences with respect to the predictive quality of the learned rule set.

All of these reward strategies can be combined with one of the following two policies. The first policy is a well known policy referred to as ϵ-greedy policy [52]. 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 threads 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, that have not been assigned randomly, to the path profile that generated the rule set that yielded the highest reward the last time it has been selected. This approach only be applied if there is a previous time span for each

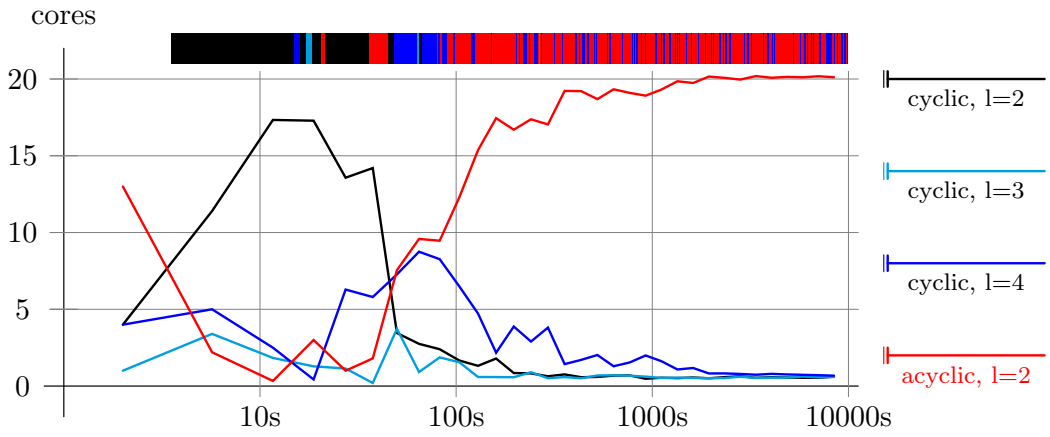

Figure 4 Greedy (top bar) and weighted policy (lines) with the reward strategy that multiplies support and confidence applied on the datasets Yago03-10 [15, 28]. Note that the x-axis uses a logarithmic scale.

profile where that profile has achieved a reward. This is obviously not the case in the first time span t_1. For that reason we associate ∞ as reward to each profile for an artificially introduced time span t_0. This results into a random selection in t_1 and ensures that each profile has been chosen once after the first few 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.[1] 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 be used to 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 path profile 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. The weighted policy is the default setting of AnyBURL.

AnyBURL uses in its default setting the weighted policy and distinguishes between five path profiles: Cyclic paths of length two to four to generate $\mathbf{B}$ rules of length one to three, acyclic paths of length two to generate $\mathbf{U_c}$ and $\mathbf{U_d}$ rules with only one body atom, and paths of length one to sample $\mathbf{U_z}$ rules. Figure 4 illustrates a typical behaviour. We omitted

[1] In Section 4.2 of [30] we explained that AnyBURL uses a canonical rule representation. While it is in principle a non-trivial problem to decide if two rules are equivalent, this is not the case for the language bias of AnyBURL. Thus, we can check in constant time if we found an equivalent rule in an earlier step.

acyclic paths of length one for generating this visualisation as this path profile is always fully exhausted after a few time steps. We also modified the default setting by updating the policy every second. The upper part show the greedy policy, which focuses on cyclic paths of length two for the first ten seconds. After a mixed phase, where the algorithm flips between different profiles, the algorithm focuses mainly on acyclic paths of length 2 from second 100 to 10000. The weighted policy, which is depicted below the bar of the greedy approach, shows a similar behaviour.

5 Rule Inference

After a set of rules is learned on a KG, we can use it for making different forms of fact predictions. Recall our definition regarding a rule prediction (Def 14) from Section 3. A fact t is predicted by a rule r, with respect to a KG, if there exists a substitution such that the rule's resulting body grounding is contained in the KG and the head grounding is equal to t. This definition builds the foundation of every scenario in which we want to put the rules into use. Nevertheless, we have to take into account further theoretical and practical considerations which will be discussed in this section. We will also discuss in detail, when performing KGC, why we cannot just put the rules into a suitable off-the-shelf framework for reasoning under uncertainty.

In general, whenever we talk about making prediction with rules, we also use the phrase *rule application*. Moreover, practically, we directly calculate rule groundings instead of explicitly defining substitutions. Throughout this section, we will use the following example of rules.

▶ **Example 20.** *Let wf denote the relation worksFor and locIn the relation locatedIn. Consider the following rules.*

r_1 [0.64] : $wf(X,Y) \leftarrow internAt(X,Y)$
r_2 [0.44] : $wf(X,Y) \leftarrow studentAt(X,A),\ locIn(A,B),\ locIn(Y,B)$
r_3 [0.41] : $wf(X,Y) \leftarrow studentAt(X,A),\ cooperatesWith(A,Y)$

The numbers in brackets are the rule confidences. The first and third rule are quite easy to understand. The second rule expresses that a person might work for a company if that company is located at the same place where this person went to university.

In the following, we let $\mathcal{R}$ denote a set of rules learned on the facts $\mathcal{G}$ of a KG. For a given target fact t, we define $\mathcal{R}_{\mathcal{G}}(t) \subseteq \mathcal{R}$ as the subset of rules that predict t with respect to $\mathcal{G}$.

5.1 Rule Application

There are different possibilities of how to make predictions with rules. We distinguish between three prediction modes, *materialization*, *triple prediction* and, based on our definition of KGC, *query answering*.

For *materialization*, our viewpoint starts with a given rule r and KG $\mathcal{G}$. We want to compute all facts that are predicted by the rule with respect to $\mathcal{G}$. We do this by iterating over all possible distinct body groundings of the rule and confirming individually if they are contained in $\mathcal{G}$. If a body grounding is contained, we store the respective head grounding as one prediction. The set of all computed fact predictions is termed the materialization of the rule.

For *fact prediction*, we are given a particular target fact t, the KG $\mathcal{G}$ together with rule set $\mathcal{R}$. We have to find all the rules (if any) from $\mathcal{R}$ that predict t w.r.t. $\mathcal{G}$. That is, we have to compute the set $\mathcal{R}_{\mathcal{G}}(t)$. For every relevant rule, we have to compute the possible body groundings. [2] However, in this case, the target triple already constrains the possible groundings. For instance, if the target triple is $worksFor(lisa, google)$, then we only have to look for groundings in which the underlying substitution maps $X \mapsto lisa$ and $Y \mapsto google$.

Query answering corresponds to calculating multiple fact predictions for a given query. We already introduced queries and ranking-based evaluation in Section 2.2. In this prediction mode, we are given a query q, the KG $\mathcal{G}$ and the set of rules $\mathcal{R}$. As above, we compute all possible body groundings for each of the relevant rules. The entity within the query constrains the valid body groundings. For instance, if the query is $worksFor(lisa, ?)$, we only search for body groundings for which it holds that $X \mapsto lisa$. If we find a body grounding that is contained in $\mathcal{G}$, we read of the entity that is assigned to Y in the underlying substitution, say c', and we store the fact prediction $worksFor(lisa, c')$. We term c' a candidate proposal for the query and we can use the candidate, for example, in a ranking.

5.2 Deterministic Confidence Aggregation

For any of the discussed prediction modes, we need to calculate fact predictions by searching for respective body groundings of a particular rule. We also discussed already that we can assign as a prediction score the confidence of the rule that made the prediction. In practical scenarios, however, it rarely occurs that a fact prediction t is predicted by only one rule. In this case it holds that $|\mathcal{R}_{\mathcal{G}}(t)| > 1$. In some cases, a target fact can even be predicted by hundreds of rules simultaneously. Therefore, the question arises of how to assign fact prediction scores, when a fact is predicted by multiple rules. These prediction scores are especially important when we want to perform fact classification or for creating ordered candidate rankings for queries.

▶ **Definition 21** (Confidence Aggregation Problem)**.** *Let t be a target fact, let $\mathcal{G}$ be the facts of a KG, and let $\mathcal{R}$ be the set of learned rules. The subset of rules that predict t w.r.t. $\mathcal{G}$ is denoted by $\mathcal{R}_{\mathcal{G}}(t)$. Let $k = |\mathcal{R}_{\mathcal{G}}(t)|$. Each rule in $\mathcal{R}_{\mathcal{G}}(t)$ has a confidence which we enumerate with an index $\{conf_1, ..., conf_k\}$. The* ***confidence aggregation*** *problem is concerned with finding a scoring function $\phi : [0, 1]^k \to \mathbb{R}$, which maps the confidences of the predicting rules to a real-valued plausability score.*

We will make this explicit with an example regarding the rules from Example 20.

▶ **Example 22.** *Let us assume that all three rules from Example 20 predict anna to work for google. The scoring function ϕ takes as input the three confidences $\{0.64, 0.44, 0.41\}$ and should output one value. This value should also reflect if, e.g., anna is more likely to work for google compared to a person which was only predicted by the first two rules to work for google.*

It might be tempting to simply sum up the confidences which corresponds to weighted voting. However, when rules are learned automatically, many rules are to some extent redundant. Consider the two rules:

$$[0.82] \quad speaks(X, english) \leftarrow lives(X, london)$$
$$[0.81] \quad speaks(X, english) \leftarrow lives(X, UK)$$

[2] We only have to check for rules that have as a head relation the same relation of the target fact.

If the first rule predicts a person to speak *english*, the second rule will provide little additional value to this prediction given the relationship of UK and $london$. Accounting for both rules in the confidence aggregation will therefore lead to an overestimation of the score. On the other hand, consider the two rules

$$[0.05] \quad playsFor(X, germany) \leftarrow bornIn(X, germany)$$
$$[0.30] \quad playsFor(X, germany) \leftarrow playsFor(X, bayern\_munich)$$

The rules describe in which cases a football player might play for the german national team. In this case, if the first rule predicts a player to play for $germany$, the second rule will still provide additional evidence if it makes the same prediction. If we only would consider the confidence of the first rule, the final score would be underestimated.

5.2.1 Aggregation based on the Maximum Confidence

An easy approach is given by defining the final score as highest confidence of all the rules that predicted a fact. We always assume in the following that a reference KG $\mathcal{G}$ and a learned set of rules $\mathcal{R}$ are given. Likewise, as above, t is a target fact and $\mathcal{R}_{\mathcal{G}}(t)$ denotes the set of rules that predict t.

▶ **Definition 23** (Max-Aggregation). *The Max-Aggregation score $\phi_M(t)$ is calculated according to the rule with the highest confidence of all predicting rules:*

$$\phi_M(t) = \max\{conf(r) \mid r \in \mathcal{R}_{\mathcal{G}}(t)\}. \tag{19}$$

Max-aggregation was first used in the context of KGC by Galárraga et al. [16]. Clearly, only one rule contributes to the final score. This will naturally result in many ties, i.e., many facts or candidates are assigned with the same score as we argued in [31]. Therefore, when a ranking of candidates is desired, Max-aggregation can be adjusted to Max+ aggregation which takes into account more than one rule for the discrimination of candidates.

▶ **Definition 24** (Max+ Ranking.). *A Max+ ranking orders candidate predictions lexicographically with respect to the confidences of their predicting rules.*

For the Max+ strategy, like in Max-aggregation, the predicting rule with the highest confidence still is of highest importance. For example, a prediction made by only one rule with confidence 0.8 would be ranked on top of a prediction that is made by hundreds of rules with a confidence of 0.6. Empirically, the adaption from Max-aggregation to Max+ has a strong effect on predictive performance in the context of KGC.

5.2.2 Aggregation based on Multiplication

If smaller sets of rules are learned, it can be beneficial to take all rules into account that predicted the target fact.

▶ **Definition 25** (Noisy-or aggregation). *The Noisy-or score $\phi_N(t)$ is calculated as the Noisy-or product over the predicting rules:*

$$\phi_N(t) = 1 - \prod_{r \in \mathcal{R}_{\mathcal{G}}(t)} (1 - conf(r)). \tag{20}$$

The Noisy-or product originates from Bayesian networks where it is used to express independent causes [42] and it was proposed by Galárraga et al. [16] for KGC. The empirical robustness of Noisy-or aggregation depends largely on the rule redundancies present in the learned set of rules as the score for a fact increases by construction in the number of predicting rules. In cases where many rules are learned and many redundancies exist, the aggregation function might perform relatively poor. This can be mitigated, however, by only allowing for a fixed number of confidences of the predicting rules: the top-h number of confidences [6].

▶ **Definition 26** (Noisy-or top-h). *Let $\mathcal{R}_{\mathcal{G}}(t)^h$ denote the set of the h rules with the highest confidences that predict target t. The Noisy-or top-h score is calculated as the noisy-or product over these top-h rules:*

$$\phi_{Nh}(t) = 1 - \prod_{r \in \mathcal{R}_{\mathcal{G}}(t)^h} (1 - conf(r)). \tag{21}$$

It is easy to see that Noisy-or top-h constitutes a compromise between Max-aggregation and Noisy-or aggregation. The score calculated with Noisy-or top-h will always be equal or larger compared to the Max-aggregation score and it will be smaller or equal compared to the Noisy-or score. This is especially beneficial as we already discussed that Max-aggregation will likely provide an underestimation and Noisy-or an overestimation of the predicted score. We will conclude this section with an example containing all the previously defined aggregation functions.

▶ **Example 27.** *We consider the rules from Example 20. Let us assume that anna is predicted by all three rules to work for google, while lisa is predicted by only the second and third rule to work for google. The aggregation scores for anna are:*

$$\phi_M\big(worksFor(anna, google)\big) = 0.64$$
$$\phi_N\big(worksFor(anna, google)\big) = 0.88$$
$$\phi_{N2}\big(worksFor(anna, google)\big) = 0.80.$$

And for lisa, the aggregation scores are:

$$\phi_M\big(worksFor(lisa, google)\big) = 0.44$$
$$\phi_N\big(worksFor(lisa, google)\big) = 0.67$$
$$\phi_{N2}\big(worksFor(lisa, google)\big) = 0.67.$$

5.3 A Probabilistic Perspective on Confidence Aggregation

Recall the definition of the rule confidence from equation (3). It is defined as the number of true predictions a rule makes, divided by the overall number of predictions of the rule. We can interpret this as the likelihood estimate of the parameter of a Bernoulli distribution. Therefore, a rule can be represented by a random variable while the confidence is an estimate for the probability that this random variable is true. However, as we have seen earlier, multiple rules can be learned from a KG with possible dependencies. Therefore, we assume that there might exist an underlying joint distribution over all rules. That is, the confidence is merely an estimate for a marginal probability with respect to the joint distribution. Finally, we cannot observe the truth values of the rules from the KG directly, and we treat them as latent variables in the following.

We will slightly abuse notation for this section and overload some of the previously defined symbols. In particular, we treat a set of rule objects, e.g., $\mathcal{R}$, as a collection of Bernoulli random variables. Likewise we treat facts as Bernoulli random variables. For every fact in a

given KG $\mathcal{G}$, we set its probability to one. Let $N = |\mathcal{R}|$ be the number of learned rules. We seek to calculate the probability that a target fact $t \notin \mathcal{G}$ is true, given the observed facts $\mathcal{G}$. We write this probability as $P(t|\mathcal{G})$. As the rules are latent, we calculate it by marginalizing over all possible rule realisations

$$P(t|\mathcal{G}) = \sum_{\mathbf{r} \in \{0,1\}^N} P(t|\mathbf{r}, \mathcal{G}) P(\mathbf{r}|\mathcal{G}), \tag{22}$$

where $\mathbf{r}$ is a possible realisation of the truth values of all rules. The conditional probability $P(t|\mathbf{r}, \mathcal{G})$ is set deterministically. We set it to one, if there exists at least one rule in $\mathbf{r}$ that is true and it predicts t and to zero otherwise. The underlying assumption is that a prediction must be true if the corresponding predicting rule is known to be true. However, we cannot observe if an individual rule is true, as mentioned above. This is reflected by the more problematic joint distribution over the rules given the observed KG, $P(\mathbf{r}|\mathcal{G})$.

In [6], we have shown that depending on certain assumptions made about the joint distribution $P(\mathbf{r}|\mathcal{G})$, we can exactly recover the Max or Noisy-or aggregation scoring functions from equation (22). While it is intuitive that Noisy-or aggregation has a probabilistic interpretation, it is more surprising for Max-aggregation.

To recover Max-aggregation, we need to introduce the Fréchet-Hoeffding bound which defines the maximal achievable correlation of two random variables [21]. Let π_i and π_j be the marginal probabilities for two Bernoulli variables, then it holds for the correlation ρ_{ij} that $\rho_{ij} \leq U(i,j)$ with

$$U(i,j) = \min \left\{ \left(\frac{\pi_i(1-\pi_j)}{\pi_j(1-\pi_i)} \right)^{1/2}, \left(\frac{\pi_j(1-\pi_i)}{\pi_i(1-\pi_j)} \right)^{1/2} \right\}. \tag{23}$$

We can now assume that the pairwise correlations of all the rules in $\mathcal{R}$ regarding the joint $P(\mathbf{r}|\mathcal{G})$ are maximal. The values for the marginals in equation (23) are simply estimated with the individual rule confidences. This leads to the following result [6].

▶ **Theorem 28.** *Let $\Omega \in [-1,1]^{(N,N)}$ be the correlation matrix for the random variables representing all the learned rules. If, for every entry ρ_{ij} of Ω it holds that $\rho_{ij} = U(i,j)$, then a unique distribution for $p(\mathbf{r}|\mathcal{G})$ is induced, such that the query probability is equal to the Max-aggregation score, i.e., $p(t|\mathcal{G}) = \phi_M(t)$.*

Max-aggregation was previously believed to merely be a computational heuristic but the theorem shows that it has indeed a probabilistic interpretation in which all rules are assumed to be maximally correlated.

Not surprisingly, to retrieve the Noisy-or scoring function, we need to make an assumption from the opposite end of the spectrum.

▶ **Proposition 29.** *If the N rules in $p(\mathbf{r}|\mathcal{G})$ are mutually independent, then the target probability is equivalent to Noisy-or aggregation, i.e., $p(t|\mathcal{G}) = \phi_N(t)$.*

The proofs of the results and a more detailed treatment is provided in [6]. In Section 5.2.2, we mentioned that Max-aggregation tends to underestimate the final score for a target fact whereas Noisy-or scoring results in an overestimation. This is confirmed by the theoretical results of this section regarding the inferred probabilities. Quite intuitively, maximal correlation or mutual independence are both assumptions that are too strict in most cases.

5.4 Learnable Aggregation

Instead of defining deterministic aggregation functions like in the previous sections, we can also cast the whole problem into a supervised learning setting. Our goal is then to learn a function that takes as input the rules that predicted a fact and outputs a real valued score. We let ϕ_{sv} denote such a supervised scoring function. Then, rules can act as features which was likewise already proposed in [18]. Particular parameterizations of the scoring function are based on different design decisions that we will discuss in the following. First we will describe how a labelled dataset can be created by using rule application. Subsequently, we will discuss some possible specifications for ϕ_{sv}. A more detailed treatment and extensive experimental results can be found in [8, 39].

5.4.1 Data Construction

We assume that we are given a training KG $\mathcal{G}^*$ on which a set of rules $\mathcal{R}$ is learned. Our goal is to create a labelled dataset with labels for facts $y \in [0, 1]$ and features given by the predicting rules. As previously, $\mathcal{E}$ and $\mathcal{P}$ are the sets of entities and relations of the KG, respectively.

For each fact $t \in \mathcal{G}^*$, a head and a tail query is formed. For example, for $t = p(s, o)$, we obtain the tail query $p(s, ?)$ and the head query $p(?, o)$ with $p \in \mathcal{P}$ and $s, o \in \mathcal{E}$. We will continue with the tail queries; the procedure is the same for the head queries.

For each tail query $p(s, ?)$, all fact predictions $s(p, o')$ are calculated with the set of rules and with respect to $\mathcal{G}^*$. That is, we use the prediction mode *query answering*. The intricacy is that the queries are created from the same KG on which the rules are also applied. For every predicted fact $t' = s(p, o')$ where $o' \in \mathcal{E}$, the set of predicting rules $\mathcal{R}_{\mathcal{G}^*}(t')$ is stored. Moreover, for every predicted fact, we additionally check if it is a true prediction, i.e., if $t' \in \mathcal{G}^*$. If it is true, we can label it with one, otherwise we label it with zero.

After performing the same procedure with the head queries, we obtain a labelled dataset $\mathcal{D} = \{y_i, t_i, \mathcal{R}_{\mathcal{G}^*}(t_i)\}_{i=1}^{|D|}$ where $|D|$ is the number of constructed examples, $y_i \in [0, 1]$ is the label, t_i is the underlying fact, and $\mathcal{R}_{\mathcal{G}^*}(t_i)$ is the rule feature set. Note that $|D|$ is much larger than the number of facts in the training KG, $|\mathcal{G}^*|$, as the dataset also contains negative examples.

5.4.2 Learnable Scoring Functions

The Noisy-or aggregation is based on the predefined confidences as described in Section 5.2.2. A natural extension is to learn the confidences directly on the labelled dataset. To that end, every rule in $\mathcal{R}$ is assigned with one learnable parameter or weight $\beta \in \mathbb{R}$. Let β_r denote the learnable parameter for rule $r \in \mathcal{R}$. We can define the scoring function under the Noisy-or specification as follows.

$$\phi_{sv}(t) = 1 - \prod_{r \in \mathcal{R}_{\mathcal{G}^*}(t)} \Big(1 - \sigma(\beta_r)\Big) \tag{24}$$

Here, σ is the Sigmoid function and $\sigma(x) \in [0, 1]$ for $x \in \mathbb{R}$. The calculated 'score' is by construction contained in the interval $[0, 1]$. Therefore, we can learn the parameters $\boldsymbol{\beta}$ on the dataset $\mathcal{D}$ by, for instance, defining a likelihood-based loss function such as binary cross-entropy and using gradient-based optimisation. Furthermore, we argued in [39] that equation (24) is generalized by a simple linear model such as a logistic regression,

$$\phi_{sv}(t) = \sigma\Big(\beta_0 + \sum_{r \in \mathcal{R}_{\mathcal{G}^*}(t)} \beta_r\Big), \tag{25}$$

where β_0 is a global intercept term which also can be defined relation-wise. Again, we can learn the parameters β when defining a loss criterion and using gradient-based optimisation. We also proposed a formulation in which the parameters are squared such that the elements within the sum become non-negative.

The two presented formulations are simple. Potentially, the scoring function can be arbitrary complex. One possibility is to use multi-dimensional latent representations for rules. In this case, a rule is not assigned with only one parameter but a vector $\boldsymbol{\beta}_{\mathbf{r}} \in \mathbb{R}^d$ where d is the vector length. As we discussed in [8], the scoring can then be executed, for example, by a transformer encoder [56] with a scoring layer on top (termed *Dense* in the experimental section). However, the complexity of the scoring function might lead to less model interpretability. While the formulations in equations (24) and (25) provide full interpretability of a calculated score, this can not be guaranteed for a transformer. We also proposed a compromise in [8] that is based on embeddings of rules, on the one hand, but still provides interpretability through a scoring formulation with a strong inductive bias (termed *Sparse* in the experimental section).

5.5 Rule Application and Reasoning

We mentioned already that using full logical entailment for performing inference with rules learned from KGs is not feasible in settings where millions of rules are learned. Nevertheless, in this section, we will discuss the question of what are the benefits when using a full reasoning paradigm. A suitable candidate approach is ProbLog [12] as it allows for logical inference under uncertainty and it closely resembles the probabilistic modelling we discussed in Section 5.3. ProbLog calculates the probability that a fact is true by summing up the probabilities of all logic programs that entail the fact. The probability of a logic program is given by a product over the clause probabilities that are contained in the program and the counter probabilities of clauses not contained in the program (but contained in the knowledge base). We can readily use ProbLog for performing inference with rules. In fact, we treat rule confidences as clause probabilities and the KG as evidence, i.e., ground facts with probability one. We continue with an example.

▶ **Example 30.** *We are given a learned rule set $\mathcal{R}$, containing three rules.*

$$[0.82]: \quad speaks(X, english) \leftarrow lives(X, london)$$
$$[0.81]: \quad speaks(X, english) \leftarrow lives(X, UK)$$
$$[0.78]: \quad lives(X, UK) \leftarrow lives(X, london)$$

The numbers in brackets are the rule confidences. Additionally, we are given the following KG $\mathcal{G}$ on which the rules can be applied.

$$\mathcal{G} = \{lives(marta, london),\ lives(marta, UK)\}$$

Finally, we are given the target query $speaks(marta, ?)$. The only resulting fact prediction for the query is $speaks(marta, english)$.

In Section 5.2, we argued already for a similar example that the second rule may not provide additional evidence if we know already that the first rule made the prediction. Indeed, the Noisy-or score for the target fact is relatively high with $1 - (1 - 0.82)(1 - 0.81) = 0.9658$. On the other hand, the Max-aggregation score is 0.82. For both aggregation functions, the third rule does not play any role as it does not predict the target.

```
0.82::speaks(X,english) :- lives(X,london).
0.81::speaks(X,english) :- lives(X,uk).
0.79::lives(X,uk) :- lives(X,london).

lives(marta,london).
lives(marta,uk).

query(speaks(marta,english)).
```

Figure 5 ProbLog program based on Example 30.

It is tempting to believe that performing proper reasoning can exploit the given relationship between living in *london* and living in the UK and and will result in a more accurate prediction score. Therefore, we translate the example into a ProbLog program shown in Figure 5. The resulting query probability is exactly the same as calculated by Noisy-or aggregation (0.9658)[3]. Also for ProbLog, the third rule does not modify the query probability in this particular example as $lives(marta, uk)$ is already contained in the evidence.

It is easy to modify the example such that Noisy-or and ProbLog calculate different results and we will show a an example below. In cases where a target fact is not directly predicted by a rule, ProbLog potentially can nevertheless derive the fact via the reasoning steps. There are many cases where such a behaviour is beneficial. However, the cases that one faces with millions of rules learned in the context of KGC, are most often of the form of Example 30. The typical challenge in these scenarios is the aggregation of the confidences of many rules that make the same prediction. We have seen that introducing reasoning to the inference mechanism does not help in these cases. Making the prediction in the first place, on the other hand, is less of a concern. Furthermore, it is always possible to express reasoning steps involving multiple rules by using one rule with more body atoms.

An alternative scenario is given when two rules make the same prediction although one rule is a more general form of the other rule, say, $likes(X, Y) \leftarrow friends(X, Y)$ and $likes(X, lisa) \leftarrow friends(X, lisa)$. If both rules predict *bernd* and *lisa* to be friends, then Noisy-or aggregation as well as ProbLog would simply aggregate both rule confidences with the Noisy-or product.

We continue with a theoretical result given in [6] that shows the relationship between Noisy-or aggregation and inference with ProbLog.

▶ **Theorem 31.** *Let $\mathcal{G}$ be a KG and let $\mathcal{R}$ be a set of rules annotated with confidences. Let t be a target fact with $t \notin \mathcal{G}$. In regard to ProbLog, $\mathcal{G}$ is the evidence, and $\mathcal{R}$ denotes the annotated clauses. Then it holds that the query probability calculated by ProbLog for t is larger or equal than Noisy-or aggregation.*

The proof is provided in [6]. It follows from the fact that ProbLog sums the probabilities of all programs that entail the target fact. This includes 1) the programs that entail the target fact but do not contain a clause that directly predicts the fact and 2) programs in which some clause directly predicts the fact. Noisy-or, on the other hand only takes into account 2). We will explain this in more detail with an example provided in Figure 6 which is a modification of Example 30.

[3] The example is calculated with the ProbLog online editor.

```
0.82::speaks(X,english) :- lives(X,london). (r1)
0.81::speaks(X,english) :- lives(X,uk).     (r2)
0.79::lives(X,uk) :- lives(X,london).       (r3)

lives(marta,london).

query(speaks(marta,english)).
```

Figure 6 ProbLog program based on a modification of Example 30.

When ignoring the probabilities in the example, $speaks(marta, english)$ is predicted directly by r_1. This can also be captured by Noisy-or. The target fact is also predicted by r_2 after considering that rule r_3 will derive $lives(marta, uk)$, which is not captured by Noisy-or. Indeed, the Noisy-or score is $0.82 = 1 - (1 - 0.82)$. The query probability calculated with ProbLog is larger and results in 0.935182. We show in Table 4 how this probability is calculated. Subsequently, we explain how it is also possible to derive the Noisy-or score from the table.

Table 4 Fine-grained probability calculation with ProbLog based on the program given in Figure 6. The target fact is $t = speaks(marta, english)$.

Program $\mathcal{L}$	**$\mathcal{L} \models t$**	**Probability**	
$\{\}$	No	-	-
$\{r_1\}$	Yes	$0.82 \cdot (1 - 0.81) \cdot (1 - 0.79)$	$= 0.032718$
$\{r_2\}$	No	-	-
$\{r_1, r_2\}$	Yes	$0.82 \cdot 0.81 \cdot (1 - 0.79)$	$= 0.139482$
$\{r_3\}$	No	-	-
$\{r_1, r_3\}$	Yes	$0.82 \cdot (1 - 0.81) \cdot 0.79$	$= 0.123082$
$\{r_2, r_3\}$	Yes	$(1 - 0.82) \cdot 0.81 \cdot 0.79$	$= 0.115182$
$\{r_1, r_2, r_3\}$	Yes	$0.82 \cdot 0.81 \cdot 0.79$	$= 0.524718$
			$\sum$ 0.935182

The first column enumerates every possible logic program that can be formed from the given set of rules. The evidence is not taken into account as every ground fact implicitly is annotated with a probability of one. The second column encodes if the respective program entails the target fact. The third column calculates the probability of the logic program given in the first column. The probability is calculated as the product of the probabilities (counter probabilities) of the clauses contained (not contained) in the respective program. The target probability under ProbLog is calculated by summing up all the probabilities of programs that entail the target which is shown in the bottom row of the table.

We can interpret Noisy-or, on the other hand, as the counter probability of the probability that all rules that directly predict the target are false. This is the same as the probability that at least one of the rules that predict the target is true. We can easily calculate this from the table. In fact, we have only one rule (r_1) that predicts the target. Therefore, we have to sum up all probabilities in which this rule is assumed to be true. This corresponds to all the rows in which r_1 is contained in the logic program. The resulting sum is calculated as $0.032718 + 0.139482 + 0.123082 + 0.524718 = 0.82$ and is equivalent to the value we

already retrieved for Noisy-or when calculated via the standard formulation. The proof for Theorem 31, is intuitively based on the observation that, with Noisy-or, the summation involves fewer or equal as many rows compared to ProbLog.

6 Experimental Results

In the previous sections, we discussed how rules can be learned from a KG and what challenges arise when performing inference with rules. In this section, we present some selected results regarding the discussed approaches and methods in the context of KGC. For a more fine-grained view on the experimental results and further details, we refer to the respective publications [30, 39, 8]. The experiments are separated into two parts depending on the size of the KGs. In Section 6.1, we evaluate on moderately sized KGs (80k-270k facts) and we evaluate the deterministic aggregation functions and the learnable formulations. In Section 6.2, we evaluate larger KGs (1 million - 300 million facts) where we report results for the deterministic aggregation approach Max+, which is the most efficient approach for large graphs.

The used evaluation protocol is described in Section 2.2.1 and we report the filtered metrics MRR, Hits@1, and Hits@10. For filtering, all facts from the training, validation, and testing KGs are used. Rankings for the test sets are created by performing prediction mode *query answering* (Sec 5.1) while the rules are applied with respect to the training KG only. Candidates are ordered within the rankings according to the aggregated prediction scores and ties are resolved randomly.

6.1 Moderately Sized Graphs

6.1.1 Models and Datasets

We use the datasets *Fb15k-237*, *Codex-M*, and *WN18RR*. The KG *Fb15k-237* [54] is a more challenging subset of FB15K [10], where inverse relations are removed to prevent a trivial inference of test triples from the training set. *WN18RR* [15] is a subset of WN18 [10], which contains relations between lexical entities. Similar to Fb15k-237, inverse relations are excluded. *Codex-M* is a KG extracted from Wikidata [57] and Wikipedia and belongs to the Codex benchmark [46]. It was created with the motivation to have a more difficult dataset compared to the previous ones. Summary statistics of the datasets are presented in Table 5.

The rules are learned with AnyBURL on the training KGs. In particular, we use the rule sets provided in [29]. The number of learned rules are shown in Table 5. These are used for the deterministic aggregation functions. For the learnable approaches, slightly smaller rule sets are used. When the aggregation function is learned, the validation graph is used for hyperparameter searching. For the deterministic aggregation functions, the validation graph is not used. Table 6 contains the results and we will briefly introduce the compared methods.

The last section of the table describes the methods presented in Section 5. *Max+* denotes the Max+ ranking strategy, *NO* denotes Noisy-or, and *NO top-5* is the Noisy-or top-h approach with $h = 5$. These approaches are described in Section 5.2. Furthermore, the learnable Noisy-or model is *NO (learned)* as defined in equation (24), LR denotes the linear model defined in equation (25), and *LR+* denotes a specification of the linear model where the learnable parameters are squared. Finally, *Dense* is based on rule embeddings with a transformer encoder, *Sparse* is based on embeddings and a simpler scoring function, and *D+S* is an ensemble of the two. These methods are briefly discussed in Section 5.4.2. More details, including training specifics and hyperparamter searches, can be found in the original publications [39, 8].

Table 5 Datasets and summary statistics. The first two columns show the number of entities and relations, respectively. The middle three rows show the number of facts in the training, validation, and test KG. The last row shows the number of learned rules with AnyBURL where the rulesets are taken from [29]. M means million.

Dataset	$\|\mathcal{E}\|$	$\|\mathcal{P}\|$	#Train	#Valid	#Test	$\|\mathcal{R}\|$
Fb15k-237	14 505	237	272 115	17 535	20 466	5.084M
WNRR	40 559	11	86 835	3 034	3 134	97 329
CoDEx-M	17 050	51	185 584	10 310	10 311	7.409M

In the first two sections of the table, we provide comparisons in regard to related work. The first section includes various knowledge graph embedding (KGE) models. They are based on latent representations (embeddings) of the entities and relations combined with a particular scoring function. We report results for RESCAL [36], TransE [10], DistMult [60], ComplEx [55], ConvE [15], RotatE [51], and TuckER [3]. All of these results are taken from the libKGE library which supports various KGE models on a unified codebase [11].

In the middle part, we report various other models from different classes or hybrid classes. In particular we report results for the neuro-symbolic approaches DRUM [45], Neural LP [61], KBLRN [18], and RLvLR [37]. Furthermore, we include the interpretable approaches GPFL [19], the predecessor of AnyBURL, RuleN [32], and SAFRAN [40]. Finally, we report results for the recently proposed GNN architecture A*Net [63].

6.1.2 Results

Table 6 shows the results. We first have a look at the deterministic aggregation functions and also compare them with the KGE models, which are better on average than the other model classes except for A*Net. Noisy-or top-5 is slightly superior compared to Max+ with 1.7 percentage points being the strongest benefit on Fb15k-237 but only minor or no improvement on the other datasets. Noisy-or performs inferior overall. Max+ is inferior on Fb15k-237 compared to all KGE models except of TransE. It is better on WN18RR than all KGE models and only better than TransE on Codex-M. Noisy-Or Top-5 closes the gap on Fb15k-237 compared to the KGE models and ranks in the middle.

We will now compare the deterministic aggregation functions with the learnable formulations. First of all, we can observe that learning the aggregation functions provides significant benefits over the deterministic functions for Fb15k-237 and Codex-M. The best performing specification (LR+) performs 3.1 percentage points better on Fb15k-237 and 3.2 percentage points in regard to the MRR on Codex-M when compared to Max+. On WN18RR, on the other hand, we only see minor improvements with *Dense* being the best performing specification with one percentage point improvement over Max+. However, on this dataset, the Max+ strategy is already relatively strong and outperforms the embedding models clearly.

The learnable aggregation approaches also perform better than the KGE models in general. The best performing model from the family of KGE models is ComplEx on Fb15k-237 with an MRR of 0.348 while LR+ achieves 0.365. Likewise on Codex-M, ComplEx achieves 0.337 while the simple learned aggregation functions all achieve better results. In general, the best results are given by A*Net which achieves an MRR of 0.411 for Fb15k-237 and 0.549 for WN18RR.

Table 6 Filtered MRR, Hits@1, Hits@10 results for Fb15k-237, WN18RR and Codex-M.

		Fb15k-237			WN18RR			Codex-M		
	Approach	MRR	Hits@1	Hits@10	MRR	Hits@1	Hits@10	MRR	Hits@1	Hits@10
KGE	RESCAL	.357	.263	.541	.468	.439	.521	.317	.244	.456
	TransE	.313	.221	.497	.227	.053	.526	.303	.223	.454
	DistMult	.343	.250	.531	.452	.413	.531	-	-	-
	ComplEx	.348	.253	.534	.477	.438	.543	.337	.262	.476
	ConvE	.339	.248	.521	.447	.411	.508	.318	.239	.464
	RotatE	.336	.238	.531	.475	.426	.574	-	-	-
	TuckER	.352	.259	.536	.459	.430	.514	.328	.259	.458
Other	DRUM	.343	.255	.516	.486	.425	.586	-	-	-
	Neural LP	.240	-	.362	.435	.371	.566	-	-	-
	KBLRN	.306	0.220	.482	-	-	-	-	-	-
	RLvLR	.240	-	.393	-	-	-	-	-	-
	GPFL	.322	.247	.504	.480	.449	.552	-	-	-
	RuleN	-	.182	.420	-	.427	.536	-	-	-
	SAFRAN	.351	.269	.513	.502	.459	.578	.320	.253	.449
	A* Net	.411	.321	.586	.549	.495	.659	-	-	-
Aggregation	Max+	.331	.246	.506	.497	.457	.572	.316	.247	.450
	NO	.329	.247	.494	.446	.391	.559	.289	.218	.427
	NO top-5	.348	.261	.524	.498	.458	.575	.320	.244	.468
	Sparse	.352	.266	.526	.499	.459	.574	.335	.266	.467
	Dense	.335	.245	.510	.507	.466	.587	.331	.261	.465
	D+S	.354	.267	.527	.511	.469	.593	.342	.273	.476
	LR	.362	.275	.532	.500	.457	.581	.339	.268	.477
	LR+	.365	.279	.538	.500	.458	.575	.348	.278	.477
	NO (learned)	.357	.268	.535	.496	.453	.575	.346	.277	.478

6.2 Large Graphs

Within this section we focus on the large-scale applicability of AnyBURL. It has sometimes been argued that symbolic approaches, in contrast to latent approaches, cannot be applied to very large datasets. Within this section we argue that the opposite is the case. Rule-based approaches can be extremly efficient and thus it is possible to outperform latent approaches on very large datasets in both runtimes and predictive quality. Throughout the following experiment we use the default AnyBURL settings. We use the Max+ aggregation to generate the predictions, which is the most efficient and robust aggregation method. For all datasets, with the exception of Freebase, we keep all rules with a support of at least two, which is AnyBURL default setting. For Freebase, which is by far the largest dataset, we increase this parameter from two to five to avoid rule sets which might become too large. Further details related to the experimental setup can be found in [30].

6.2.1 Models and Datasets

In [23], the authors used two standard knowledge graph embedding models (ComplEx [55] and RotateE [51]) 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 [23]) we have chosen the approaches that resulted in the best MRR (below marked with b) and the approach that turned out to be the fastest (below marked with f).

While [23] 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 [47], an ensemble combining an encoder-decoder transformer model and ComplEx. For the Yago03-10 dataset, which has also been used in the meta study [44] where sixteen different methods have been evaluated, we pick additionally the best and the second best result. Finally, for Freebase we include the results of *GRASH* [24] 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 [26] in our experiments. AMIE 3 is the latest version of the rule learner AMIE [17]. 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 8) and increase the rule length from two to three (referred to as $l \leq 3$ in Table 8). We do not report about some initial experiments where we also activated rules with constants, as AMIE did not terminate within 24 hours for all three datasets that we use in the large-scale setting. 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.

Table 7 Large Datasets. The first two columns show the number of entities and relations, respectively. The third, fourth and fifth row show the number of facts in the training, validation, and test KG. M means million.

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{P}\rvert$	#Train	#Valid	#Test
Yago03-10	123 000	37	1.079M	5000	5000
Wikidata5M	4594	822	20.625M	5357	5321
Freebase	86.054M	15000	338.586M	16.929M	10 000

For our large-scale experiments, we use the three largest datasets that have also been used in [23]: Wikidata5M (WD5M), Yago03-10, and Freebase (FB). The dataset Yago03-10 is described in [28] and has first been used in the context of knowledge graph completion in [14]. It is the subset of YAGO3 that consists of entities which are described by at least 10 triples. WD5M [58] 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 [23, 27]. To make our results comparable to the results presented in [23], we use the same subset of the test set for the evaluation. The characteristics of these datasets are described in Table 7. 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 [23]. They are thus well suited for assessing if our rule-based approach can deal with challenging large-scale settings.

Table 8 Rule-based results compared to the models and parallelization techniques analyzed in [23]. 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. All metrics are filtered.

Dataset	Model	Runtimes					Rules	Predictive Quality		
		Index	Learn	Predict	HS	Σ		Hits@1	Hits@10	MRR
Yago03-10	AnyBURL [30]	8s	100s	186s	-	4.9m	738k	0.495	0.683	0.561
		8s	1000s	221s	-	20.5m	4079k	0.497	0.689	0.565
	AMIE, default	7s	23s	1s	-	0.5m	224	0.303	0.517	0.375
	AMIE, $s \geq 2$	7s	38s	1s	-	0.8m	519	0.305	0.519	0.377
	AMIE, $l \leq 3$	7s	4440s	2s	-	74m	2561	0.327	0.557	0.405
	ComplEx (b)		8627s		486s - 14580s	151.9m - 386.8m			0.675	0.542
	ComplEx (f)		7600s		380s - 11400s	133m - 316.7m			0.669	0.538
	RotatE (b)		29640s		1482s - 44460s	518.7m - 1235m			0.637	0.451
	RotatE (f)		13260s		816s - 24480s	234.6m - 629m			0.607	0.438
	1st in [44], ComplEx					*>20h*		**0.505**	**0.704**	**0.576**
	2nd in [44], TuckER					*>10h*		0.466	0.681	0.544
Wikidata5M	AnyBURL [30]	122s	100s	1285s	-	25.1m	85k	0.275	0.378	0.310
		122s	1000s	2591s	-	61.8m	642k	0.300	0.413	0.338
		122s	10000s	7009s	-	285.5m	4294k	**0.312**	**0.433**	**0.353**
	AMIE, default	104s	1443s	4s	-	25.8m	3.7k	0.178	0.204	0.188
	AMIE, $s \geq 2$	104s	1942s	6s	-	34.2m	27.5k	0.190	0.221	0.202
	AMIE, $l \leq 3$	104s	>24h	-	-	>24h	-	-	-	-
	ComplEx (b&f)		63840s		4560s - 136800s	0.8 days - 2.3 days			0.398	0.308
	RotatE (b&f)		35003s		9334s - 280020s	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	AnyBURL [30]	1150s	100s	264s	-	25.2m	113k	0.690	0.714	0.699
		1150s	1000s	412s	-	42.7m	505k	0.698	0.725	0.707
		1150s	10000s	971s	-	202m	3245k	**0.702**	**0.728**	**0.711**
	AMIE, default	886s	>24h	-	-	>24h	-	-	-	-
	ComplEx (b)		7046s		118s - 3540s	119.4m - 176.4m			0.529	0.426
	ComplEx (f)		5916s		26s - 780s	99.0m - 111.6m			0.523	0.421
	RotatE (b)		64957s		92s - 2760s	1084.15m - 1128.6m			0.627	0.566
	RotatE (f)		9383s		92s - 2760s	157.9m - 202.4m			0.621	0.562
	ComplEx [24]				32274s	>537.9m				0.678

6.2.2 Results

We now discuss the results shown in Table 8. As explained above, the entries for ComplEx and RotatE marked with (b) and (f) correspond to the best and fastest configuration in [23]. The first and second best models from [44] are added as well as the current state-of-the-art on WD5M [47] and Freebase [23]. 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 [23] for each of the three datasets when we look at the results based on the rule sets that have been learned after 100 seconds. If we increase the learning time to 10000 seconds, 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 [47].[4] Moreover, AnyBURL outperforms the previous best results [24] on Freebase with respect to MRR and Hits@10.

Our results for Yago03-10 outperform ComplEx and RotatE results when using different parallelization techniques [23], however, several publications reported slightly better results. While we achieve a score of 0.565, in [25] the authors report an MRR of 0.58 and in [44] an MRR of 0.576. Runtimes have not been stated in [25], even though it's known that the authors use a very large embedding dimension which may lead to high runtimes. The results reported in [44] require a training time (not including the hyperparameter search) of more than 20 hours for Yago03-10 while AnyBURL require only slightly more than 20 minutes. These runtime comparisons might indicate what effort is required to achieve these MRR scores with an embedding approach. 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 [44] 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 extensive 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 hours. 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 [24].

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 hours on these datasets when activating constants. Whenever AMIE generated a result within 24 hours it performed clearly worse than the results obtained by the 100 second learning time run of AnyBURL.

[4] There are some works that use instead a sampled MRR [62, 27], where the correct entity is not ranked against all other entities but against a relatively small sampled subset. We refer to the considerations in [23], 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 [23], which are again clearly outperformed by our version of AnyBURL.

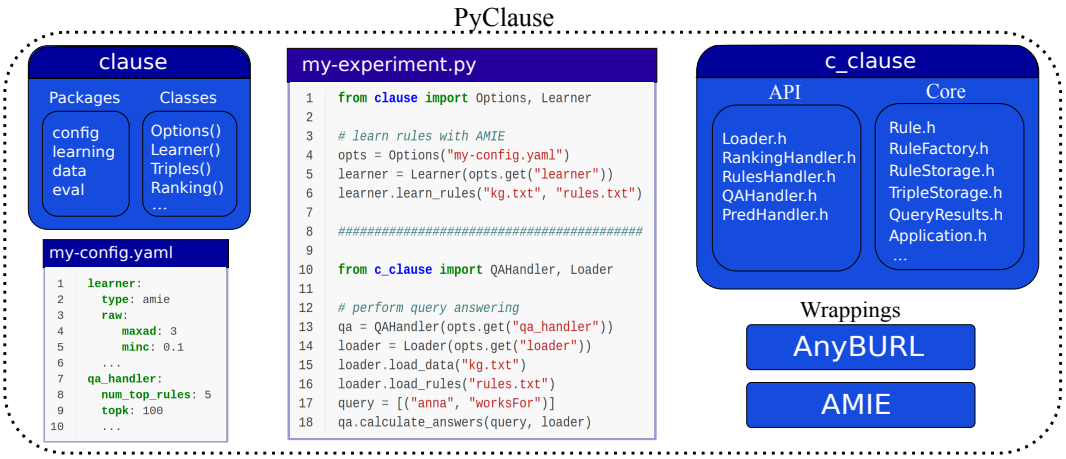

Figure 7 Usage-centric library overview. The example code mines rules with AMIE, which are used to predict the employer of a person.

7 Using and Learning Rules with the PyClause Library

We will now turn to the more practical aspects and introduce the Python-based PyClause library in the first part of this section [5]. Subsequently, we show an application example based on the biomdedical KG Hetionet [20], executed with PyClause. The development of PyClause was motivated by the lack of implementations for general rule application functionality in the context of KGs. One the one hand, there exist many different rule learning approaches and implementations apart from AnyBURL, such as AMIE [17, 16, 26], RuDiK [38], Aleph [49], TyRule [59], and GPFL [19]. On the other hand, when considering the implementations of the respective approaches, the functionality to actually use the rules is most often hardly accessible. To achieve different types of rule application functionalities, users would have to invest a substantial amount of additional coding effort.

PyClause makes general rule application on substatial sized KGs easily accessible in Python. Applications using the rules can be incorporated into existing Python projects with a few lines of code. PyClause can flexibly switch between different input and output modes, e.g., from Numpy arrays containing integers to lists of strings. Computations can be performed by user defined options and all implementation details are documented in a default configuration file. PyClause supports multithreading for all the implemented features and its core mechanics are implemented in C++ for maintaining runtime efficiency. Finally, PyClause supports rule learning based on AnyBURL or AMIE with Python. The library is publicly available at `https://github.com/symbolic-kg/PyClause`.

7.1 Introduction to PyClause

PyClause is structured in two Python packages `clause` and `c_clause`. Figure 7 shows the library components when using different features. The `c_clause` package is written in C++, and implements most of the library core features such as data storing and rule grounding. It is accessible from within Python via a high-level API that is structured into multiple handler classes providing access to different functionality. The `clause` package is pure Python and provides different utilities such as option handling and rule learning. PyClause supports all the rule types that are discussed in this work. An overview about the detailed syntax and examples are provided in the library documentation.

PyClause provides convenience wrappers to the rule learners AMIE and AnyBURL. Both systems can be run from Python, and can be fully configured by the PyClause option handling. The AMIE system was modified to conform to a common interface regarding rule types and syntax. Figure 7 (center top) shows how a rule mining system can be called easily from within PyClause. In the following, we assume that the rules have been learned by any system and focus on the application functionalities.

A typical workflow can be seen in Figure 7 (center bottom): we first load a KG and a rule set into memory by help of the `c_clause.Loader`. It is given as a reference to a particular handler class, that additionally takes feature specific input (e.g., facts) to perform its task. Then, PyClause supports the following operations on the KG and set of rules.

PREDICT. Three different types of making factual predictions are provided. 1) Given a KG and one or more input queries (head or tail), propose candidate answers for the missing entity of the query and provide a likelihood score for each candidate by using confidence aggregation. 2) Given a set of complete facts, calculate the confidence aggregation score for each fact, which will be zero if it is not predicted by any rule. Different deterministic aggregation functions can be configured in the options (as defined in Section 5.2). 3) Given an additional test KG, PyClause can directly compute all the rankings for all head and tail queries formed from all facts of the test KG (defined in Section 2.2.1). PyClause also provides utilities to calculate the ranking-based evaluation metrics defined in this work.

EXPLAIN. For the prediction modes described above, the library can additionally compute and output all the rules that made the respective candidate or fact predictions. These constitute the reasons why a prediction was made and they could, e.g., be used as features for neuro-symbolic models. Finally, when input facts are given for scoring them, their predicting rules and additionally all body groundings of each rule can be calculated. For example, an explanation for the prediction *citizen(Tom Hanks, US)* can be given by the rule $citizen(X, Y) \leftarrow livesIn(X, A), locatedIn(A, Y)$ together with its body groundings $\{livesIn(Tom\ Hanks, LA), locatedIn(LA, US)\}$.

SCORE. Given a set of rules, PyClause can compute different metrics such as the number of predictions, the support, and the rule confidence, on any specified KG.

MATERIALIZE. Given a set of rules, PyClause can find all predictions of the rule, and materialize them into a new set of facts. These facts can, for instance, be added to the original KG.

For all of these functionalities, PyClause supports different types of data representation. Inputs and outputs can directly be read or written from and to files or they can be hold in memory and be processed further with Python. Facts and rules can be either represented as Numpy arrays containing indices or as lists of strings. The full list of PyClause operators as well as further details are provided in the library documentation.

7.2 Application Example

We will now show some features of PyClause via a use case in the biomedical domain based on the KG Hetionet [20] which contains more than two million facts. The KG describes relations bwetween drugs (compounds), diseases, genes, and anatomic structures such as nose or brain. There are roughly 700 facts for the target relation *Compound-treats-Disease*. The

example is created in conjunction with a domain expert. We will only show code snippets and assume below that a `Loader` object (loader) is instantiated and has loaded the KG and a respective rule set. We additionally assume the handler objects are instantiated and configured, e.g., `QAHandler` (qa), `RulesHandler` (rh) and `PredictionHandler` (ph). [5]

7.2.1 Pattern Analysis

Our user Phia is provided with the Hetionet KG. She wants to learn something about the patterns of the KG and is interested in new use cases, i.e., diseases, for given compounds. In terms of the KG this can be represented by asking tail queries in regard to the target relation *Compound-treats-Disease*. The user learns a rule set by using and configuring AnyBURL from within PyClause (see, e.g., Figure 7).

She first wants to use PyClause to achieve some overview over the KG. She decides to only load rules with length one and with a high confidence. The rules are outputted to Python and sorted. Some interesting examples are:

```
1.0  se(X,Fever) <= se(X,Enteritis)
0.85 expresses(X,Y) <= upregulates(X,Y)
```

Here *se* is short for *causes side effect*. The first rule has a confidence of 1.0 and it says that a drug causing the side effect *Enteritis* also causes a high body temperature. Enteritis is an acute inflammatory process which is known to increase body temperature. The second rule has a confidence of 0.85 and says that if an anatomic structure, e.g, the kidney, is involved in the upregulation of a gene it may also be responsible for its expression. Next, she wants to test for some simple regularities within the graph. In fact, the description of the relation *Compound-resembles-Compound* suggests it is symmetric. To test this, she defines a rule expressing symmetry for the relation and calculates the support and the number of predictions of the rule.

```
rules = ["resembles(X,Y) <= resembles(Y,X)"]
rh.calculate_predictions(rules, loader)
rh.get_statistics()
[[6486, 0]]
```

The number of predictions is 6486 but the support is zero. The dataset seemingly does not respect symmetry for the relation. Phia could now, for instance, obtain the materialization of the rule and augment the initial KG.

7.2.2 Query Answering

In the drug repurposing problem, the task is to propose additional ailments that can be treated by a given drug. Phia defines the query *treats(Isoetarine, ?)* where *treats* describes the target relation *Compound-treats-Disease*. She calculates a list of candidates with the *QAHandler*, given the KG and the rules. Additionally, for each of the candidates, the relevant rules are accessed. The candidates are assigned with a score based on the selected aggregation function. The candidate with the highest score is `Asthma`:

[5] The full example can be found at `https://github.com/symbolic-kg/PyClause/blob/master/examples/hetionet-demo.ipynb`

```
queries = [("Isoetarine", "treats")]
qa.calculate_answers(queries, loader, "tail")
answers = qa.get_answers(as_string=True)
pred_rules = qa.get_rules(as_string=True)
answers[0][0] # return first candidate
'asthma'
```

The rules that predict `Asthma` for the query are stored in the variable *pred_rules* above, two examples are shown below:

```
treats(X,asthma) <= binds(X,ADRB2)
treats(X,Y) <= resembles(A,X), treats(A,Y)
```

The first rules says that `Asthma` might be treated with a compound that binds the protein encoded by the gene ADRB2. The respective protein is a receptor, which is also present in the bronchi and can lead to their relaxation. The second rule simply says that a compound might treat an ailment if it resembles a drug that is already known to treat the ailment.

The user is curious about why the second rule predicted `Asthma`. She uses the prediction handler to output the relevant body groundings of the rule.

```
triple = [("Isoetarine", "treats", "asthma")]
ph.calculate_scores(triple, loader)
targets, pred_rules, groundings \
  = ph.get_explanations(as_string=True)
```

The variable `groundings` contains the body groundings of the relevant rules. In this case there exist three body groundings for the second rule. We show the output when accessing the first one.

```
['Salbutamol', 'resembles', 'Isoetarine']
['Salbutamol', 'treats', 'asthma']
```

We can thus see that one reason for the prediction `Asthma` is that the resembling drug `Salbutamol` is already known to treat `Asthma`.

8 Conclusion

In this work, we covered all the fundamental aspects of building a rule-based approach for KGC. We discussed how rules can be learned by sampling and generalizing paths from a KG. When rules are learned from a KG, they allow to uncover the underlying patterns present in the graph. Ultimately, however, a user might want to use the learned rules to make new fact predictions. This also is required in the context of KGC. Here, rules need to make fact predictions for performing query answering. We showed how the rules can be used efficiently for this task and which problems arise. When multiple rules make the same prediction, a decision needs to be made of how to aggregate the respective rule confidences. Alternatively, an aggregation function can be learned in a supervised setting. We also contrasted this type of rule inference from using full logical entailment. We then showed that rule-based approaches are not only interpretable but also competitive in regard to predictive performance when compared to embedding-based models. Moreover, they are shown to scale well to very large KGs. We introduced PyClause, which is a Python-based library for general rule application functionality. It also allows to learn rules directly from Python while using the rule learners AMIE and AnyBURL. Finally, we showed a practical example, executed with PyClause, on the biomedical KG Hetionet.

References

1 Mona Alshahrani, Mohammad Asif Khan, Omar Maddouri, Akira R Kinjo, Núria Queralt-Rosinach, and Robert Hoehndorf. Neuro-symbolic representation learning on biological knowledge graphs. *Bioinformatics*, 33(17):2723–2730, 2017. doi:10.1093/BIOINFORMATICS/BTX275.

2 Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007. doi:10.1007/978-3-540-76298-0_52.

3 Ivana Balazevic, Carl Allen, and Timothy Hospedales. 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 (EMNLP-IJCNLP)*, pages 5185–5194, 2019.

4 Patrick Betz. *Symbolic Rule-Based Knowledge Graph Completion*. Universitaet Mannheim (Germany), 2025.

5 Patrick Betz, Luis Galarraga, Simon Ott, Christian Meilicke, Fabian M Suchanek, and Heiner Stuckenschmidt. Pyclause-simple and efficient rule handling for knowledge graphs. In *IJCAI, demo track*. Ijcai.org, 2024. URL: https://www.ijcai.org/proceedings/2024/991.

6 Patrick Betz, Stefan Lüdtke, Meilicke Christian, and Stuckenschmidt Heiner. Rule confidence aggregation for knowledge graph completion. In *International Joint Conference on Rules and Reasoning*. Springer, 2024.

7 Patrick Betz, Christian Meilicke, and Heiner Stuckenschmidt. Adversarial explanations for knowledge graph embedding models. In *Proceedings of the 31th International Joint Conference on Artificial Intelligence*, pages 2820–2826. Ijcai.org, 2022.

8 Patrick Betz, Christian Meilicke, and Heiner Stuckenschmidt. Supervised knowledge aggregation for knowledge graph completion. In *Extended Semantic Web Conference*, pages 74–92. Springer, 2022. doi:10.1007/978-3-031-06981-9_5.

9 Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008. doi:10.1145/1376616.1376746.

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

11 Samuel Broscheit, Daniel Ruffinelli, Adrian Kochsiek, Patrick Betz, and Rainer Gemulla. Libkge-a knowledge graph embedding library for reproducible research. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 165–174, 2020. doi:10.18653/V1/2020.EMNLP-DEMOS.22.

12 Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 2462–2467. ijcai.org, 2007. URL: http://ijcai.org/Proceedings/07/Papers/396.pdf.

13 Luc Dehaspe and Hannu Toivonen. Discovery of relational association rules. In *Relational data mining*, pages 189–212. Springer, 2001.

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

15 Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1811–1818, 2018. doi:10.1609/AAAI.V32I1.11573.

16 Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. Fast rule mining in ontological knowledge bases with AMIE+. *The VLDB Journal*, 24(6):707–730, 2015. doi:10.1007/S00778-015-0394-1.

17 Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*, pages 413–422. International World Wide Web Conferences Steering Committee, 2013. doi:10.1145/2488388.2488425.

18 Alberto García-Durán and Mathias Niepert. Kblrn: End-to-end learning of knowledge base representations with latent, relational, and numerical features. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 372–381. AUAI Press, 2018. URL: http://auai.org/uai2018/proceedings/papers/149.pdf.

19 Yulong Gu, Yu Guan, and Paolo Missier. Towards learning instantiated logical rules from knowledge graphs, 2020. arXiv:2003.06071.

20 Daniel Scott Himmelstein, Antoine Lizee, Christine Hessler, Leo Brueggeman, Sabrina L Chen, Dexter Hadley, Ari Green, Pouya Khankhanian, and Sergio E Baranzini. Systematic integration of biomedical knowledge prioritizes drugs for repurposing. *Elife*, 6, 2017.

21 Harry Joe. *Multivariate models and multivariate dependence concepts*. CRC press, 1997.

22 Michael N Katehakis and Arthur F Veinott Jr. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268, 1987. doi:10.1287/MOOR.12.2.262.

23 Adrian Kochsiek and Rainer Gemulla. Parallel training of knowledge graph embedding models: a comparison of techniques. *Proceedings of the VLDB Endowment*, 15(3):633–645, 2021. doi:10.14778/3494124.3494144.

24 Adrian Kochsiek, Fritz Niesel, and Rainer Gemulla. Start small, think big: On hyperparameter optimization for large-scale knowledge graph embeddings. In *Accepted at the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2022.

25 Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In *ICML*, pages 2869–2878, 2018. URL: http://proceedings.mlr.press/v80/lacroix18a.html.

26 Jonathan Lajus, Luis Galárraga, and Fabian Suchanek. Fast and exact rule mining with amie 3. In *Proceedings of the Extended Semantic Web Conference*, pages 36–52. Springer, 2020. doi:10.1007/978-3-030-49461-2_3.

27 Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large scale graph embedding system. *Proceedings of Machine Learning and Systems*, 1:120–131, 2019.

28 Farzaneh Mahdisoltani, Joanna Biega, and Fabian M Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *Proceedings of CIDR 2015*, 2015.

29 Christian Meilicke, Patrick Betz, and Heiner Stuckenschmidt. Why a naive way to combine symbolic and latent knowledge base completion works surprisingly well. In *3rd Conference on Automated Knowledge Base Construction*, 2021.

30 Christian Meilicke, Melisachew Wudage Chekol, Patrick Betz, Manuel Fink, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for large-scale knowledge graph completion. *The VLDB Journal*, 33(1):131–161, 2024. doi:10.1007/S00778-023-00800-5.

31 Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 3137–3143. International Joint Conferences on Artificial Intelligence Organization, 2019. doi:10.24963/IJCAI.2019/435.

32 Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. Fine-grained evaluation of rule- and embedding-based systems for knowledge graph completion. In *Proceedings of the International Semantic Web Conference*, pages 3–20. Springer, 2018. doi:10.1007/978-3-030-00671-6_1.

33 Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion. In *International Semantic Web Conference*, pages 3–20. Springer, 2018. doi:10.1007/978-3-030-00671-6_1.

34 Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994. doi:10.1016/0743-1066(94)90035-3.

35 Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015. doi:10.1109/JPROC.2015.2483592.

36 Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, volume 11, pages 809–816, 2011. URL: https://icml.cc/2011/papers/438_icmlpaper.pdf.

37 Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. Scalable rule learning via learning representation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2149–2155. International Joint Conferences on Artificial Intelligence Organization, July 2018. doi:10.24963/ijcai.2018/297.

38 Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. Robust discovery of positive and negative rules in knowledge bases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1168–1179. IEEE, 2018. doi:10.1109/ICDE.2018.00108.

39 Simon Ott, Patrick Betz, Daria Stepanova, Mohamed H Gad-Elrab, Christian Meilicke, and Heiner Stuckenschmidt. Rule-based knowledge graph completion with canonical models. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 1971–1981, 2023. doi:10.1145/3583780.3615042.

40 Simon Ott, Christian Meilicke, and Matthias Samwald. SAFRAN: An interpretable, rule-based link prediction method outperforming embedding models. In *3rd Conference on Automated Knowledge Base Construction*, 2021.

41 Stefano Pallottino. Shortest-path methods: Complexity, interrelations and new propositions. *Networks*, 14(2):257–267, 1984. doi:10.1002/NET.3230140206.

42 Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

43 Thomas Pellissier Tanon, Gerhard Weikum, and Fabian Suchanek. Yago 4: A reason-able knowledge base. In *The Semantic Web: 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31–June 4, 2020, Proceedings 17*, pages 583–596. Springer, 2020. doi:10.1007/978-3-030-49461-2_34.

44 Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(2):1–49, 2021. doi:10.1145/3424672.

45 Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. Drum: End-to-end differentiable rule mining on knowledge graphs. In *Advances in Neural Information Processing Systems*, pages 15321–15331, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/0c72cb7ee1512f800abe27823a792d03-Abstract.html.

46 Tara Safavi and Danai Koutra. CoDEx: A Comprehensive Knowledge Graph Completion Benchmark. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 8328–8350. Association for Computational Linguistics, 2020. doi:10.18653/V1/2020.EMNLP-MAIN.669.

47 Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. Sequence-to-sequence knowledge graph completion and question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2814–2828, 2022. doi:10.18653/V1/2022.ACL-LONG.201.

48 Giovanni Semeraro, Floriana Esposito, Donato Malerba, Clifford Brunk, and Michael Pazzani. Avoiding non-termination when learning logic programs: A case study with foil and focl. In *Logic Program Synthesis and Transformation—Meta-Programming in Logic*, pages 183–198. Springer, 1994. doi:10.1007/3-540-58792-6_12.

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

50 Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1242572.1242667.

51 Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *Proceedings of the ICLR 2019*, 2019.

52 Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2 edition, 2018.

53 Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. Completeness-aware rule learning from knowledge graphs. In *International Joint Conference on Artificial Intelligence*, pages 507–525. Ijcai.org, 2017. doi:10.1007/978-3-319-68288-4_30.

54 Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, Beijing, China, July 2015. Association for Computational Linguistics. doi:10.18653/V1/W15-4007.

55 Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pages 2071–2080, 2016. URL: http://proceedings.mlr.press/v48/trouillon16.html.

56 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing system*, 30, 2017.

57 Denny Vrandecic and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *CACM*, 57(10):78–85, 2014. doi:10.1145/2629489.

58 Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. Kepler: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194, 2021. doi:10.1162/TACL_A_00360.

59 Hong Wu, Zhe Wang, Kewen Wang, and Yi-Dong Shen. Learning typed rules over knowledge graphs. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, pages 494–503, 2022.

60 Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

61 Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems*, pages 2319–2328, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/0e55666a4ad822e0e34299df3591d979-Abstract.html.

62 Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. DGL-KE: Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 739–748, 2020. doi:10.1145/3397271.3401172.

63 Zhaocheng Zhu, Xinyu Yuan, Michael Galkin, Louis-Pascal Xhonneux, Ming Zhang, Maxime Gazeau, and Jian Tang. A* net: A scalable path-based reasoning approach for knowledge graphs. *Advances in Neural Information Processing Systems*, 36, 2024.