RDFRules: Making RDF Rule Mining Easier and Even More Efficient

Václav Zeman [*], Tomáš Kliegr and Vojtěch Svátek
Department of Information and Knowledge Engineering, Faculty of Informatics and Statistics, Prague University of Economics and Business, nám W Churchilla 4, 13067 Czech Republic
E-mail: first.last@vse.cz

Editor: Agnieszka Lawrynowicz, Poznan University of Technology, Poland
Solicited reviews: Jedrzej Potoniec, Poznan University of Technology, Poland; Emir Muñoz, Genesys, US; Two anonymous reviewers

Abstract. AMIE+ is a state-of-the-art algorithm for learning rules from RDF knowledge graphs (KGs). Based on association rule learning, AMIE+ constituted a breakthrough in terms of speed on large data compared to the previous generation of ILP-based systems. In this paper we present several algorithmic extensions to AMIE+, which make it faster, and the support for data pre-processing and model post-processing, which provides a more comprehensive coverage of the linked data mining process than does the original AMIE+ implementation. The main contributions are related to performance improvement: (1) the top-*k* approach, which addresses the problem of combinatorial explosion often resulting from a hand-set minimum support threshold, (2) a grammar that allows to define fine-grained patterns reducing the size of the search space, and (3) a faster projection binding reducing the number of repetitive calculations. Other enhancements include the possibility to mine across multiple graphs, the support for discretization of continuous values, and the selection of the most representative rules using proven rule pruning and clustering algorithms. Benchmarks show reductions in mining time of up to several orders of magnitude compared to AMIE+. An open-source implementation is available under the name RDFRules at https://github.com/propi/rdfrules.

Keywords: Rule Mining, Rule Learning, Exploratory data analysis, Machine Learning, Inductive Logical Programming

1. Introduction

Finding interesting interpretable patterns in data is a frequently performed task in data science workflows. Software for finding association rules, as a specific form of patterns, is present in nearly all data mining software bundles. These implementations are based on the *Apriori* algorithm [1] or its successors. While very fast, these algorithms are severely constrained with respect to the shape of analyzed data – only single tables or transactional data are accepted. Algorithms for logical rule mining developed within the scope of Inductive Logical Programming (ILP) do not have these restrictions, but they typically require negative examples and do not scale to larger knowledge graphs (KGs) [2].

Commonly used open KGs, such as Wikidata [3], DBpedia [4], and YAGO [5], are published in RDF [6] as sets of triples – statements in the form of binary relationships. Most of these RDF KGs[1] operate under the *Open World Assumption* (OWA). It means that the KG is regarded as inherently incomplete and is open for adding more statements that are currently missing. For example, if the description of a person does not contain any information about employment, it is not correct to infer that the person is unemployed; however, a straightforward application of ILP approaches to rule learning requires such inferences to be made to generate negative examples. Hence, it is not appropriate to use standard ILP tools for mining rules from RDF KGs

[*]Corresponding author.

[1]Also "RDF Knowledge Graph", which is a term used in related research (e.g., [7, 8]) to denote a KG formed as a collection of statements in the RDF format.

due to their reliance on the *Closed World Assumption* (CWA) in input data.

The current state-of-the-art approach for rule mining from RDF KGs is AMIE+ [2]. AMIE+ combines the main principles that make association rule learning fast with the expressivity of ILP systems. AMIE+ mines Horn rules, which have the form of implication and consist of one atomic formula (or simply *atom*) on the right side, called head, and conjunction of atoms on the left side, called body:

$$B_1 \wedge B_2 \wedge \ldots \wedge B_n \Rightarrow H$$

The restricted variant of atom relevant for RDF KG mining has the form of a triple. For instance,[2]

$$(\text{?a <hasChild> ?c}) \wedge (\text{?b <hasChild> ?c}) \Rightarrow (\text{?a <isMarriedTo> ?b}).$$

In this example, each atom consists of a predicate and two variables at its subject and object positions. A variable can also be instantiated to a specific constant, e.g.,

$$(\text{?a <hasChild> <Carl>}).$$

AMIE+ uses its own strategy to evaluate the quality of mined rules with respect to the OWA and, therefore, it is also appropriate for mining rules from open KGs forming the Semantic Web.

In our view, AMIE+ constitutes a big leap forward in learning rules from KGs, similar in magnitude to what the invention of *Apriori* meant for rule learning from transactional data. However, AMIE+ also shares some notable limitations with the original *Apriori* algorithm. Decades of research in association rule learning and frequent itemset mining continuously show how difficult it is for users to constraint the search space so that meaningful rules are generated, and combinatorial explosion is avoided. In the presented work, we address these limitations by drawing inspiration from techniques proven in rule mining from transactional databases. The extensions to AMIE+ introduced in this paper include a top-*k* approach, which can circumvent the need for the user manually tuning the support threshold, fine-grained search space restrictions, avoidance of some repetitive calculations, and the ability to discretize values in numerical literals.

[2]Since all atoms are binary, we use an infix notation, which is more readable here than the prefix (first-order logic) notation used in ILP. We also distinguish variables with a question mark, and the Internationalized Resource Identifiers (IRIs) with angle brackets. See Sec. 3 for more details.

Together, these optimisations substantially reduce the large search space of potential rules that have to be checked. All the aforementioned approaches have been implemented within the RDFRules framework and evaluated in comparison with the original AMIE+ implementation. Additionally, this article describes two post-processing approaches – rule clustering and pruning – adopted for RDF KGs. These techniques are necessary to address the high number of rules that are typically on the output of rule mining .

The scope of functionality of RDFRules is inspired by the widely used *arules* framework [9] for rule learning from tabular data. Similarly to *arules*, RDFRules covers the complete data mining process, including data pre-processing (support for numerical attributes), various mining settings (fine-grained patterns and measures of significance), and post-processing of mining results (rule clustering, pruning, filtering and sorting).

We provide benchmarks demonstrating the benefits of the proposed performance enhancements. For example, for mining rules with constants – which are an essential element of association rules mined from tabular data – the AMIE+ algorithm can take hours (or days), but the presented approach has a more than an order of magnitude shorter mining time on YAGO and DBpedia. Similarly, the top-*k* approach can provide a more than ten times shorter mining time compared to the standard approach supported by AMIE+ when all rules conforming to the user-set minimum support threshold are first mined and then filtered. We also show that our implementation scales better than AMIE+ when additional CPU cores are added.

This paper is organised as follows. Section 2 provides a broader overview of related work. A digest of the AMIE+ approach is ranged as Section 3. Section 4 presents a list of limitations of AMIE+, providing the motivation for our work. The proposed approach is described in Section 5. Section 6 briefly describes its reference implementation. The results of the evaluation are presented in Section 7. The conclusion summarises the contribution and provides an outlook for future work.

A very limited work-in-progress version of this research was published in the proceedings of the 2018 RuleML Challenge workshop [10].

2. Related Work

Approaches applicable for rule learning from RDF KGs come principally from two domains. Mining

Horn rules from KGs has been for several decades studied within the scope of Inductive Logical Programming (ILP). A second area that inspired the development of recent algorithms, including AMIE+, is association rule learning, where the upward closure property [11] has been introduced to prune the space of hypotheses. At present, one of the main drivers for development of rule-based approaches is a potentially better explainability and traceability of the generated models compared to alternative approaches (e.g. graph embeddings).

Inductive Logical Programming Algorithms based on the principles of ILP learn Horn rules on binary predicates. Examples of applicable ILP approaches include ALEPH[3] and WARMeR [12].

Much progress related to ILP happened in the domain of mining from the Semantic Web. In [13], the authors have introduced the Semintec algorithm for mining patterns from ontologies. An example of a generated pattern is "Client(key), livesIn(key, x1), Region(x1); support = 1.0". As can be noticed, this system does not generate rules, but rather frequent patters, and thus uses *support* as the main measure. Support is also used in the extension of this algorithm called Fr-ONT [14], which was the first algorithm of this type to support variable-free notation of description logic. In terms of implementation, Fr-ONT uses the Pellet reasoner and Jena API[4], the latter is also used in our RDFRules framework. It should be noted that support in these algorithms is defined differently in AMIE+/RDFRules, in Fr-ONT support of a concept is calculated relatively to the number of instances of a user-specified reference concept. A subsequent version of Fr-ONT called Fr-ONT-Qu [15] can use an arbitrary interest (quality) measure that the patterns have to meet to be included in the output. This system was primarily developed for mining patterns represented as SPARQL queries. An extension of this approach called substitutive itemsets allows to discover pairs of items that are interchangeable as they appear in similar context [16]. Interestingly, both Fr-ONT-Qu and the substitutive itemset mining were made available as RapidMiner[5] extensions. One possible use case for this type of pattern mining is automatically suggesting *rdfs:subClassOf* relations, which can facilitate the work of ontology engineering [17].

[3]http://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph_toc.html
[4]https://jena.apache.org/
[5]https://rapidminer.com/

While ILP systems were primarily designed for learning from closed collections of ground facts, as has been demonstrated, e.g., in [2], they can also be used for rule learning from KGs. However, there are several challenges:

- ILP systems expect negative examples and are designed for the CWA.
- Logic-based reasoning approaches can multiply errors which are inherently present in most KGs [18]. For example, for KGs sourced from Wikipedia, they can arise due to parsing errors.
- ILP systems have been reported to be too slow to process real-world KGs such as YAGO. Galárraga et al. [2] benchmarked two state-of-the-art ILP systems (ALEPH and WARMeR) and found that these systems were under some settings unable to terminate within one day, while AMIE+ processed the same task within several seconds or minutes.
- ILP systems such as ALEPH may not generate all specializations of a given rule, while the association rule mining approach, such as AMIE+, would return all rules matching given thresholds [19].

It should be noted that some recent approaches based on ILP principles, such as STRiKE [20] or AnyBURL [19], ameliorate some of these problems (such as performance issues and error propagation). However, due to use of a heuristic to guide rule learning, these algorithms do not typically return a complete set of rules given the user set thresholds (such as minimum support and minimum confidence) as AMIE+ does.

Association Rule Mining Some algorithms for rule mining from RDF KGs use adaptations of the seminal *Apriori* algorithm [21] for discovering frequent itemsets or association rules from *transactions*. A transaction is a set of items typically related to a single contextual object, such as a shopping basket, and the whole transactional database contains objects of the same type. An example association rule is:

$$\{\mathsf{milk}, \mathsf{bread}\} \Rightarrow \{\mathsf{butter}\}.$$

This contrasts with ILP systems, where a logical rule may span across several contextual object types.

The AMIE+ algorithm uses the upward closure property used in the *Apriori* algorithm to reduce the search space by a minimum support threshold and other mechanisms for making rule mining faster than the mentioned ILP systems [2]. AMIE+ was recently used, e.g., for completing missing statements in KGs

[19] and also used for rule learning in SANSA-Stack [22], which is a general-purpose toolbox for distributed data processing of large-scale RDF KGs based on Apache Spark[6].

Another algorithm adapting association rule mining for RDF data is called SWARM [8]. In the Semantic Web we usually divide an RDF KG into two components: an A-Box containing instance triples and a T-Box defining a schema for them. The SWARM algorithm, proposed by Barati et al., mines *Semantic Association Rules* (as the authors call the algorithm's output) from both the A-Box and the T-Box. Compared with the AMIE+ algorithm, which only mines rules from the A-Box, SWARM generates so-called *Semantic Items*, forming a set of transactions which is used as input for association rule mining. Hence, SWARM does not mine typical ILP rules with variables, but only semantically-enriched association rules, such as:

$$\mathsf{Person} : (\mathsf{instrument}, \mathsf{Guitar}) \Rightarrow (\mathsf{occupation}, \mathsf{Songwriter}).$$

There are also other approaches that transform RDF data into transactions and mine typical association rules or itemsets, for example, in the specific contexts of ontology classes [23] or Wikipedia categories [24]. Nebot and Berlanga [25], in turn, proposed an extension of the SPARQL query language to generate transactions of a user-defined context and to mine association rules using the *Apriori* algorithm.

Combining ILP with Association Rule Mining A hybrid approach that involves both reasoning and association rule mining is presented in [26]. The intended application of this type of rule mining is suggesting new axioms to be added to the ontology [27]. Despite the advantages of the hybrid approach, the use of a reasoner in combination with association rule mining is likely to exceed what one can consider a "reasonable time" [26]. On the other hand, the advantage of approaches involving reasoning is that they generate rules, which are not necessarily closed, while AMIE+/RDFRules outputs only closed rules. Another advantage is that rules, which are not consistent with reference ontology can be pruned [27], which is not possible in AMIE+/RDFRules since these systems do not work with ontological knowledge. However, even not considering performance issues, the ability to apply reasoning may not always be an advantage [18]. In this research, the authors report on an experiment with type inference on real-world RDF datasets, using logical entailment rules. They report that if the KB has error rate of only 0.0005 (99.9% correctness), the reasoning approach would induce too many incorrect types. In contrast, a statistical analysis of types performed by the proposed SDType algorithm has accuracy over 90% [18]. The AMIE+/RDFRules approach also does not use formal semantics embedded in an ontology (e.g., cardinality restrictions) and instead used information only from RDF triples.

Graph Embeddings The main limitation of the previously mentioned approaches is the need to store the entire KG in the memory to allow for fast exploration of the search space. This may be a problem for large KGs since they have high resource requirements, and the existing systems are not able to effectively scale input data and the mining process. Graph embedding methods, e.g. RESCAL [28], HolE [29] and TransE [30], transform a KG or its individual components (nodes and edges) into vectors. With this representation, fast and easily scalable vector operations can be performed, and the number of vector dimensions can be kept under control.

The RLvLR algorithm [31] uses low-dimension embeddings of RDF KG resources and predicates for fast search of rules. This approach is even faster than the state-of-the-art AMIE+ algorithm but is focused only on learning rules for a specific predicate and cannot discover rules with constants.

Another rule mining system using embeddings is RuLES[7] [32]. It mines the same kind of rules as AMIE+ (with or without constants) and requires an embedding model pre-trained by TransE, HolE or SSP [33].

While learning rules from embeddings has certain advantages, it is also known to have multiple weaknesses. One is its poor capability of encoding sparse entities [34]. Another problem is that the results of learning embeddings are highly susceptible to the choice of dimensionality, typically requiring the time-consuming process of training the embeddings with different dimensionality and evaluating them in functional tests [34, 35]. Irrespective of the choice of dimensionality, the set of rules extracted from embeddings would not exactly match the exhaustive set of rules that is valid in the input knowledge graph, which is extracted by AMIE+. A recent independent evaluation performed on the real-world task of graph com-

[6]https://spark.apache.org/

[7]https://github.com/hovinhthinh/RuLES

pletion indicated superior results of AMIE+ in terms of precision compared to multiple algorithms based on embeddings, including TransE, HolE, or RESCAL [36], on some completion tasks.

Accuracy–Interpretability Tradeoff As can be seen from the review presented above, KG completion – as a representative of a common application domain for relational learning – can be approached with many types of statistical learning methods. These do not include only rule-based algorithms, but also other approaches, like SDType [18], that exploit links in the graph in a statistical way, or even the latest generation of reasoning algorithms. For example, the probabilistic case-based reasoning algorithm proposed by Das et al. [37] is highly competitive to rule-based approaches on the KG completion task, which were discussed above.

The higher accuracy of a learning algorithm is often associated with their increased sophistication [38, 39], which can make it more difficult to explain a particular prediction, or the learned model as a whole, to the human user. Rules are generally considered an intrinsically explainable representation [38, 40], although not without caveats [41]. In our opinion, the utility of "AMIE-style" rule learning approaches such as AMIE+ or RDFRules is in providing an appealing combination of predictive performance and explainability. In contrast to most other relational rule learning algorithms, AMIE+ outputs all rules matching the user-set constraints and the language bias of the algorithm (for instance, AMIE+ only generates closed rules). It is also relatively straightforward to explain why a particular rule was generated by AMIE+ and why another rule was not. However, the exhaustive approach to rule generation becomes a disadvantage in those learning scenarios where long rules are needed and it is at the same time impossible to apply other mining constraints or patterns limiting the search space. In such cases, other types of statistical learning or reasoning approaches may be more appropriate.

AMIE 3 It should be noted that a new successor of AMIE+, called AMIE 3, has been reported recently by the same research group [42]. Some of the implemented enhancements follow the same direction as those present in RDFRules. Since the AMIE 3 system only came out when most experiments on RDFRules had been completed (and the present article was already in a late phase of the review process), we do not present empirical comparisons of both systems here.

3. A Review of AMIE+

The following paragraphs describe the basic features of the AMIE+ algorithm on which our approach builds. While the central concepts of the approach mainly correspond to those from the original AMIE+ paper [2], we translated their definitions to the triple notation of the RDF model and added some further notions that allow us to smoothly progress to the description of our own RDFRules system in the following parts of the paper.

We describe, in turn: the format of source data (RDF triples), the format of rules, the notion of instantiation (of rules by the data), and the significance measures computed for the rules wrt. the data.

3.1. RDF Knowledge Graphs and Triples

AMIE+ mines Horn rules from RDF Knowledge Graphs (abbreviated as KG). A KG consists of a set of statements (or ground facts) in the *triple* form $\langle s, p, o \rangle$, where the predicate p expresses a relationship between the subject s and the object o. In the Semantic Web any subject or predicate is identified by an IRI[8] or (this only holds for the subject) a blank node identifying the resource. The object resource is represented by an IRI, a blank node, or a literal value with some data type. Individual statements may be linked to each other by sharing the same resources within the current graph or even across several graphs.

3.2. Rule Structure

An AMIE+ (Horn) rule $\vec{B} \Rightarrow H$ consists of a single atom H at the head position (consequent) and a conjunction of atoms $\vec{B} = B_1 \wedge \ldots \wedge B_n$ at the body position (antecedent). The *rule length* is the number of atoms in the rule.

3.2.1. Atoms

An AMIE+ atom is a statement in the form of 3-tuple, whose *atom items* are denoted, analogously to those of an RDF triple, as subject, predicate and object. Similarly as in a triple, the predicate of an atom has to be a *constant*, i.e., an IRI. On the other hand, at least one of the subject and object items, or possibly both, are *variables*; the remaining item, if any, is then a constant of the type allowed at this position in a triple (we will thus, for brevity, only use the notion of 'atom'

[8]Internationalized Resource Identifier

for non-ground atoms; the ground ones will be simply called triples).

For example, the atom (?a <livesIn> ?b) contains the variables ?a and ?b, whereas the atom (?a <livesIn> <Prague>) contains only one variable ?a at the subject position and the constant <Prague> at the object position.

Remark on notation The AMIE+ literature describes rules using the Datalog notation [43] common in the ILP domain. An example rule in Datalog notation is:

wasBornIn(a, b) $\Rightarrow$ diedIn(a, b).

In this paper we use an infix notation derived from the RDF serialization Notation3.[9] The same rule is then written as

(?a <wasBornIn> ?b) $\Rightarrow$ (?a <diedIn> ?b).

All variables are prefixed by a question mark and atoms are enclosed in parentheses for better readability of a conjunction of more atoms. A specific triple (containing only constants, i.e. a ground fact) is enclosed in angle brackets, and commas are used as argument separators, to make a distinction from (non-ground) atoms, e.g.,

$\langle$<John>, <livesIn>, <Prague>$\rangle$.

3.2.2. Allowed Forms of Rules

The output rule has to fulfil several conditions. First, the rule atoms must be *connected*. That means variables are shared among the atoms to form one connected graph:

(?a <isMarriedTo> ?c) $\wedge$ (?c <directed> ?b)

$\Rightarrow$ (?a <actedIn> ?b).

Second, only *closed* rules are allowed. A rule is *closed* if each of its atoms contains a variable at the object or/and the subject position and any variable appears at least twice in the rule:

(a closed rule)

(?a <livesIn> ?b) $\Rightarrow$ (?a <wasBornIn> ?b),

(an open rule)

(?a <livesIn> ?c) $\Rightarrow$ (?a <wasBornIn> ?b).

Finally, an atom cannot contain the same variable at both object and subject positions.

[9]https://www.w3.org/2000/10/swap/grammar/n3-report.html

Compared to ILP systems, AMIE+ (and, consequently, also the RDFRules) limits its expressiveness to binary predicates. Unary predicates are implicitly covered by the possibility to include atoms having the *rdf:type* predicate, in a rule; the object of the atom can then be interpreted as the name of a unary predicate in terms of Horn logic. Predicates with higher arity, as well as function symbols, are not considered.

In this respect, Horn rules conform even syntactically to the eco-system of Semantic Web knowledge representations. On the other hand, they do not aim to tightly integrate with the so-called rule languages for the Semantic Web, such as RIF [44] or SWRL [45].[10]

The previous example of AMIE+ closed rule can be rewritten in RIF (which has, by the logic programming conventions, the head atom as the left-hand side) as:

```
Forall ?a ?b
(?a[<wasBornIn> -> ?b]
:- ?a[<livesIn> -> ?b])
```

Note that this RIF rule explicitly includes the universal quantifier, as common in (crisp) first-order logic. In contrast, rules learned by inductive tools such as AMIE+ are often not valid universally but the degree of their validity is quantitatively estimated from data. This limits their direct applicability for predicting new facts in the KGs: the learned rules are typically assumed to be checked by a human expert before they can be used for inference.

Notably, SWRL and RIF also support anonymous concept expressions, not all of which can be learned by AMIE+. For example, the SWRL rule

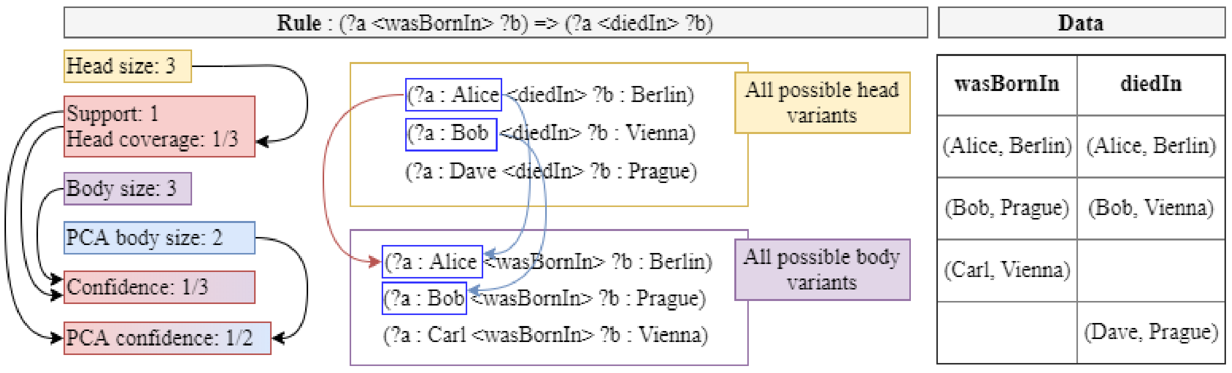
```
livesIn(?a,?b),
livesIn max 1 Place(?a)
-> wasBornIn(?a,?b)
```

expressing that "a person that only lived in one place is assumed to have been born in this place" (where `livesIn max 1 Place` is an anonymous concept expression – a complex unary predicate) is out of the scope for the AMIE+ representation.

3.3. Instantiation and prediction

Since the 3-tuple notation makes the fulfilment of logical formulas less apparent, we have to formally in-

[10]It should be noted that some related approaches, which aim at discovering rules that can be directly added to the ontology, such as that by d'Amato et al. [26], represent the mining results in SWRL.


Rule : (?a <wasBornIn> ?b) => (?a <diedIn> ?b)
Head size: 3
Support: 1
Head coverage: 1/3
Body size: 3
PCA body size: 2
Confidence: 1/3
PCA confidence: 1/2
(?a : Alice <diedIn> ?b : Berlin)
(?a : Bob <diedIn> ?b : Vienna)
(?a : Dave <diedIn> ?b : Prague)
All possible head variants
(?a : Alice <wasBornIn> ?b : Berlin)
(?a : Bob <wasBornIn> ?b : Prague)
(?a : Carl <wasBornIn> ?b : Vienna)
All possible body variants
Data

wasBornIn	diedIn
(Alice, Berlin)	(Alice, Berlin)
(Bob, Prague)	(Bob, Vienna)
(Carl, Vienna)	
	(Dave, Prague)

Figure 1. Computation of measures of significance in AMIE+.

troduce an explicit operator for instantiation (which is only introduced verbally in the AMIE+ paper).

Let the symbol $\prec$ denote the relationship between a ground and non-ground formula such that the ground formula is constructed from the non-ground one via replacing all its variables with constants. We will say that the ground formula *instantiates*[11] the non-ground one. The ground formula can be, in our context, either a triple, or a conjunction of triples, or a ground rule (with all of its atoms being triples). The non-ground formula can be an atom, a conjunction of atoms, or a rule. For the triple-to-atom case, e.g.:

$$\langle\text{<John>, <livesIn>, <Prague>}\rangle \prec (\text{?a <livesIn> ?b}).$$

More formally, a ground formula f_g instantiates a non-ground formula f_n if and only if there exists a *ground substitution* $\theta = \{V_1/c_1, \ldots, V_n/c_n\}$, where $\{V_1, \ldots, V_n\}$ are variables and $\{c_1, \ldots, c_n\}$ are constants, such that the application of the substitution θ on f_n produces f_g. We can say that f_g *instantiates* f_n *under* θ.

For example, the particular substitution and its application of the previous atom instantiation example are:

$$\theta = \{\text{?a/<John>, ?b/<Prague>}\},$$

$$(\text{?a <livesIn> ?b})\theta = \langle\text{<John>, <livesIn>, <Prague>}\rangle.$$

Note that if multiple atoms share the same predicate then one triple can instantiate more than one atom (in the conjunction) at the same time, e.g.:

$$\langle\text{<John>, <hasParent>, <Bob>}\rangle \prec (\text{?a <hasParent> ?b}) \wedge (\text{?a <hasParent> ?c})$$

This effect is known as *non-injective mapping*. Moreover, *injective mapping* does not allow to replace multiple variables with a single constant, therefore, duplicate triples can not occur after instantiation of a conjunctions of atoms.[12]

Next, we have to define the notion of *prediction*. A rule $\vec{B} \Rightarrow H$ predicts a triple t with respect to a given knowledge graph KG if and only if $t \prec H$ under some θ and there is a set of triples $\{t_1, t_2, \ldots, t_n\} \in KG$ such that $(t_1 \wedge t_2 \wedge \ldots \wedge t_n) \prec \vec{B}$ under the same θ.

Note that t may or may not be in the KG. If $t \in KG$ then the rule *correctly* predicts t. If $t \notin KG$ then the rule would *incorrectly* predict t under the CWA; however, under the OWA we cannot say if the prediction is correct or incorrect.

3.4. Measures

Each rule mined from a particular KG has some significance with regard to the chosen measure. Generally, in the context of rule mining, we use *support* and *confidence* as the two main measures of significance.

The speed of discovering the desired rules depends on minimising the search space including all possible rules. This can be achieved by efficient breadth-first search, which prunes the branches that would not yield rules matching the user-defined pruning conditions such as the minimum support threshold.

3.4.1. Atom Size

One of the key functions used to calculate other measures is $size(A)$, which counts the number of

[11]The use of the term 'instantiation' in this context should not be mismatched with the notion of class instantiation with an individual, expressed by the `rdf:type` property in knowledge graphs.

[12]Since non-injective mapping may cause under-estimation of the confidence or over-estimation of the support, newer implementations such as RDFRules as well as the very new successor of AMIE+, AMIE 3 [42], thus allow to switch to injective mapping.

triples in the KG that instantiate the given atom A. The $size(A)$ function is defined as

$$size(A) = \#\langle s, p, o\rangle \in KG : \langle s, p, o\rangle \prec A,$$

where the $\#$ symbol refers to the number of distinct triples. For example, *size*((?a <livesIn> ?b)) returns the number of all distinct triples in the KG with predicate <livesIn>, whereas *size*((?a <livesIn> <Prague>)) returns the number of all distinct triples in the KG with predicate <livesIn> and object <Prague>.

Each rule has a head predicate size, shortly *head size*, which is the number of triples from the KG having the same predicate that occurs in the rule head:

$$hsize(\vec{B} \Rightarrow (s\ p\ o)) = size((?a\ p\ ?b)).$$

For instance, in Figure 1, there is a simple example showing several statements, a rule induced from them, and its measures of significance. Notice that the head predicate <diedIn> is present in three triples; therefore, the head size of the rule is three. (In this particular case it is equivalent to the size of the rule's head, since the head is free of constants.)

All rules returned by AMIE+ have to reach or exceed the minimum head size threshold *minHS*:

$$hsize(\vec{B} \Rightarrow H) \geqslant minHS.$$

3.4.2. Support and Head Coverage

Support is a measure of significance used as the main pruning threshold in AMIE+. In the context of association rule mining, the support of a rule indicates the number of transactions in the database that conform to this rule. The function for calculating the support measure is *monotonous*. This means that once new atoms have been added to the body of a rule and the rule length thus increases, the support of the rule decreases or remains unchanged. This property is crucial for search space pruning. If a rule does not meet the minimum support threshold and thus is considered *infrequent*, then any extension of this rule, created by appending one or more new atoms to its body, is also *infrequent* (this is called the *upward closure property*) [11]. Hence, a whole branch of infrequent extensions can be skipped.

In AMIE+, the support measure is simply defined as the number of correctly predicted distinct triples, i.e., the number of triples that instantiate the head atom H given at least one conjunction of triples instantiating the body $\vec{B}$. The formal definition[13] is as follows:

$$supp(\vec{B} \Rightarrow H) = \#\langle s, p, o\rangle \in KG : \exists t_1, \dots, t_n \in KG \\ : ((t_1 \wedge \dots \wedge t_n) \Rightarrow \langle s, p, o\rangle) \prec (\vec{B} \Rightarrow H)$$

In the example depicted in Figure 1, there is only one triple instantiating the rule head,

⟨<Alice>, <diedIn>, <Berlin>⟩,

for which there is at least one connected triple instantiating the body atom; therefore, the support is 1.

The relative value of support to the head size is called *head coverage* (*hc*):

$$hc(\vec{B} \Rightarrow H) = \frac{supp(\vec{B} \Rightarrow H)}{hsize(\vec{B} \Rightarrow H)}.$$

This measure has the value range from zero to one. The minimal head coverage threshold *minHC* can be used as the relative support threshold for the search space pruning, where:

$$hc(\vec{B} \Rightarrow H) \geqslant minHC.$$

3.4.3. Confidence

The support of a rule only indicates the number of correctly predicted triples, but it does not convey the quality of the prediction made by the rule, because it disregards the false positives. In the context of association rule mining, the main measure of predictive quality of a rule is *confidence*. It expresses the empirical conditional probability of the head of the rule given the body: $p(H|\vec{B})$. AMIE+ uses two variations of confidence: the standard rule confidence and the *Partial Completeness Assumption* (PCA) confidence.

The standard confidence is computed as the ratio of support and *body size* (*bsize*):

$$conf(\vec{B} \Rightarrow H) = \frac{supp(\vec{B} \Rightarrow H)}{bsize(\vec{B} \Rightarrow H)},$$

where *bsize* is the number of distinct triples predicted by the rule:

$$bsize(\vec{B} \Rightarrow H) = \#\langle s, p, o\rangle : \exists t_1, \dots, t_n \in KG \\ : ((t_1 \wedge \dots \wedge t_n) \Rightarrow \langle s, p, o\rangle) \prec (\vec{B} \Rightarrow H)$$

The standard confidence operates under the CWA. That means if some predicted statement is missing in the KG, it should be considered a negative example.

[13]Our definition is equivalent to the definition in the AMIE+ paper [2], which uses the infix notation and relies on the first-order logic semantics: $supp(\vec{B} \Rightarrow r(x, y)) := \#(x, y) : \exists z_1, \dots z_m : \vec{B} \wedge r(x, y)$, with r corresponding to our p.

However, the Semantic Web applications generally operate under the OWA. Hence, for many KGs the standard confidence may not be appropriate [2].

For this purpose, AMIE+ defines the so-called PCA confidence. A predicted statement $\langle s, p, o\rangle$ which is missing in the KG is only considered a negative example if the subject s already appears in the KG with property p (and with some other object o'). For instance, if the subject s does not have the property <isMarriedTo> within any triple, s may not necessarily be unmarried. However, once we know of some triple $t' = \langle s, \text{<isMarriedTo>}, o'\rangle$, we can assume that the KG involves *all* facts associated with subject s and predicate <isMarriedTo>. As a consequence, all other missing statements related to this predicate and subject are regarded as negative examples. The PCA confidence is defined as:

$$conf_{pca}(\vec{B} \Rightarrow H) = \frac{supp(\vec{B} \Rightarrow H)}{bsize_{pca}(\vec{B} \Rightarrow H)},$$

where $bsize_{pca}$ is defined as follows. Let $dsize_{KG}(p)$ denote the number of unique subjects in the (empirical) domain of p in KG, i.e., $\#s : \langle s, p, o\rangle \in KG$, and, analogously, $rsize_{KG}(p)$ the number of unique objects in the (empirical) range of p in KG, i.e., $\#o : \langle s, p, o\rangle \in KG$. If $dsize_{KG}(p) \geqslant rsize_{KG}(p)$ then we can call the subject position the *higher-cardinality side* of the predicate, and the object position will be the *lower-cardinality side*. Then:

$$\begin{aligned} bsize_{pca}(\vec{B} \Rightarrow H) = \#\langle s, p, o\rangle &: \exists t_1, \ldots, t_n, t' \in KG \\ &: ((t_1 \wedge \ldots \wedge t_n) \Rightarrow \langle s, p, o\rangle) \prec (\vec{B} \Rightarrow H) \\ &\wedge t' = \langle s', p, o'\rangle \end{aligned}$$

such that $s' = s$ if $dsize_{KG}(p) \geqslant rsize_{KG}(p)$, and $o' = o$ otherwise.

The motivation for altering the formula according to the higher/lower cardinality of the predicate sides[14] follows from the intuition that statements are typically filled to RDF KGs starting from the higher-cardinality side of a property, since the resource descriptions are then smaller and more manageable. The PCA confidence assumes that those smaller resource descriptions have already been created as complete in terms of containing all values of every property, for a given subject. The completeness may however not hold in the opposite direction of the property. Therefore, if the object position is the higher-cardinality position for some property p (which is a situation less often observed in practice, as Galárraga et al. [2] witnessed), $bsize_{pca}$ checks if the *object* of the predicted triple is already connected, using p, to some *subject* in the KG (for this predicate) and not the other way around.

An example of PCA confidence is depicted in Figure 1. The empirical domain of (?a <diedIn> ?b) will likely be larger than its empirical range, since a person died at one location (at the given granularity, such as that of a city) at the most, while many people typically died at the same location. While there are three triples instantiating the body of the rule (?a <wasBornIn> ?b), i.e. the body size is three, only for two subjects of these triples holds that there also exists at least one triple containing the head predicate <diedIn>: both the standard and PCA confidence assume that the rule correctly predicts Alice's death in Berlin and incorrectly predicts Bob's death in Prague). About the third subject <Carl> we have no record relating to his death; for the standard confidence, his predicted death in Vienna is counted as incorrect, which is however not the case for the PCA confidence (note that Carl's death record may be just missing – perhaps the whole data snapshot has been taken prior to his death – and he may still have died in Vienna). Therefore the PCA confidence is higher than the standard confidence, for this rule.

3.5. Rule Mining

Before the mining phase, all input parameters should be pre-set with respect to a specific task. AMIE+ uses three main input parameters that affect the speed of mining the most: the maximum rule length, the minimum head size *minHS* and the minimum head coverage threshold *minHC*. These input parameters can be further extended by the minimum confidence threshold (standard or/and PCA), constants that must occur in a rule, the number of threads, and other rule constraints mentioned in [2].

The AMIE+ algorithm (see Algorithm 1) first enumerates all atoms whose size is higher than the preset head size threshold *minHS* (lines 4 to 8). All these atoms become the heads of rules, and will be further expanded. Each head is saved into a queue which collects all potential rules to be either refined or moved into the result set. The queue can be processed in parallel since each rule forms its own branch within the search space and the refinement process does not alter any previous state. Moreover, the input KG, from

[14]The AMIE+ paper rather speaks about the degree of 'functionality' and 'inverse functionality' of the predicate, which may however mix up with the corresponding boolean characteristics of properties in ontologies.

which the rules are constructed and measures counted, is also immutable.

The algorithm gradually passes the rules from the queue to the *refine* operation, which adds one atom at a time to the body of the rule (line 15). The added atom is either dangling (the output rule is open), or closing (the output rule is closed), or containing a constant:

(rule before refinement)

$$\varnothing \Rightarrow \text{(?a <wasBornIn> ?b)},$$

(closing atom added)

$$\text{(?a <livesIn> ?b)} \Rightarrow \text{(?a <wasBornIn> ?b)},$$

(dangling atom added)

$$\text{(?a <livesIn> ?c)} \Rightarrow \text{(?a <wasBornIn> ?b)},$$

(instantiated atom added)

$$\text{(?a <livesIn> <Prague>)} \Rightarrow \text{(?a <wasBornIn> ?b)}.$$

The extended rule is further added into the queue only if it exceeds the pre-set pruning thresholds (line 17). The queue is being processed until it is empty. All found rules which satisfy all mining constraints are stored in the result set (lines 11 to 13). When refining a rule, the algorithm calculates all the measures of significance needed for pruning. Furthermore, the queue is designed to eliminate any duplicate rules, thus making the mining process much faster. For detailed information about the mining algorithm refer to [2].

Algorithm 1. *The basic workflow of AMIE+.*

```
1  function AMIE(maxRuleLength, minHS, minHC, KG)
2    out = {}
3    queue = ()
4    for each predicate $p : p \in t \wedge t \in KG$ do   // t is a triple
5      if size((?a $p$ ?b)) $\geqslant$ minHS then
6        queue.enqueue($\varnothing \Rightarrow$ (?a $p$ ?b))
7      end if
8    end while
9    while queue.nonEmpty do in parallel
10     rule = queue.dequeue()
11     if acceptedForOutput(rule) then
12       out += rule
13     end if
14     if length(rule) < maxRuleLength then
15       refinedRules = refine(rule)
16       for each newRule in refinedRules do
17         if hc(newRule) $\geqslant$ minHC $\wedge$ newRule $\notin$ queue then
18           queue.enqueue(newRule)
19         end if
20       end for
21     end if
22   end while
23   return out
24 end function
```

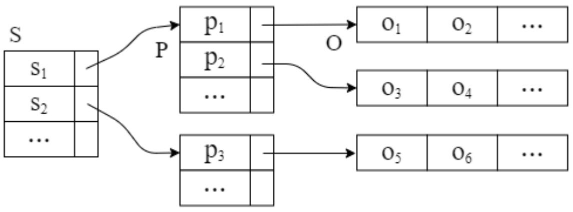


Figure 2. A sample of the SPO fact index.

3.6. Memory Indexing

To enable fast rule refinement and measure computation, the AMIE+ algorithm uses an in-memory index containing all the triples of the analyzed KG. The index consists of six *fact indices*: SPO, SOP, PSO, POS, OSP, and OPS. Each fact index is a hash table containing other, nested hash tables. For example, for the SPO fact index, depicted in Figure 2, a subject s points to a subset of predicates P where each predicate $p \in P$ points to a subset of objects O.

4. Limitations of AMIE+

As noted earlier, AMIE [46] and consequently AMIE+ [2], constituted a breakthrough in rule mining from RDF graph data. AMIE+ very well addresses the core of the rule mining problem – extracting an exhaustive set of rules, given RDF data and the right settings (typically user-set minimum support and confidence thresholds). At the same time, the algorithm as well as the accompanying implementation pay relatively little attention to the problems of data pre-processing, meta-parameter tuning, and post-processing; these steps are assumed to be addressed by external algorithms and software components.

Successful systems in the domain of rule mining from tabular and transactional data, such as the popular *arules* ecosystem [9], offer an integrated approach supporting the complete data mining life cycle: from pre-processing the input data to the selection of representative rules. Some other association rule learning packages also support automatic tuning of the rule learning meta-parameters, such as of the minimum support threshold [47].

An integrated approach is even more necessary in the linked data context as general algorithms for data pre-processing are difficult to apply to RDF datasets due to the different structure of inputs as well as outputs. For example, numerical attributes need to be discretized (quantized, binned, categorized [48]) prior to association rule mining. Discretization – as a pre-processing step – is supported in many libraries for tabular or CSV data, yet to our knowledge, there is no such tool for RDF triples.

Also, in the tabular or transactional context, it is typically computationally feasible to pre-process all available data. However, for RDF data such an non-discriminative approach to pre-processing may be prohibitively expensive. Instead, it is desirable to integrate data pre-processing (such as discretization) directly into the mining algorithm, so that only the values that can prospectively appear in the generated rules are processed.

Similar observations hold for the post-processing of results. In association rule mining from tabular or transactional data, it is sufficient for the mining algorithm to support only coarse requirements on the rules mined: it is computationally cheap to mine more rules and then apply finer requirements on the content of the mined rules within post-processing. However, the size of RDF KGs and the richer expressiveness of Horn rules make such an approach prohibitively expensive. While AMIE+ was very fast in processing synthetic benchmarks, as reported in [2], it may still be unusably slow or memory-intensive in practical tasks where – for example – the user does not know the precise minimum support threshold.

Some of the limitations described in this section do not only call for enhancing or extending the functionality of AMIE+ as a tool but also to inefficiencies we identified within the core AMIE+ algorithm. This is the case of repetitive and exhaustive calculations performed during the mining process.

Finally, some limitations relate to the lack of various features which were found useful for mining rules from transactional or tabular data. These include, such as the support for rule mining with the top-*k* approach or support for selection of most representative rules (by clustering or pruning). Other limitations stem from absence of features generally required from systems processing linked data, such as the support for multiple graphs.

4.1. Inability to Process Numerical Data

Association rule learning algorithms do not work well with numerical data due to the downward closure property, which requires that not only the complete rule but also each subset of atoms composing it should meet the minimum support threshold. Since numeric attributes typically have many values, a single distinct value may not assure the required support. Such a value will thus be excluded from all generated rules.

Consider an RDF dataset with prices of public contracts. This dataset may include many facts containing a particular price of a contract. Each of these facts contains a numerical value at the object position related to a specific contract, e.g.,

$$\langle \text{<Contract-1>}, \text{<hasPrice>}, 40000\rangle.$$

If every contract has just one price which is unique in the dataset, then any atom in the form (?a <hasPrice> *C*), where *C* is a numerical constant, has the size of at most one. In AMIE+, for a higher minimum support threshold, the above-mentioned atom with the constant will not be contained in any rule as a head atom due to the small atom size. For a rule $\vec{B} \Rightarrow H$ where the body consists of only one atom with a constant, e.g.,

$$(\text{?a <hasPrice> C}) \Rightarrow$$
$$(\text{?a <authority> <MinistryOfDefense>}),$$

the atom size must be greater than or equal to the minimum support threshold. For more atoms in the body the minimum size of an atom with a constant depends on other atoms in the body and on user-defined thresholds (more in 5.2).

An approach used to address this problem in association rule learning frameworks for transactional data, such as the *arules* library, is to replace multiple neighbouring distinct values of a numerical attribute with an interval of values.

4.2. Absence of the Top-k Approach

Decades of research in association rule learning and frequent itemset mining continuously show how difficult it is for users to set the minimum support threshold properly [49, 50]. Similarly to the standard association rule learning, AMIE+ will generate all rules complying with the user-set support threshold. A too-small threshold leads to an enumeration of too many – millions and more – frequent itemsets (and consequently, rules), eventually resulting in an out-of-memory situation. In contrast, a too high threshold value may return no results.

Generating all rules already posed problems when association rule learning was executed on transactional databases like those collecting the contents of shopping baskets in a supermarket, where the analyst could still use their knowledge of the analyzed data to set these thresholds. This problem is further exacerbated when data may be very large.

In the top-*k* approach, the user is only returned the *k* rules with the highest values of the chosen measure, rather than all rules. This approach allows additional pruning strategies, alleviating or completely removing the risk of combinatorial explosion, the biggest problems of association rule mining [49, 51].

4.3. Coarse Rule Patterns

Association rule learning tasks are in constant risk for combinatorial explosion, even on small datasets. This problem cannot be completely addressed by the top-*k* approach alone. Without additional guidance by the user, the top-*k* approach often generates rules that reflect patterns in data that are obvious or uninteresting for the user. For this reason, association rule learning frameworks provide various means for controlling the content of the generated rules. For example, in the *arules* library, the user can set a list of *items* (attribute-value pairs) that can appear in the antecedent and consequent of the generated rules. The LISp-Miner system[15] offers much ampler capabilities: it provides a structure of an arbitrary number of granular patterns that the rule must match in order to be generated.

AMIE+ adopts a similar approach to *arules* when it allows the user to provide a list of *relations* that should be included in (or excluded from) the body and head of the rule. In addition, there are several linked data specific settings relating to constants. The user can choose whether the constants are allowed, or even enforced, in all atoms of the generated rules. This approach, which is taken in AMIE+, does not take full advantage of the RDF data model. In particular, it is not possible to define independent, fine-grained constraints on subjects, predicates, and objects appearing in the discovered rules. For example, the user may wish to mine for rules that contain a triple with a specific value in the antecedent, such as:

$$(\text{?a rdf:type dbo:Writer}) \wedge \cdots \Rightarrow (\text{?a ? ?}).$$

This pattern covers all rules where the consequent contains variable ?a at the subject position, where ?a has to cover an instance of the dbo:Writer class. Informally, the user wishes to find all rules that involve writers. Such a pattern cannot be enforced in AMIE+.

[15]http://lispimner.vse.cz

4.4. Repetitive Calculations

During the refinement process, AMIE+ binds variables to constants in order to count the support for each fresh[16] atom separately: AMIE+ first constructs the closing atoms, then the dangling atoms and finally, the instantiated atoms. For example, let the following rule be subject to the refinement process:

$$(\text{?a <livesIn> ?b}) \Rightarrow (\text{?a <wasBornIn> ?b}).$$

AMIE+ sequentially adds the fresh atoms (?a ?p ?b) and (?b ?p ?a) as closing atoms, and (?a ?p ?c), (?c ?p ?a), (?b ?p ?c) and (?c ?p ?b) as dangling atoms, to the rule body or head. Each newly added atom is connected to the rule, and the variables ?p, ?a, ?b and ?c are bound with constants in order to calculate the support measure and to enumerate the instantiated atoms.

This approach results in repetitive calculations, mainly in terms of variables binding. For example, the binding process starting from (?a <livesIn> ?b) and (?a <wasBornIn> ?b) is performed repeatedly for each fresh atom.

4.5. Exhaustive Calculations

AMIE+ first computes the value of support for each refined rule, and only afterwards it applies the pruning step based on a chosen support threshold. The same technique is used for the confidence calculation, where the algorithm first computes the confidence value and then filters the rules using confidence thresholds.

Searching of all triples instantiating the rule, which is necessary for computing the final value of confidence and support, may be very expensive. The process outlined above, used in AMIE+, can be made more efficient by terminating this counting early when it becomes clear that the final result of confidence or support will not meet the threshold.

4.6. Lack of Support for Multiple Graphs

In Semantic Web terms, a KG identified by a specific IRI is called a *named graph*.[17] Multiple graphs can be part of one dataset if each triple has assigned the information about association to a particular graph. This structure is called a quad $\langle s, p, o, g \rangle$, where g is the IRI of the graph. Same resources from various graphs stored under different identifiers can be unified

[16]"Fresh" is a term used in [2] to denote a newly added atom.
[17]https://www.w3.org/TR/rdf11-concepts/

by interconnecting them using the *owl:sameAs* property.

AMIE+ does not support mining across multiple graphs such that one rule would contain resources from two or more different graphs. Also, AMIE+ is not able to resolve the *owl:sameAs* predicate to join resources from two different namespaces. For example, consider the following statements:

⟨dbr:Sofia, owl:sameAs, nyt:N8209⟩,

⟨dbr:Sofia, rdf:type, dbo:PopulatedPlace, <dbpedia>⟩,

⟨nyt:N8209, rdf:type, opengis:Feature, <nytimes>⟩.

AMIE+ would not be able to infer that

⟨nyt:N8209, rdf:type, dbo:PopulatedPlace⟩.

In AMIE+, to find this rule it would be necessary to merge these statements into a single graph. However, it is not possible to track the provenance of individual atoms in the discovered rules. Additionally, the discovered rules would have to contain extra atoms corresponding to the *owl:sameAs* relation, which can have a significant impact on the time of rule mining [2].

4.7. Lack of Support for Rule Clustering and Pruning

Association rule discovery can result in the generation of a high number of potentially interesting rules. Grouping – or clustering – of similar or overlapping rules can be effective for presenting the mining results in a concise manner to the end user.

Association rule learning frameworks, such as *arules*, provide, for this purpose, measures that express the similarity between association rules. Such support for rule clustering is not provided in the scope of AMIE+.

Another approach for addressing the problem of too many rules on the output is removal of some of the rules based on analysis of their overlap with respect to the input data. In rule learning literature, this supervised process is often called *pruning* [52]. While pruning may not be applicable for explorative association rule learning[18], where the goal is to find all rules valid in the data that match the user-defined interest measure thresholds, it is a key ingredient of adaptations of association rule learning for classification [54].

Some search strategies supported by ILP systems, such as ALEPH, are able to discover concise theories consisting of a small number of rules covering the input KG. Achieving the same coverage, AMIE+ mines all rules matching the user-specified mining thresholds without any pruning strategies, which would remove overlapping or redundant rules.

[18]E.g., the motto of the GUHA rule learning framework [53] is "GUHA offers everything interesting", which translates as "all hypotheses of the given form true in the data".

5. Proposed Approach

In the following, we present a collection of enhancements to AMIE+ that address the limitations summarized in the previous section.

5.1. Faster Projection Counting

AMIE+ recursively binds variables each time when new atoms are added. The binding process is important for finding valid connections to a rule being refined and for calculation of the support measure. However, it has a major impact on the overall mining time.

During the refinement process of a rule $\vec{B} \Rightarrow H$, AMIE+ constructs the set of new atoms A_n which includes all closing and dangling variants compatible with the rule being refined. A new atom $(x\ ?\mathrm{r}\ y) \in A_n$ contains the relational variable ?r, which is not yet bound, and the variables x and y, where each of them either closes another variable or is dangling. For each new atom, a count projection query is run. Furthermore, AMIE+ also runs the count projection query for each dangling atom while searching for instantiated rules.

The count projection query (described in Algorithm 2) binds all variables in the rule and enumerates all bound variants of the newly added atom connected to the rule with a cardinality. This process is inefficient in that it may redundantly bind variables in atoms that have already been mapped in the past (line 7 and 8).

Algorithm 2. *The AMIE+ count projection query*

```
1  function countProjections(A_n, $\vec{B} \Rightarrow H$, k, KG)
2    out = {}
3    for each $(x\ ?r\ y) \in A\_n$ do
4      map = empty HashMap //where key is a predicate and value is an integer
5      $q = \vec{B} + H + (x\ ?r\ y)$
6      for each $\langle s,p,o \rangle \prec H : \langle s,p,o \rangle \in KG$ do
7        q' = bind(q, $\langle s,p,o \rangle$, H)
8        $\chi$ = select(?r, q')
9        for each $r \in \chi$ do
10         if ¬map.contains(r) then map[r] = 1 else map[r]++
11       end for
12     end for
13     for each $(r \rightarrow n) \in$ map : $n \geqslant k$ do
14       out += $((x\ r\ y) \wedge \vec{B} \Rightarrow H)$
15     end for
16   end for
17   return out
18 end function
```

The *bind* function (line 7) maps all variables of atom H to constants of triple $\langle s, p, o\rangle$ and propagates this binding to all shared variables in q. The *select* query function (line 8) recursively binds all variables in q' until all possible bindings for variable ?r are collected and returned. The *map* is a hashtable containing the cardinality for each binding of variable ?r. The cardinality n must be at least k, where k is the minimum support threshold (line 13).

In our approach, we try to reduce the number of calls to the binding functions (in line 6 to 12 of Algorithm 2). In the refinement process, the bindings of the head atom H should be performed only once. This process is described in Algorithm 3 within the *refine* function, which returns the set of atoms to be added into the rule being refined.

Algorithm 3. *The RDFRules refinement process*

```
1  function refine($\vec{B} \Rightarrow H, k$, KG)
2    map = empty HashMap
3    $q = \vec{B} + H$
4    maxSupp = 0
5    remainingSteps = size(H)
6    A_n = newAtomVariants($\vec{B} \Rightarrow H$)
7    for each $\langle s,p,o\rangle \prec H : \langle s,p,o\rangle \in KG$ do
8      A_r = bindProjections(A_n, $\vec{B}$, bind(q, $\langle s,p,o\rangle$, H), KG)
9      for each x ∈ A_r do
10       if ¬map.contains(x) then map[x] = 1 else map[x]++
11       maxSupp = max(map[x], maxSupp)
12     end for
13     remainingSteps = remainingSteps - 1
14     if maxSupp + remainingSteps < k then
15       return {}
16     end if
17   end for
18   return $\{(x \wedge \vec{B} \Rightarrow H) : map[x] \geqslant k\}$
19 end function
```

The *bindProjections* function, described in Algorithm 4, is called for each instance of the head atom (lines 7 and 8). Notice that the binding is performed for all added closing and dangling atom variants together (lines 6 and 8). In each iteration, the *bindProjections* function returns a set of atoms A_r, with a resolved relation and an instantiated dangling variable, which is connected to the current instance of the head H and to the remaining atoms of the body $\vec{B}$ (line 8). At the end of each iteration, the atoms from A_r are added to the hashtable *map* and the atom cardinality $map[x \in A_r]$ is increased by one (line 10). Finally, we add only such atoms to the rule for which $map[x \in A_r] \geqslant k$, where k is the minimum support threshold (line 18).

Remark. *If the A_r set is empty, the current binding of the head $\langle s, p, o\rangle$ can be omitted within any other refinements of subsequent rules having the basis of the current rule.*

The whole refinement process can be completed faster if we know that none of the found atoms in a certain moment can reach the minimum support threshold. The variable *remainingSteps* holds the number of iterations that still have to be done within the count projections query (lines 7 to 17). If

$$maxSupp + remainingSteps < k,$$

where *maxSupp* is the support value of a new atom with the highest cardinality, we can immediately terminate the refinement process since adding none of the atoms will lead to reaching or exceeding the support threshold k (line 14 to 16).

Algorithm 4. *The RDFRules bind projections query*

```
1  function bindProjections(A_n, $\vec{B}$, q, KG)
2    A_r, A_c, A_b = {}
3    // PHASE I
4    for each (x ?r y) ∈ A_n do
5      if $z \in \{x, y\} : z \in q \vee isDangling(z)$ then
6        A_c += (x ?r y)
7      else
8        A_b += (x ?r y)
9      end if
10   end for
11   // PHASE II
12   if ¬isEmpty(A_c) ∧ exists(q) then
13     for each (x ?r y) ∈ A_c do
14       A_r = A_r ∪ bindFreshAtom((x ?r y), q)
15     end for
16   end if
17   if ¬isEmpty(A_b) then
18     // PHASE III
19     best = argmin_i(size(B_i ∈ $\vec{B}$))  // B_i is the i-th atom of $\vec{B}$
20     for each (x ?r y) ∈ A_b do
21       if size((x ?r y)) ⩽ size(best) then
22         $\chi$ = bindFreshAtom((x ?r y), q)
23         for each (**x** p **y**) ∈ $\chi$ do
24           q' = bind(q, p, (x ?r y))
25           if exists(q') then A_r += (x p y)
26         end for
27         A_b = A_b \ (x ?r y)
28       end if
29     end for
30     if ¬isEmpty(A_b) then
31       // PHASE IV
32       for each $\langle s,p,o\rangle \prec best : \langle s,p,o\rangle \in KG$ do
33         A_r = A_r ∪ bindProjections(A_b, $\vec{B}$ - best, bind(q,$\langle s,p,o\rangle$,best), KG)
34       end for
35     end if
36   end if
37   return A_r
38 end function
```

The *bindProjections* function invocation is composed of four phases. In the first phase, new atoms are divided into two sets: bound atoms A_b and unbound atoms A_u (lines 4 to 10). All variables of the atoms in A_b can be immediately bound by the function *bindFreshAtom* in the second phase (line 12 to 16). This function returns the set of instantiated atoms with all possible relations and instantiated variables with respect to variables so far bound in q. In this phase, the binding is valid only if there *exists* a binding for all remaining unbound variables in q where all atoms are connected (line 12). In the third phase, which is performed only if A_b is not empty, the *best* unbound atom with the smallest size is selected from $\vec{B}$ (line 19). For each unbound atom from A_u with a smaller size than $size(best)$, the binding process is performed, and the connectivity with other atoms is checked as in the second phase (lines 20 to 29). If A_b is still not empty after the third phase, then the fourth phase is performed. In the fourth phase, the *best* atom is bound with a particular instance, and the bind projection query is recursively called with this new binding, without the *best* atom in $\vec{B}$ and with remaining unbound atoms from A_u (lines 32 to 34). The recursion is stopped after all fresh atoms have been resolved (A_b is empty). All resolved atoms are saved into A_r and returned on line 37.

This approach eliminates a considerable number of repetitive calculations and makes AMIE+ much faster, especially for mining of rules with constants (see evaluation in Section 7).

5.2. Processing of Numerical Attributes

As noted in Section 4.1, the standard solution used for handling numerical data in association rule mining is to perform the discretization (or binning) of numeric values with low frequencies into intervals. This step not only reduces the search space, but can also increase the support of some rules.

The proposed approach is performed within pre-processing, however, it is not completely decoupled from the mining phase. In order to avoid generation of intervals that are too narrow or unnecessarily broad for the purpose of the mining task, the user needs to provide the minimum head coverage threshold *minHC* and minimum head size threshold *minHS* as parameters to the pre-processing algorithm. These thresholds are then used to inform the discretization process.

5.2.1. Preliminaries

Data Type Determination As a first step, all predicates that have a numerical range have to be found. Most RDF serializations allow to directly encode data types, but it is up to the data producer to make use of this feature. Data types can also be determined from associated RDF vocabularies, which contain information about the range of the predicates.

Equal-Frequency Binning Equal-frequency binning is a common approach for discretizing numerical data for association rule learning, e.g., it is the default discretization method in the popular *arules* package. In the following, we discuss how this approach can be adapted for discretizing data in knowledge graphs.

In the context of our approach, the frequency of an interval I of predicate p is calculated as $size((?\mathsf{a}\ p\ I))$, where each triple $\langle s, p, o\rangle \in KG$ instantiates the atom $(?\mathsf{a}\ p\ I)$ if o is in the range of I. Let L_p be the sorted list of numerical objects of predicate p. Intervals are gradually constructed by merging the numerical values from the smallest number to the largest number of L_p until the frequency of the currently expanded interval reaches or exceeds a predefined threshold. The next interval is constructed from the largest number of the previous interval, as the excluded endpoint, by the same way. If the last interval does not reach the required frequency, it is merged with the previous interval. For example, the sorted list

$$L_{\text{<hasGrade>}} = (1, 1, 2, 2, 2, 3, 3, 4, 4, 5)$$

can be divided into two equal-frequent intervals with respect to the discretization threshold 5:

$$I[1; 2] = (1, 1, 2, 2, 2)$$

$$I(2; 5] = (3, 3, 4, 4, 5).$$

Discretization in the Rule Head Let an instantiated atom $H_i = (?\mathsf{a}\ p\ C)$, containing a numerical literal C, be derived from an atom $H = (?\mathsf{a}\ p\ ?\mathsf{b})$. To ensure that atom H_i can appear as the consequent of a rule, the following inequality must be true:

$$size(H_i) \geqslant size(H) \cdot minHC, \tag{1}$$

where $size(H) \geqslant minHS$. This observation is beneficial for merging numerical values in atoms into intervals, e.g., by the equal-frequency binning in order to reach the minimum support threshold required for a head with a constant. A constructed interval I should be broadened until the atom $(?\mathsf{a}\ p\ I)$ satisfies the condition in Eq. 1.

Discretization in the Rule Body The minimum support threshold depends on the minimum head coverage threshold and the head size of the rule; therefore, there are different minimum support thresholds for different heads. In AMIE+, the support is the number of correct predictions of the rule. Consider a rule with one atom A in the body. Such a rule predicts as many triples as is the size of this atom; therefore, if the following condition for the size of the atom A is not met, such a rule cannot reach or exceed the minimum support threshold:

$$size(A) \geqslant hsize(A \Rightarrow H) \cdot minHC. \tag{2}$$

This condition can be used to merge the numerical value in atom A with neighbouring values into an interval I. Values are merged until the frequency of this interval is greater or equal to the minimum support threshold. Compliance to this condition ensures that for a rule containing only the atom (?a p I) in the body the number of predicted triples will be greater than or equal to the minimum support threshold. However, not all of the predicted triples will be necessarily predicted correctly – the predictions will not match the head of the rule; therefore, if an atom complies to the condition in Eq. 2, it is not guaranteed that the support of a rule including this atom will be high enough to meet the support threshold.

Note that even if an atom does not meet the condition in Eq. 2, a rule containing this atom can still meet the minimum support threshold if this rule contains more than one atom in the body. For example, consider the following rule:

(?a <wasBornIn> ?b) ∧ (?b <hasArea> 232.14) ⇒

(?a <diedIn> ?b)).

If the size of the atom (?a <hasArea> 232.14) is lower than the minimum support threshold, then the whole rule can still reach or exceed this threshold since the atom is connected to another atom, which can have a larger size.

5.2.2. *Proposed Discretization Heuristic*

In the previous, we showed that – given the *minHC* and *minHS* thresholds and considering only rules with at most one atom in the body – a head atom and a body atom need to satisfy the requirements in Eq. 1 and Eq. 2, respectively, to be included in a rule. While these conditions are quite restrictive – in practice, mining tasks will involve also other measures of significance and longer rules – we use these conditions as a basis of the proposed general heuristic to guide interval generation in the pre-processing phase.

Let P be a set of predicates occurring in the input KG that is constrained by the minimum head size threshold:

$$P = \{p : size((?\text{a}\ p\ ?\text{b})) \geqslant minHS\}.$$

We define the *lower bound* of the minimum support threshold as:

$$MinSupp_L = \min_{p \in P}(size((?\text{a}\ p\ ?\text{b}))) \cdot minHC.$$

In the proposed heuristic, the frequency of any constructed interval I must be greater than or equal to $MinSupp_L$. In other words, the lower bound $MinSupp_L$

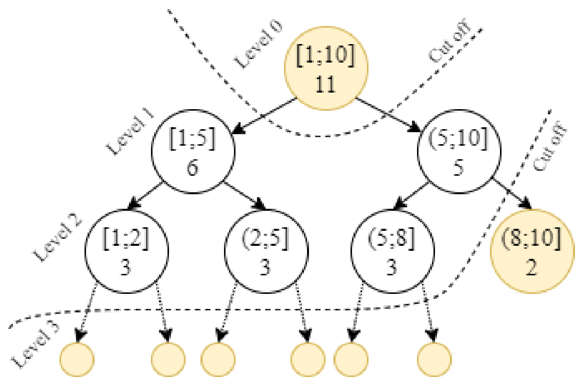


Figure 3. An example of the tree of intervals for some predicate where $MinSupp_L = 3$ and $MinSupp_U = 5$.

expresses the minimum possible atom size requirement for an atom A containing an interval I at the object position to be included in the rule with the smallest head size:

$$size(A) \geqslant MinSupp_L \qquad (3)$$

We define the *upper bound* of the minimum support threshold. This expresses the maximum possible atom size requirement for an atom A containing an interval I at the object position to be included in the rule with the highest head size.

$$MinSupp_U = \max_{p \in P}(size((?\text{a}\ p\ ?\text{b}))) \cdot minHC.$$

Once $MinSupp_L$ and $MinSupp_U$ have been computed based on the user-set *minHC* and *minHS* thresholds, we can use these values to create a *tree of intervals* for each numerical predicate in the input KG.

Tree of Intervals Suppose we want to discretize the list L_p of all numerical values related to a predicate p. First, we aggregate all numerical values from L_p to the root of the tree, which is denoted as the zero-level interval I_p^0 with the range from the minimum value to the maximum value of L_p. We recursively split each interval I_p^i into two smaller equal-frequent intervals $I1_p^{i+1}$ *and* $I2_p^{i+1}$. If the frequency of a newly constructed interval I_p^i does not satisfy the $MinSupp_L$ condition in Eq. 3 (where A = (?a p I_p^i)), the splitting is stopped and the interval is discarded. Finally, the algorithm cuts off intervals whose all children have the interval size greater than or equal to $MinSupp_U$. An example of a tree of intervals is depicted in Figure 3.

Data Enrichment All created intervals generate new triples which are added into the input KG in the pre-processing phase before mining. For each triple

$\langle s, p, o\rangle \in KG$ and each node I_p^i of the tree of intervals related to the predicate p, we add a new triple $\langle s, p_i, I_p^i\rangle$ into the input KG if the number o is in the range of I_p^i and the interval covers more than one number.

The original data in the KG are preserved, since the newly added triples containing intervals use a different predicate (namely, one equipped with a tree-level suffix). Hence, this solution does not reduce the search space of the rule mining algorithm but rather expands it.

Discussion For any atom containing an interval with the size lower than $MinSupp_L$ the conditions in Eq. 1 and Eq. 2 cannot be met. Conversely, if an atom has a size greater than or equal to $MinSupp_U$ the conditions in Eq. 1 and Eq. 2 are satisfied for any head of a rule; therefore, it is not necessary to create intervals with a higher frequency.

The lower and upper bounds reduce the number of constructed intervals and thus limit the overall increase in the size of the enrichment of the input KG generated by discretization. Nevertheless, the heuristic can incorrectly skip intervals with a low frequency, since, as previously discussed, if a body with more than one atom is considered, atoms with size lower than $MinSupp_L$ can still be included in the rule to satisfy the minimum support threshold. Conversely, if an atom containing an interval satisfies the condition in Eq. 2, the whole rule containing this atom must still correctly predict as many triples as required by the minimum support threshold. More potentially useful or interesting rules could, therefore, be generated if the $MinSupp_L$ and $MinSupp_U$ thresholds were not applied.

Despite these limitations, the experiments (cf. Section 7.4) showed that the proposed merging of numerical values results in discovery of many new rules which contain predicates that would never appear in the output when mining directly from the original data.

5.3. Multiple Graphs

Efficiently working with multiple graphs requires data structures supporting *quads* throughout the learning process. In AMIE+ this is not supported – the resulting rule always consists of atoms corresponding to triples.

In our approach, atoms in a rule may be extended with a fourth item that indicates the graph assignment; such an extended rule is called a *graph-aware* rule. This additional item is always generated as a specific graph resource and not as a variable. For instance,

(?a <wasBornIn> ?b <YAGO>) $\Rightarrow$
(?a dbo:deathPlace ?b <DBpedia>).

The discovery of such a rule of course depends on the previously established identity of the resources across the used graphs. For instance, this condition is not met for concepts in YAGO and DBpedia, because the YAGO resource <Prague> has a different description than the DBpedia resource <http://dbpedia.org/resource/Prague> (abbreviated as dbr:Prague). Assuming that a source of triples with the owl:sameAs predicate are given, this inconsistency can be resolved by *owl:sameAs* predicate, which would join these two descriptions:

$\langle$<Prague>, owl:sameAs, dbr:Prague$\rangle$

Support for named graphs needs also to be reflected in the user-set pattern for rule learning. Before mining, the user can decide which atoms (or, how many of them) will be only based on triples from a certain graph, or whether to enable *graph-aware* mining at all. For example, the user can define the following pattern of a mining task where all rules have to consist of parts belonging to various graphs:

(?a ? ?b <YAGO>) $\Rightarrow$ (?a ? ?b <DBpedia>).

This pattern is applied in the mining process, which returns only such matching rules where the body atom is from YAGO, and the head is from DBpedia. The notation used in the pattern above is introduced in detail in the following Section 5.4.

The inclusion of graph information in the rule mining process also requires an extension of fact indices, described in 3.6. These indices allow to check the affiliation to a given graph in constant time for all predicates (PG), predicate-subject pairs (PSG) and predicate-object pairs (POG), and for any predicate-subject-object triples (PSOG).

5.4. Improvements to Expressiveness of Rule Patterns

AMIE+ only provides basic capabilities for restricting the content of the generated rules. These are generally limited to providing a list of predicates that can appear in the antecedent and consequent of the generated rules. Inspired by the LISp-Miner system, we defined a formal grammar-based pattern language (used in the RDFRules framework) for expressing more complex rule patterns in order to find desired rules for a specific task.

Consider $IPG = (N, \Sigma, \Pi, \Theta, P, S)$ as the item pattern grammar to generate any valid pattern for an atom item, e.g., grammar for items p, s and o in atom (s p o). N is a set of all non-terminal symbols, Σ is a set of all terminal symbols for resources and literals occurring in the input KGs, Π is a set of all terminal symbols for variables, Θ is a set of all special terminal symbols especially for grouping, P is a set of all grammar rules applicable in this grammar, and S is the start symbol representing the whole item pattern:

$$\begin{aligned}
&N = \{A, B, C\},\\
&\Sigma = \text{constants from input KGs},\\
&\Pi = \{?\text{a}, ?\text{b}, \ldots, ?\text{z}\},\\
&\Theta = \{?, ?_v, ?_c, \neg, [,]\},\\
&P = \{A \to ?; A \to ?_v; A \to ?_c;\\
&\qquad A \to x \in \Pi; A \to x \in \Sigma;\\
&\qquad B \to A; B \to B, B;\\
&\qquad C \to A; C \to [B]; C \to \neg[B]\},\\
&S = \{C\}.
\end{aligned}$$

Symbol ? is a pattern for any item, symbol $?_v$ is a pattern for any variable and symbol $?_c$ is a pattern for any constant. A concrete variable is written as a single alphabetic character prefixed by symbol ?. All these terminal patterns are expressed by the non-terminal symbol A. A pattern collection B, where one of the inner patterns must match an item, is constructed inside square brackets, e.g, [<Prague>, <Berlin>]. The complement of this collection, where none of the inner patterns must match an item, is prefixed by symbol $\neg$, e.g, $\neg$[<Prague>, <Berlin>].

A rule pattern RP is an implication, where the right side contains just one head atom pattern and the left side consists of a conjunction of body atom patterns:

$$RP = AP_1 \wedge \ldots \wedge AP_n \Rightarrow AP_h.$$

Let item pattern IP be generated by the IPG grammar. The atom pattern AP is defined by the 4-tuple including four item patterns for subject, predicate, object, and graph:

$$AP_{n3} = (IP_s\ IP_p\ IP_o\ IP_g) \mid (IP_s\ IP_p\ IP_o).$$

If the last graph item pattern IP_g is not used, it may be omitted. Here is an example of a valid rule pattern and some matching rule:

(a rule pattern)

$$\begin{gathered}
(\text{?a <wasBornIn> ?b}) \wedge\\
(\text{?b } ?_c\ ?_c \text{ <DBpedia>}) \Rightarrow\\
(\text{?a [<livesIn>, <deadIn>] ?b}),
\end{gathered}$$

(a matching rule)

$$\begin{gathered}
(\text{?a <wasBornIn> ?b}) \wedge\\
(\text{?b dbo:isCityOf <USA> <DBpedia>}) \Rightarrow\\
(\text{?a <deadIn> ?b}).
\end{gathered}$$

During the mining process, rules are pruned based on all input rule patterns. A rule pattern RP is applied from right to left as well as the rule refinement process. For example, rules with $length = 1$ (with an empty body) are pruned if their heads do not match the head atom pattern. Subsequently, rules with $length = 2$ are pruned if their first atom from the right direction of the body does not match the first body atom pattern from the right direction.

5.5. Top-K Approach

In the top-k mode, the result set contains at most k rules with the highest value of a chosen measure. This mining strategy alleviates the user from searching for the right mining threshold, and can also substantially reduce the mining time.

Our proposal is a variation on the top-k approach introduced by Wang et al. [51] for mining top-k frequent itemsets with the highest support from transactional data. The main mining phase includes searching for rules reaching a support threshold derived from a given head coverage threshold. If the support threshold or the head coverage is unknown we may set a k value as the maximum number of rules to be returned. For this case, the mining algorithm saves all found appropriate rules into a priority queue with fixed length k, where the head of this queue (the head rule) is the rule with the lowest head coverage. Once the capacity of the queue is reached the minimum support threshold is set to the head coverage of the head rule. Then the following rules are pruned based on this support threshold. At the moment when some new rule has its head coverage greater than the minimum, the head rule is removed from the queue, and the new rule is added. Then the minimum support threshold is modified by a next head element in the queue (see Figure 4). The support threshold is continuously increasing during min-

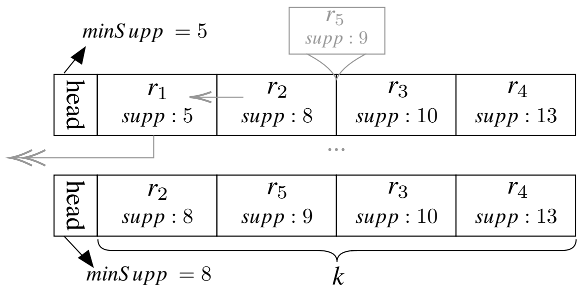


Figure 4. The top-*k* strategy using a priority queue with modification of the minimum support threshold.

ing and the result set always contains at most k rules with the highest head coverage.

The same strategy can be used for confidence calculation from a set of rules. Let k be the maximum length of the result rule set with highest confidences (standard or PCA). Once the capacity of the priority queue has been reached, the lowest confidence of the head rule is used as the minimum confidence threshold. This threshold is continuously increasing if the following rules have a higher confidence value.

Increasing the minimum confidence threshold *minConf* is important since it may speed up the confidence calculation. If the *minConf* value is set, then the following inequality must apply:

$$bsize(\vec{B} \Rightarrow H) \leqslant \frac{supp(\vec{B} \Rightarrow H)}{minConf}.$$

During the calculation of the body size, we can immediately stop the process as soon as the body size value is greater than the ratio between the rule support and *minConf*, because the rule is then guaranteed to have its confidence lower than *minConf*.

5.6. Rule Clustering

In order to cluster the rules, it is necessary to have some means of determining the similarity between an arbitrary pair of rules. The rule similarity can be computed from rule features, such as the content of the rule and the values of measures of significance.

Let the rules be represented with a matrix $R_{n\times m}$, where rows correspond to individual rules and columns to features of them. For each i-th feature, there is a similarity function $sim_i(\cdot,\cdot)$. The similarity between two rules, R_1 and R_2, is computed as:

$$sim(R_1, R_2) := \sum_{i=1}^{m} w_i \cdot sim_i(R_{1,i}, R_{2,i}),$$

where w_i is a weight of the feature i. The weights have to be normalized:

$$\sum_{i=1}^{m} w_i = 1.$$

It is straightforward to compare the measures of significance of two rules, e.g., the head coverage or the confidence values, as two numerical features. Nevertheless, in practice, for example within the *arules* library [9], rule clustering is mainly performed with respect to the rules content, in particular, using the similarity of atoms and their parts. Hence, for this purpose, we defined several similarity functions taking into account the content of the Horn rules.

Let an atom of a rule consist of a predicate p and of two atom items in the position of subject s and object o. These two atom items are either variables or constants. The similarity function between two atom items, s_1 and s_2 (or, analogously, o_1 and o_2) from the atoms $(s_1\ p_1\ o_1)$ and $(s_2\ p_2\ o_2)$, returns one of the three pre-defined values:

$$sim_t(s_1, s_2) = \begin{cases} 1 & \text{if } (s_1 = s_2 \wedge \nexists x \in \{s_1, s_2\} \\ & : IsVar(x)) \vee (p_1 = p_2 \wedge \\ & \forall x \in \{s_1, s_2\} : IsVar(x)), \\ 0.5 & \text{if } s_1 \neq s_2 \wedge p_1 = p_2 \ \wedge \\ & \exists x \exists y \in \{s_1, s_2\} \\ & : IsVar(x) \wedge \neg IsVar(y), \\ 0 & \text{otherwise,} \end{cases}$$

where $IsVar(x)$ is a function determining whether the atom item x is a variable. The result of the sim_t function also depends on the predicates p_1 and p_2 of the atoms in which s_1 and s_2 (o_1 and o_2, respectively) are contained. For instance, the similarity between the constant <Prague> and the variable ?b in atoms (?a <livesIn> <Prague>) and (?a <livesIn> ?b) is 0.5, since the items are not identical but the <Prague> constant can instantiate variable ?b.

The similarity function for two predicates only tests their equality.

$$sim_p(p_1, p_2) = \begin{cases} 1 & \text{if } p_1 = p_2, \\ 0 & \text{otherwise.} \end{cases}$$

For the atoms $a_1 = (s_1\ p_1\ o_1)$ and $a_2 = (s_2\ p_2\ o_2)$ we are able to compute the atom similarity based on their item similarities.

$$\begin{aligned} sim_a(a_1, a_2) = & \frac{1}{3}(sim_t(s_1, s_2) + sim_t(o_1, o_2) \\ & + sim_p(p_1, p_2)). \end{aligned}$$

```
Rule A is ranked higher than rule B
  1. if conf(A) > conf(B),
  2. if conf(A) = conf(B) and hc(A) > hc(B),
  3. if rule A has the shorter body (fewer atoms)
     than rule B.
```

Listing 1. Rule ranking criteria for rule pruning.

Suppose two rules U and V, where U has the length greater than or equal to the length of V, $|U| \geqslant |V|$. To calculate the content similarity of these rules, the rules $U = (u_1, \ldots, u_{|U|})$ and $V = (v_1, \ldots, v_{|V|})$ are regarded as sequences of atoms with the head atom and body atoms grouped together in any order. From all k-permutations of U, notated as $P(U, k)$, the maximum similarity is taken for $k = |V|$. The similarity of each permutation is computed as the sum of atom similarities between atoms from the permutation and atoms from V with the same index of the given sequences. The maximum similarity is normalized by the length of V:

$$sim_c(U, V) = \frac{1}{|V|} \max_{T \in P(U,|V|)} \sum_{i=0}^{|T|-1} sim_a(t_i, v_i).$$

While the generation of permutations (allowing to fit the two rules to one another) may look computationally demanding, for the rule lengths considered by RDFRules this overhead does not seriously impact the clustering performance, as witnessed by the experiments.

5.7. Rule Pruning

For selection of the most representative rules from the list of mined rules we propose to adapt *data coverage pruning*, which is a technique that is commonly used in association rule classification [54].

This technique, described in Algorithm 5, processes input rules in the order specified in Listing 1. For each rule, the algorithm checks whether the rule correctly predicts at least one triple in the input KG (line 10). If it does, the rule is kept and the triple is discarded (for the purpose of pruning). If the rule does not classify any (remaining) triple correctly, it is discarded.

In the list of rules mined by AMIE+, it is often the case that a single triple is correctly predicted by multiple rules. After data coverage pruning, many rules are removed, but it is still ensured that the reduced set of rules predicts the same set of triples as the original rule set (this is empirically evaluated in Section 7.7). Also, on transactional data, it has been empirically shown that the data coverage pruning (in combination with default rule pruning[19]) reduces the number of input rules by up to two magnitudes, while maintaining good classification performance. For example, experiments performed on 26 datasets, reported in [55], showed that, on average, 35.140 input rules were pruned into 69 final rules.

Algorithm 5. *Rule data coverage pruning*

```
1  function dataCoveragePruning(rules, KG)
2    KG' = KG
3    rules = sorted rules according to criteria in Listing~1
4    prunedRules = ()
5    for each $(\vec{B} \Rightarrow H) \in$ rules do
6      rulePredictsNewTriples = false
7      for each $\theta : \vec{B}\theta = (t\_1 \wedge \ldots \wedge t\_n) \wedge t\_1, \ldots, t\_n \in$ KG do
8        // where $\theta$ is the substitution defined in Sec. 3.3
9        $t\_h = H\theta$
10       if $t\_h \in$ KG' then
11         KG' = KG' $\setminus t\_h$
12         rulePredictsNewTriples = true
13       end if
14     end for
15     if rulePredictsNewTriples then prunedRules += $(\vec{B} \Rightarrow H)$
16   end for
17   return prunedRules
18 end function
```

6. RDFRules: Reference Implementation

While AMIE+ constitutes a breakthrough approach for rule learning from RDF data, the implementation accompanying the paper of Galárraga et al. [2] has several limitations in terms of practical usability compared to modern algorithmic frameworks for association rule learning from tabular datasets, such as the *arules* package for R [9], or Spark MLlib.[20]

In this section, we briefly describe a new framework for rule mining from RDF KGs called RDFRules, which is the reference implementation of the enhancements to AMIE+ described in the previous section, and is also used in our benchmarks (see Section 7).

[19]The well-known Classification Based on Associations (CBA) algorithm [55] essentially corresponds to data coverage pruning combined with 'default rule pruning'. In default rule pruning, during the pruning process, we keep track of which rule led to the smallest number of misclassifications, when all rules ranked lower than this rule are replaced by a default rule assigning the most frequent class among the remaining instances.

[20]https://spark.apache.org/mllib/

RDFRules is freely available under the GPLv3[21] open-source license and is hosted on GitHub[22].

6.1. Overview

The core of the reference RDFRules implementation is written in the Scala language. In addition to the Scala API, it also has a Java API, a RESTful service and a graphical user interface (GUI), which is available via a web browser. The Scala and Java APIs can be used as frameworks for extending another data mining system or application. The RESTful service is suitable for modular web-based applications and remote access. Finally, the GUI based on the RESTful service, can be used either as a standalone desktop application or as a web interface used to control the mining service deployed on a remote server. All modules are shown in Figure 5.

6.2. Architecture

The architecture of the RDFRules core is composed of four main data structures: *RDFGraph*, *RDFDataset*, *Index*, and *RuleSet*. These structures are created in the listed order during the RDF data pre-processing and rule mining. Inspired by Apache Spark, each structure supports several operations which either *transform* the current structure or perform some *action*.

Transformations The data structures are formed in the following order:

$$RDFGraph^{*} \rightarrow RDFDataset \rightarrow Index \rightarrow RuleSet$$

All the transformations are lazy[23] operations. A transformation converts the current data structure to a target data structure. The target data structure can be either of the same type or of the succeeding type. For example, a transformation of the *RDFDataset* structure creates either a new *RDFDataset* or an *Index* object.

Actions An action operation applies all pre-defined transformations on the current and previous structures and processes the (transformed) input data to create the desired output, such as rules, histograms, triples, statistics, etc. Compared to transformations, actions may load data into the memory and perform time-consuming operations. Actions are further divided into the *streaming* and *batch* ones. The streaming actions process data as small chunks (e.g., triples or rules) without large memory requirements, while the batch actions need to load all the data or a big part thereof into the memory.

Caching If several action operations are applied, e.g., with various input parameters, on the same data and with the same set of transformations, then all the defined transformations would normally be performed repeatedly for each action. This is caused by the lazy behavior of the data structures and the streaming process lacking the memory of previous steps. RDFRules eliminates those redundant and repeating calculations by caching the accomplished transformations. Each data structure has a *cache* method that can perform all the defined transformations immediately and store the result either into the memory or on the disk. The stored information can be reused when the already transformed data is to be further processed.

6.3. Graphs and Datasets

The *RDFGraph* structure is built once we load an RDF graph from either a file or a stream of triples or quads. For input data processing the RDFRules implementation uses modules from the Apache Jena[24] framework, which supports a range of RDF formats including N-Triples, N-Quads, JSON-LD, TriG or TriX. Besides these standard formats, RDFRules also has its own native binary format for faster serialization and deserialization its data structures (like rules, triples, and indices) on/from a disk for later use. During data loading, the system creates either one or multiple *RDFGraph* instances. Multiple instances are created when the input data format supports and uses named graphs.

An *RDFGraph* instance corresponds to a set of triples, on which applicable transformation operations are defined. These operations include *filtering* the triples using a condition, *replacement* of selected resources or literals, and *merging* numeric values using discretization algorithms. The transformed data may be *exported* to an RDF file. Several further operations focus on data exploration. These include the statements aggregation on histograms of triple items, the predicate ranges determination, and the triples counting.

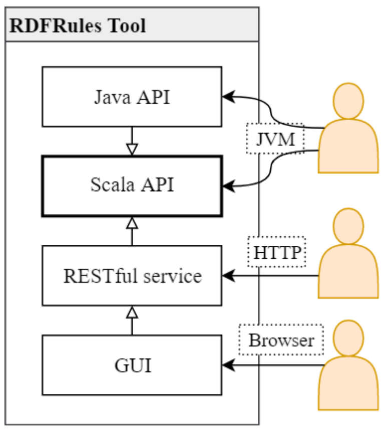
The *RDFDataset* structure is created from one or more *RDFGraph* instances. It is composed of quads

[21] https://www.gnu.org/licenses/gpl.txt

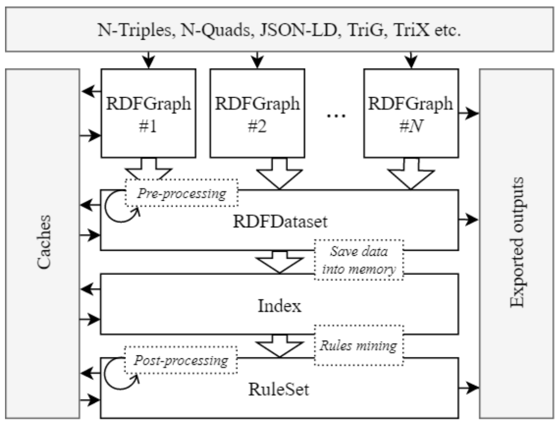
[22] https://github.com/propi/rdfrules

[23] A lazy transformation is not evaluated until a result of the transformation is required within an action.

[24] https://jena.apache.org/



(a) Available interfaces



(b) Main data structures and processes

Figure 5. Architecture of RDFRules.

where the triples are additionally associated with a particular named graph. This data structure supports transformation of all triples/quads within a dataset, as well as in the case of a single graph, with or without regard to the graphs assignment.

6.4. Indexing

Before the mining, the input dataset has to be indexed in the memory, which allows for the fast enumeration of atoms and computation of the measures of significance.

In the first phase of the indexing, each element of the triples, whether an identifier or a literal, is mapped into a unique number. This mapping is stored in a special hash map and eliminates any duplicates.

In the second phase of the indexing, the program only deals with the mapped numbers and creates the six fact indices described in Section 3.6. These indices are in one of the two modes: *preserved* or *in-use*. The *preserved* mode keeps the data in the memory for the whole duration of the index object, whereas the *in-use* mode only loads the data into the memory if the index is needed and after the use of the index, the memory is again released.

The *Index* instance can be created from the *RDFDataset* structure or loaded from the cache. The image of the fact indices can, therefore, be saved on the disk for further reuse. The *Index* structure contains the prepared data and has operations for rule mining using the AMIE+ algorithm.

Table 1

Available user-defined pruning thresholds used in the mining process in RDFRules.

MinHeadSize	Minimum number of triples instantiating the rule head
MinHeadCoverage	Minimum head coverage
MinSupport	Minimum absolute support
MaxRuleLength	Maximum length of a rule
TopK	Maximum number of returned rules sorted by head coverage
Timeout	Maximum mining time in minutes

6.5. Rule Mining

RDFRules implements the extensions to the AMIE+ rule mining algorithm covered in Section 5. Besides the indexed data itself, the values of three kinds of parameters also enter the mining phase in RDFRules; these parameters are (pruning) thresholds, rule patterns, and constraints.

Pruning Thresholds In addition to the thresholds defined in AMIE+, RDFRules also offers the top-k approach (see Section 5.5), and the timeout threshold, which determines the maximum mining time. All the mining thresholds are listed in table 1. Notice that the list of thresholds does not contain any confidence measures. These additional measures of significance can only be calculated after the main mining phase within the *RuleSet* structure.

Rule Patterns All the mined rules must match at least one pattern defined in the list of rule patterns. If the user has an idea of what kinds of atoms the mined rules should contain, this information can be defined through one or several rule patterns. The grammar of the rule pattern is described in Section 5.4. Since the process of matching the rules with patterns is performed during the mining phase, the enumeration of rules can be significantly sped-up.

We can define two types of rule pattern: *exact* and *partial*. The number of atoms in any mined rule must be the same as in the *exact* rule pattern. For a *partial* pattern, if some rule matches the pattern, all its refined extensions also match the pattern.

Constraints Finally, the last mining parameter specifies additional constraints, thus further shaping the output of the mining. Here is a list of the implemented constraints that can be used:

- *OnlyPredicates(x)*: the rules may only contain the predicates from the set *x*.
- *WithoutPredicates(x)*: the rules must not contain any of the predicates from the set *x*.
- *OnlyVariablesAt*: restricts to rules where constants are disabled at an atom position or all atom positions.
- *WithoutDuplicatePredicates*: disallows rules containing the same predicate in more than one atom.
- *GraphAwareRules*: the atoms in the discovered rules will be extended with information on their graph of origin.

Mining The mining process can be run in different behavior modes, with respect to the entry thresholds and constraints. Compared to the pure AMIE+ implementation, RDFRules does not calculate the confidence while browsing the search space of possible rules, thus saving time. Additionally, it applies various extensions described in Section 5. The rule mining process is performed in parallel (see Algorithm 1) and tries to use all available cores.

The mining result is an instance of the *RuleSet* structure which contains all the mined rules conforming to the input restrictions.

6.6. Rule Post-Processing

The *RuleSet* is the last defined data structure in the RDFRules workflow. It implements the operations for rule analysis, calculation of additional measures of significance, rule filtering and sorting, rule clustering and pruning, and finally, an export of the discovered rules for use in other systems. Every rule in the rule set consists of the head, the body, and the values of the measures of significance. The basic measures of significance are: *rule length*, *support*, *head size* and *head coverage*. Other measures may be calculated individually on user demand within the *RuleSet* structure. These measures include: *body size*, *confidence*, *PCA body size* and *PCA confidence*. The rules can be filtered and sorted according to all these measures.

Table 2
Feature comparison between the reference AMIE+ implementation and RDFRules

Mining phase	Reference AMIE+ implementation	RDFRules
Data exploration	Not supported	Histograms, statistics, graph filtering by conditions, search in triples
Pre-processing	Not supported	Equifrequent binning for numerical values
Mining	AMIE+ algorithm	AMIE+ algorithm with extensions (top-*k*, pattern language, performance improvements)
Post-processing	Not supported	Rule clustering, pruning, sorting and filtering by patterns
Rule export	Text	Text and JSON

RDFRules supports rule clustering with the DBScan algorithm [56], using similarity functions proposed in Section 5.6. The clustering process returns an assignment to a cluster for each rule based on input parameters including selected features, a minimum number of neighbors to create a cluster, and a minimum similarity value for two rules to be in the same cluster. The user can also opt to use similarity counting to determine the top-*k* most similar or dissimilar rules to a selected rule.

All the mined rules are stored in the memory, but, as in the case of the previous data structures, all transformations defined in the *RuleSet* are lazy. Therefore, this structure also allows to cache the rules and transformations on the disk or the memory for repeated usage. The complete rule set (or its subsets) can be exported and saved into a file in a human-readable text format or in a machine-readable JSON format.

6.7. Comparison to AMIE+ implementation

In the following, we will briefly compare the features of the AMIE+ implementation by AMIE+ authors[25] with our RDFRules framework. AMIE+ is a

[25]https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/amie/

Table 3
Used datasets in experiments.

	Triples	Subjects	Predicates	Objects
YAGO2 core	948K	470K	36	400K
YAGO3 sample	34K	14K	55	30K
DBpedia 3.8	11M	2.2M	650	1.5M
DBpedia literals sample	85K	13K	286	55.6K
DBpedia objects sample	121K	13K	325	45K

reference implementation for the algorithmic approach proposed in [2] and as such it focuses solely on the modeling phase, and does not offer any functionality supporting other phases of the data mining process. Table 2 provides a comparison between AMIE+ and RDFRules in terms of support the respective implementation provides for individual phases of a typical data mining task.

7. Experiments

We performed two kinds of experiments. Within the first group of experiments, we compare our proposed enhancements presented in Section 5 and implemented within our RDFRules framework with the original implementation of the AMIE+ algorithm. The second group of experiments is focused on evaluation of the newly proposed enhancements and algorithms. In particular, we evaluate discretization of numerical attributes proposed in Section 5.2, mining across multiple graphs (Section 5.3), rule patterns (Section 5.4), top-*k* approach (Section 5.5), clustering (Section 5.6), and rules pruning (Section 5.7).

7.1. Experimental Setup

For our experiments, we mainly used the YAGO2 core dataset and YAGO3 core samples of *yagoLiteralFacts*, *yagoFacts* and *yagoDBpediaInstances* available from the Max Planck Institute website[26]. For more time consuming tasks we used DBpedia 3.8 with person data and mapping-based properties. For other experiments we used samples of mapping-based literals and mapping-based objects. The number of triples and their unique elements for each dataset are shown in Table 3.

All the mining tasks are set with the minimum head size threshold 100 and maximum rule length 3. Unless explicitly mentioned, the number of used threads is set to 8. Each experiment is composed of various mining thresholds, constraints, patterns, the input dataset, and a selected framework (AMIE+ or RDFRules). Each experiment was executed 10 times. The experimental outcome consists of the average mining time together with the standard deviation and the number of found rules.

All experiments were launched on the scientific grid infrastructure of the CESNET MetaCentrum[27]. This grid architecture offers up to several hundred CPUs to be used per machine. For our purpose we used from 1 to 24 cores per experiment on a machine with these parameters:

- CPU: 4x 14-core Intel Xeon E7-4830 v4 (2GHz),
- RAM: 512 GB,
- OS: Debian 9.

A part of the implemented RDFRules framework is the *Experiments* module[28], within which all the experiments described below were conducted. Hence, all the reported experiments can be easily reproduced.

7.2. RDFRules vs AMIE+

In this section we compare our proposed and implemented enhancements of the AMIE+ algorithm with the original AMIE+ implementation.[29]

We performed two experiment types: 1) mining rules only with variables at the subject and object positions – and 2) mining rules with enabled constants. The mining process involves the computation of both types of confidence, i.e. standard confidence and PCA confidence. The results are reported for fixed standard and PCA confidence threshold 0.1 (MinConf+PCA) and a various minimum head coverage threshold (MinHC). RDFRules does not use the perfect rules pruning[30] designed in AMIE+; therefore, it is disabled on the AMIE+ side. The approximation of confidence, proposed in [2] to speed up computation of rule confidence, was disabled in our experiments, because this option can result in erroneous (incomplete) output as

[26]https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/downloads/

[27]https://metavo.metacentrum.cz/en/index.html

[28]https://github.com/propi/rdfrules/tree/master/experiments

[29]https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/amie/

[30]As an optional feature, AMIE+ stops refining rules as soon as confidence of the rule reaches 1.0.

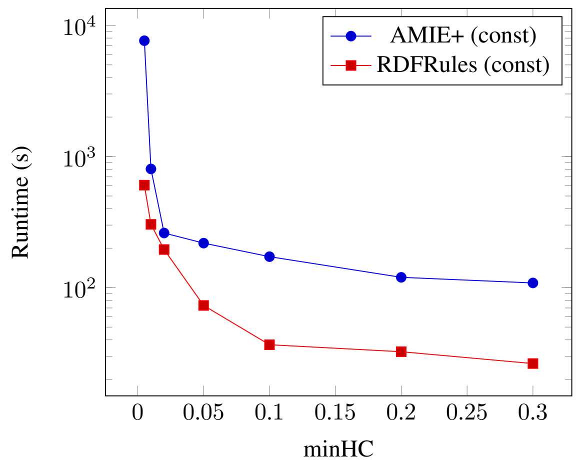


Figure 6. AMIE+ vs RDFRules: Runtime comparison for rule mining with constants for YAGO2 core.

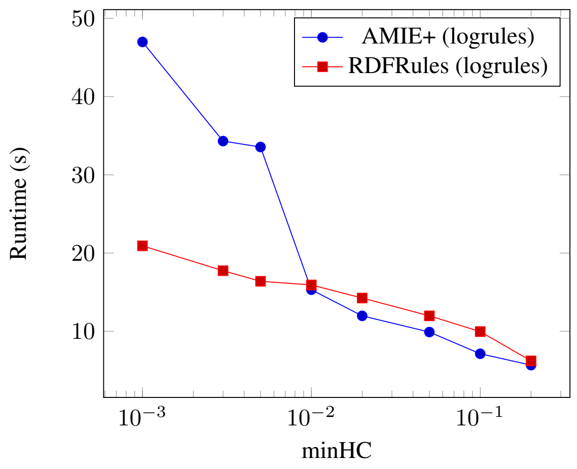


Figure 7. AMIE+ vs RDFRules: Runtime comparison for rule mining without constants for YAGO2 core.

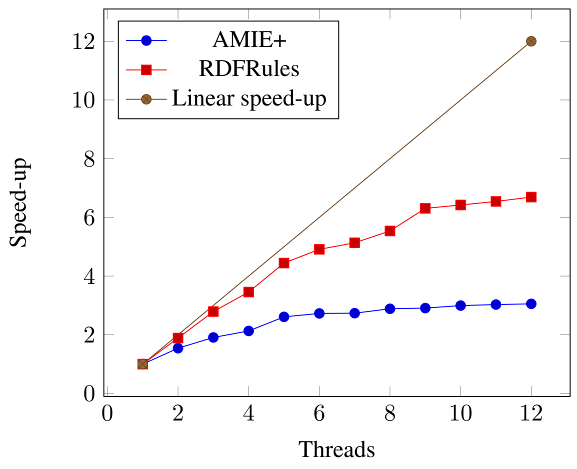


Figure 8. Scalability of RDFRules and AMIE+. Mining with constants and with *minHC* $= 0.01$.

some rules matching the minimum confidence threshold may be removed.

The number of output rules for RDFRules may be different from AMIE+. This is because AMIE+ does not return to the output the rules whose all parents have a higher confidence measure. A parent of the rule is a rule derived from the given rule by removing any one atom from its body. RDFRules has this functionality turned off by default; therefore, it can return more rules. This filtering is performed when the output rules are returned, and thus it does not affect the reported mining runtime.

The comparative experiments were launched for the YAGO2 core dataset and the DBpedia 3.8 dataset. Some selected tasks, their settings, and results are shown in Table 4. The *Diff* column contains the difference between the runtimes of AMIE+ and RDFRules. The *Rules* column contains the number of rules returned by AMIE+ and RDFRules.

7.2.1. RDFRules vs. AMIE+ on YAGO2 Core

Beside the Table 4 the results for the YAGO2 core dataset are also shown in Figure 7 for mining without constants, and in Figure 6 for mining with constants. We observed that for more difficult tasks with lower head coverage thresholds, which generate a larger set of rules, RDFRules is faster than the original AMIE+. On the contrary, for simpler tasks lasting several seconds and without constants, both approaches are almost at the same level of mining time.

Our performance improvements proposed in Section 5.1 have a considerable impact on mining with constants. For instance, the mining task with constants with *minHC* $= 0.005$ has a runtime of over 2 hours, whereas RDFRules only needs about 10 minutes to complete the same task.

The experiments also revealed that the original AMIE+ implementation assigns the individual jobs less efficiently into multiple threads when mining with constants. In our experiments, RDFRules used 99% of the CPU cores, whereas AMIE+ used only around 40%. To have a baseline, we also tried to mine rules with constants in a single thread. In this setting, RDFRules was still almost twice faster than AMIE+ (see the last row in Table 4). The degree to which the two systems scale when mining with constants is depicted in Figure 8.

Table 4

The mining runtime of AMIE+ and RDFRules with various settings.

Task	Cores	MinHC	MinConf +PCA	Runtime		Diff	Rules	
				AMIE+	RDFRules		AMIE+	RDFRules
YAGO2 core								
only with variables	8	0.005	0.1	54.8 s ± 2.49 s	18.72 s ± 1.04 s	-36.08 s	48	48
		0.01		15.32 s ± 222.1 ms	15.93 s ± 84.21 ms	+613.3 ms	31	31
		0.1		9.91 s ± 102.1 ms	11.98 s ± 56.4 ms	+2.07 s	10	10
with constants	8	0.005	0.1	2.12 h ± 3.96 min	10.08 min ± 13.38 s	-1.96 h	50,254	1,043,539
		0.01		13.40 min ± 9.22 s	5.06 min ± 11.56 s	-8.33 min	12,803	123,877
		0.1		2.87 min ± 5.76 s	36.7 s ± 1.63 s	-2.26 min	27	28
	1	0.01		27.94 min ± 15.38 s	15.47 min ± 12.51 s	-12.47 min	12,803	123,877
DBpedia 3.8								
only with variables	8	0.01	0.1	> 2 days	15 min			3,430
				with approx. 7.14 h		-6.89 h	2,884	
with constants				> 2 days	> 2 days, 27 min (lower-cardinality side of pred.)			386,534
		0.15		> 2 days	18.5 h			1,645,758

7.2.2. RDFRules vs. AMIE+ on DBpedia

For the DBpedia 3.8 dataset, we used default settings with minimum head coverage 0.01. The maximum mining time was set to two days. Each experiment was launched only twice with regard to more time consuming processes.

For AMIE+, the task of mining rules *only with variables* was not completed within two days. Hence, we tried to enable the confidence approximation where the task took around 7 hours, whereas RDFRules computed it without any approximation technique in 15 minutes.

The task of rule mining *with constants* was not finished within two days for both AMIE+ and RDFRules. The problem is the combinatorial explosion since the number of possible rules with constants is enormous for this task and dataset. Hence, we simplified the task for RDFRules to instantiate only variables at the lower-cardinality side of the predicate (this setting is not available within AMIE+). This task was successfully completed within 27 minutes and more than 380k rules were found (see Table 4). After increasing the minimum head coverage threshold to 0.15 the mining task was completed in 18.5 hours for RDFRules where more than 1.5M rules were found. For AMIE+ (also with the approximation technique) the task was not finished in 2 days.

7.3. Evaluation of the Top-k Approach and Rule Patterns

This section reports on results of experiments of enhancements specific to RDFRules: the top-k approach, confidence calculation, and rule patterns. For the top-k approach, described in Section 5.5, we launched the same tasks as in the previous set of experiments with the difference that the result set contained just the top 100 rules with the highest head coverage. We also tried to compare confidence computation with vs. without the top-k approach. Finally, we compare the mining time with search space constrained with rule patterns with the time required to mine all rules followed by subsequent filtering by a particular pattern.

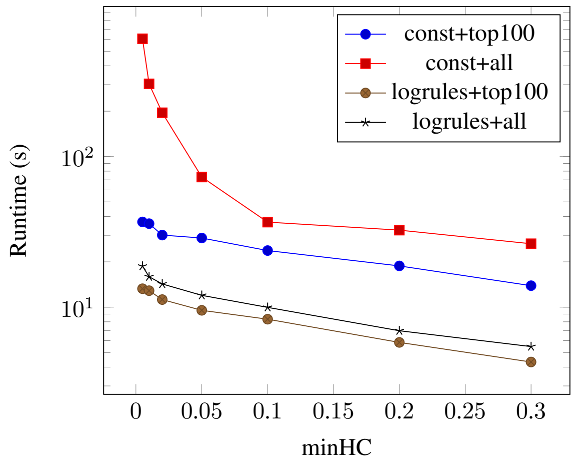


Figure 9. RDFRules rule mining, only with variables (logrules) and instantiated (const), with vs. without the top-k approach.

Figure 9 and Table 5 show how the top-k approach improves the performance of mining if only a subset of all rules with the highest head coverage is desired by the user.

Table 5
The mining runtime of the RDFRules top-k approach.

Task	MinHC	Runtime	
		Top-100	All
only with variables	0.005	13.26 s	18.72 s
	0.01	12.87 s	15.93 s
	0.1	8.320 s	11.98 s
with constants	0.005	36.80 s	10.08 min
	0.01	35.85 s	5.06 min
	0.1	23.76 s	36.7 s

Table 6
Runtimes of the confidence computation for 100,000 rules compared with the top-k approach. Settings are: $minConf = 0.1$ for both standard and PCA, the number of cores is set to 8.

Task	Runtime	
	Standard	PCA
All	1.1 s ± 249 ms	10.2 s ± 733.6 ms
Top-100	294 ms ± 4.8 ms	737 ms ± 141 ms

As described in Section 5.5, the top-k approach can also be used to speed-up confidence computation (standard or PCA). Table 6 contains results for confidence computation (both standard and PCA) of 100,000 rules. This is compared with searching for the top 100 rules with the highest confidence.

Another experiment was conducted to benchmark the mining with *partial* patterns, which were introduced in Section 5.4. We launched two tasks for two rule patterns. The first one emulates the situation when the user-specified the head of the rules to be discovered:

$$(?\ ?\ ?) \Rightarrow (?\ \text{<hasAcademicAdvisor>}\ ?). \tag{4}$$

The second one emulates the opposite case, when the user only specifies the body:

$$(?\ \text{<hasWonPrize>}\ ?) \Rightarrow (?\ ?\ ?). \tag{5}$$

Both tasks were launched with $minHC = 0.01$, 8 cores, without confidence counting, and with constants. Table 7 contains the mining time of both cases compared with the mining time without patterns, but with subsequent filtering of the desired rules by patterns. Since the rules are refined starting from the right side (the head) and ending at the leftmost atom in the body, mining with the pattern depicted in Eq. 5 is slower than with mining with the pattern depicted in Eq. 4. Listing 2 shows three examples of rules generated in these two experiments.

Table 7
Mining with and without rule patterns.

Task	Runtime	Rules
Mine all + filter by patterns	5.03 min ± 10.47 s	137,595
Mine by pattern in Eq. 4	216 ms ± 186 ms	13
Mine by pattern in Eq. 5	14.29 s ± 1.1 s	13

```
(?b <influences> ?a) ⇒ (?a <hasAcademicAdvisor> ?b)
(?b <hasWonPrize> ?c) ∧ (?a <hasWonPrize> ?c) ⇒ (?a
<hasAcademicAdvisor> ?b)
(?a <hasWonPrize> <Purple_Heart>) ⇒ (?a <hasWonPrize>
<Medal_of_Honor>)
```

Listing 2. An example of rules mined by RDFRules with patterns.

7.4. Evaluation of the Discretization of Numerical Attributes

We used the proposed discretization technique (described in Section 5.2.2) based on trees of intervals for the DBpedia literals sample dataset. The minimum head size threshold and the minimum head coverage threshold, required for the calculation of $MinSupp_L$ and $MinSupp_U$, were set as for the mining process ($MinHS = 100$ with a various head coverage). After the discretization step, the input dataset was enriched by new triples composed of constructed intervals. Here, we can observe a considerable number of newly generated triples.

The pre-processed input KG was then used in the mining process. First, we observed the number of mined rules for both variants of mining from the input KG: with vs. without the pre-processing.

Next, we evaluated the discretization on the task of *knowledge graph completion*, where we again compared the results generated with and without the discretization. The output rules were filtered by the minimum PCA confidence threshold set to 0.8. Consequently, all filtered rules were used to predict triples (explained in Section 3.3).

From the predicted triples T, we only took such triples $\langle s, p, o\rangle \in T'$ which had been missing in the KG and there was no other triple $\langle s, p, o'\rangle \in KG$:

$$T' = \{t \in T : t \notin KG \wedge \nexists \langle s, p, o'\rangle \in KG\}.$$

Based on this prediction, the input KG can be enriched with those missing triples whose prediction has the

Table 8

Data size of the DBpedia literals sample dataset before and after the discretization process using trees of intervals.

	# Predicates		# Triples	
Before discretization	All	Num.	85,075	
	286	89		
MinHC	All	New	# Triples	Runtime
0.01	475	189	149,438	1.3 s
0.02	439	153	151,333	726 ms
0.05	379	93	139,572	367 ms
0.1	349	63	130,423	323 ms
0.2	325	39	121,237	284 ms
0.3	314	28	118,708	252 ms

PCA confidence greater than or equal to 80%.[31] The input rules for prediction can be generated based on rules mined from unpreprocessed data or from data preprocessed with discretization. We compared the number of predicted triples under both scenarios (with vs. without discretization).

Table 8 summarizes the results from the various discretization processes. The header contains the number of all triples, all predicates, and all predicates that had a numerical range before the discretization process. The following rows show how many new triples and predicates were created after the discretization using the minimum head coverage threshold.

Table 9 shows that after the discretization, the system discovered many more rules (ca. 300 times more) and predicted many more new triples, even for higher thresholds, where originally no rules were found. For example, we predicted the following new triple, which is missing in the original KG:

⟨dbr:Owensboro_Kentucky,

dbo:populationTotal, "[0;499k)"⟩.

This triple was generated based on the following rule (with head coverage 0.2 and PCA confidence 0.81), which was only discovered after the discretization process:

(?a dbo:areaTotal "[440k;134M)") ⇒

(?a dbo:populationTotal "[0;499k)").

[31] The filtering of the output triples by the above formula assures that the Partial Completeness Assumption is preserved not only at the level of the rule (which has to satisfy the PCA confidence criterion as whole) but also at the level of the individual triples supplied to the KG.

The newly obtained fact is true, since the Owensboro city in Kentucky has around 60 thousand inhabitants.

It should be noted that the discretization does not only have benefits, but also extends the mining time due to the expansion of the hypothesis space. In the performed experiment, the mining time was up to six times longer.

7.5. Evaluation of Mining Across Multiple Graphs

RDFRules is able to merge multiple RDF KGs together, resolving their integration by the *owl:sameAs* predicate, which is described in Section 5.3. This linking is taken into account in the mining phase; therefore, discovered rules can contain atoms from different graphs.

We launched the rule mining process with minimum head coverage threshold 0.01 and enabled constants at the object position, separately for the YAGO3 sample and for the unified DBpedia sample (literals + objects). Subsequently, we merged the YAGO and DBpedia graphs using the *yagoDBpediaInstances* subset, containing the *owl:sameAs* linking, which is also part of the YAGO3 sample. Finally, we launched the rule mining process for the whole dataset consisting of these two different KGs.

The number of output rules from all tasks (YAGO, DBpedia, YAGO+DBpedia) is written in Table 10. For the merged YAGO+DBpedia dataset, a large number of new rules (998,240) containing atoms from both KGs were discovered. The mining time for YAGO+DBpedia is slightly longer (ca. 1.5x) than the sum of the mining times for each KG separately.

7.6. Evaluation of Clustering

The goal of clustering is generally to find cluster assignment minimizing the inter-cluster similarities and maximizing the intra-cluster similarities. For our experiment, the clustering was based on the rule content similarity function described in Section 5.6. The evaluation was focused on the quality of cluster assignments.

First, we needed a set of rules as an input for the clustering process. We used the YAGO2 core KG for mining the top 10,000 rules with constants (minimum head coverage threshold 0.01). Then, for the output rules, we launched clustering with the DBScan algorithm [56] with the minimum number of neighbours (region density *MinPts*) fixed to 1 for all experiments. The second element of the setting was the minimum

Table 9
Output comparison of results after mining with or without the discretization phase.

MinHC	# Rules		# Rules PCA $\geqslant 0.8$		# New triples		Runtime	
	Original	Discretized	Original	Discretized	Original	Discretized	Original	Discretized
0.01	938	283,263	584	81,961	965	2,329	5.970 s	22.78 s
0.02	147	71,105	56	15,188	878	2,166	2.055 s	12.14 s
0.05	20	10,348	7	1,924	689	1,373	1.399 s	6.251 s
0.1	6	1,981	2	327	35	185	1.110 s	3.766 s
0.2	1	283	0	44	0	49	903.4 ms	2.927 s
0.3	0	64	0	9	0	11	777.1 ms	1.749 s

Table 10
Rules and runtimes comparison after mining for merged and separated KGs.

Dataset	# Rules	Runtime
YAGO3 core sample	1,806,326	29.8 s
DBpedia sample (objects+literals)	88,926	29.38 s
YAGO+DBpedia	2,893,492	1.5 min

similarity threshold used as the inverse distance radius ϵ parameter of DBScan. For ϵ, we varied the threshold value.

Once the rules had been assigned to the clusters, it was possible to evaluate the quality of the assignment based on the overlap of clusters (inter-cluster similarity), and the similarities among rules inside a cluster (intra-cluster similarity). Let R_c be the set of rules in the cluster c. Then the clustering *quality index* (QI) is evaluated as the average difference between the intra-cluster similarity of any cluster c, expressed as Δ_{R_c}, and the maximum inter-cluster similarity between cluster c and another cluster, e.g. d, different from c, expressed as $\delta(R_c, R_d)$:

$$QI = \frac{1}{n}\sum_{i=1}^{n}\left(\Delta_{R_i} - \max_{j\neq i}(\delta(R_i, R_j))\right),$$

where n is the number of clusters.

The proposed QI is inspired by the average Silhouette Index [57]. The difference is that we take into account rule similarities instead of distances and the intra and inter cluster similarities are computed with the more straightforward way without having to calculate similarities among all rules across all clusters, which is very time consuming. The range of values of the QI is from -1 to 1 where -1 and 1 mean the worst and the best case of the clustering respectively.

In the rest of this section we will denote the subset of the input KG containing exactly those (distinct) triples that instantiate any atom from the body or from the head of a rule as the *matching subgraph* of this rule.

Intra-cluster Similarity Let $T_i \subseteq KG$ be the matching subgraph of rule $r_i \in R_c$, and $T = (T_1, \ldots, T_{|R_c|})$. Then the intra-cluster similarity for the ruleset R_c is calculated as the weighted average of the number of occurrences of individual triples in matching subgraphs in T:

$$\Delta_{R_c} = \frac{1}{\sum_{t\in\Theta}\omega_t}\sum_{t\in\Theta}\left(\frac{\omega_t}{|R_c|}\sum_{T_i\in T}[t\in T_i]\right),$$

where $[\cdot]$ denotes Iverson bracket notation[32], Θ is the set of all unions of sets in T:

$$\Theta = T_1 \cup \ldots \cup T_{|R_c|},$$

and ω_t is the weight of the triple $t \in \Theta$.

The weight is important since more general rules can match many more triples than more specific rules. For example, the following rule

r_1 : (?a <wasBornIn> ?b) $\Rightarrow$ (?a <diedIn> ?b)

matches more triples than

r_2 : (?a <wasBornIn> <Prague>)

$\Rightarrow$ (?a <diedIn> <Prague>).

The second rule r_2 is derived from the first rule r_1; therefore, the user would expect these two rules to be assigned to the same cluster. The problem is that many triples instantiating the atoms of r_1 do not instantiate the atoms of r_2. Hence, these triples have only one occurrence, namely in the set T_1, which can significantly decrease the intra-cluster similarity. To address this, the weight of a triple instantiating the more specific rule should be greater than the weight of a triple only instantiating the general rule. For this purpose, we defined weight ω_t for triple t as follows:

$$\omega_t = 1 + \max_{T_i\in T}|T_i| \cdot \sum_{T_i\in T}[t\in T_i] - \sum_{T_i\in T, t\in T_i}|T_i|.$$

[32] $[P] = 1$ if P is true, $[P] = 0$ if P is false.

Table 11

Results of clustering for 10,000 rules by the DBScan algorithm (*MinPts* $= 1$) and varying minimum similarity ϵ.

MinSim ϵ	# Clusters	QI	Avg intra	Avg inter	Runtime
0.3	4	0.561	0.561	0	6.9 min
0.4	6	0.621	0.635	0.005	7.2 min
0.5	7	0.639	0.648	0.005	7 min
0.6	15	0.672	0.827	0.014	7.4 min
0.7	35	0.478	0.882	0.028	6.6 min
0.8	38	0.414	0.861	0.037	4.7 min

Inter-cluster Similarity Let Θ_1 and Θ_2 be sets of triples instantiating the atoms of rules in clusters R_1 and R_2. Then the inter-cluster similarity is calculated as the number of intersections of Θ_1 and Θ_2 normalized by the size of the smallest set.

$$\delta(R_1, R_2) = \frac{|\Theta_1 \cap \Theta_2|}{\min(|\Theta_1|, |\Theta_2|)}.$$

Results For each clustering task, beside the QI, we calculated the average intra-cluster and inter-cluster similarity. All these measures, along with clustering runtimes and the number of created clusters, are shown in Table 11.

The table shows that the average intra-cluster similarity is much greater than the average inter-cluster similarity, which is desired. In our experiments, the highest value of the QI was obtained for *MinSim* $\epsilon = 0.6$. This setting generated 15 clusters. Furthermore, in this experiment, we demonstrated that rules clustered by their content similarity, proposed in Section 5.6, have small overlaps of matching subgraphs from different clusters and, conversely, higher overlaps of matching subgraphs within the same cluster.

7.7. Evaluation of Rule Pruning

In this section, we demonstrate the effectiveness of the pruning technique described in Section 5.7 on the YAGO2 core dataset. First, we mined top-*k* rules with different values of *k* and with the minimum standard confidence threshold set to 0.1. After that, we sorted rules by the confidence measure in descending order and applied the pruning step.

All results are summarized in Table 12. We compared the number of output rules with the number of rules after pruning. Notice that the number of rules after pruning increases logarithmically compared to the sizes of the set of rules before pruning. The table also shows that the number of all correctly predicted triples from the input KG by rules before and after pruning is still the same.

Table 12

The number of rules and correctly predicted triples, before and after pruning for YAGO2 core. The *Runtime* column shows the time spent by pruning.

Top-k	# Rules		# Triples		Runtime
	Before	After	Before	After	
500	350	89	16,824	16,824	478 ms
1,000	764	120	19,091	19,091	528 ms
2,000	1,604	166	19,252	19,252	691 ms
4,000	3,218	228	26,230	26,230	1.23 s
8,000	5,887	345	27,205	27,205	2.21 s
16,000	11,427	466	28,480	28,480	4.17 s
32,000	21,795	653	31,338	31,338	8.24 s

8. Conclusion

In this paper we have presented a set of extensions and enhancements to AMIE+, a state-of-the-art approach for mining Horn rules from RDF knowledge graphs. The primary aim of our work was to contribute to resolving the main challenge related to association rule learning (not only) from knowledge graphs – making it easy for the user to extract meaningful rules, without having to repeatedly change the mining parameters due to either a lack of results or a combinatorial explosion thereof. By giving the user the option to automatically group similar rules or to remove redundant rules, the result of rule learning remains explainable even when tens of thousands of rules are discovered. Other presented extensions allow for mining rules spanning multiple graphs, support for numerical data, and substantial performance enhancements in some special cases, such as when mining with constants or when mining with many CPU cores.

In a number of experiments, we have shown performance improvements of the individual extensions. More than an order of magnitude improvement compared to AMIE+ has been observed when mining for rare patterns, e.g., anomalies, which requires a very low head coverage threshold.

A reference implementation of the proposed approach is available as open source hosted on GitHub[33]. This software integrates several APIs (Java API, Scala API, and RESTful API) with a graphical user interface.

[33]https://github.com/propi/rdfrules

The project also contains the source code required for a replication of the benchmarks presented in this paper.

A promising direction for extending RDFRules is that of adding support for RDF schemas and ontologies, which would involve resource types with hierarchies into the mining process. Another challenging problem is that of matching the capabilities of the new generation of ILP systems, which, for example, support predicate invention and recursion [58]. We should also carry out comparative experiments with AMIE 3 as the very recently announced AMIE+ successor (thus being a 'close sibling' of RDFRules). Although the system currently supports multi-threading on a single machine, we would also like to add support for distributed mining and memory scaling on multiple nodes. Finally, RDFRules produces logical rules with a possibly complex structure, which may be found difficult to understand by some users. Thus, research into the human-perceived interpretability of logical rules is urgently needed. From the usability perspective, a valuable addition could be the addition of automatic tuning of mining thresholds, possibly by adapting one of the algorithms proposed for tabular data in [59]. In terms of applications, we consider investigating to what extent RDFRules can complement the recent generation of ILP systems such as Metagol [60] or MIGO [61] in the domain of learning game strategies. Specifically, we consider using RDFRules for learning an initial set of rules, leveraging the speed of its base association rule learning approach, and then refining these rules in the established ILP frameworks.

Acknowledgment

We are grateful to our three reviewers for detailed feedback, which resulted in many improvements of the article. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated. TK was supported by the Prague University of Economics and Business, Faculty of Informatics and Statistics, via the long-term institutional support of research activities, and by the IGA 43/2020 project. VZ and VS were partially supported by the CSF project no. 18-23964S.

References

[1] R. Agrawal, R. Srikant et al., Fast Algorithms for Mining Association Rules in Large Databases, in: *Proc. 20th int. conf. very large data bases, VLDB*, Vol. 1215, 1994, pp. 487–499.

[2] L. Galárraga, C. Teflioudi, K. Hose and F.M. Suchanek, Fast rule mining in ontological knowledge bases with AMIE+, *The VLDB Journal* **24**(6) (2015), 707–730, https://doi.org/10.1007/s00778-015-0394-1.

[3] D. Vrandečić, Wikidata: A new platform for collaborative data collection, in: *Proceedings of the 21st international conference on world wide web*, ACM, 2012, pp. 1063–1064, https://doi.org/10.1145/2187980.2188242.

[4] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak and S. Hellmann, DBpedia - A crystallization point for the Web of Data, *Web Semantics: science, services and agents on the world wide web* **7**(3) (2009), 154–165, https://doi.org/10.1016/j.websem.2009.07.002.

[5] T. Rebele, F. Suchanek, J. Hoffart, J. Biega, E. Kuzey and G. Weikum, YAGO: A Multilingual Knowledge Base from Wikipedia, Wordnet, and Geonames, in: *International Semantic Web Conference*, Springer, 2016, pp. 177–185, https://doi.org/10.1016/j.websem.2009.07.002.

[6] D. Beckett and B. McBride, RDF/XML syntax specification (revised), *W3C recommendation* **10**(2.3) (2004).

[7] P. Heim, S. Hellmann, J. Lehmann, S. Lohmann and T. Stegemann, RelFinder: Revealing Relationships in RDF Knowledge Bases, in: *International Conference on Semantic and Digital Media Technologies*, Springer, 2009, pp. 182–187, https://doi.org/10.1007/978-3-642-10543-2_21.

[8] M. Barati, Q. Bai and Q. Liu, Mining semantic association rules from RDF data, *Knowledge-Based Systems* **133** (2017), 183–196, https://doi.org/10.1016/j.knosys.2017.07.009.

[9] M. Hahsler, S. Chelluboina, K. Hornik and C. Buchta, The arules R-Package Ecosystem: Analyzing Interesting Patterns from Large Transaction Data Sets, *Journal of Machine Learning Research* **12**(Jun) (2011), 2021–2025.

[10] V. Zeman, T. Kliegr and V. Svátek, RdfRules Preview: Towards an Analytics Engine for Rule Mining in RDF Knowledge Graphs, in: *RuleML Challenge, 2018*, https://doi.org/10.29007/nkv7.

[11] J. Li, H. Shen and R. Topor, Mining Optimal Class Association Rule Set, *Knowledge-Based Systems* **15**(7) (2002), 399–405, https://doi.org/10.1007/3-540-45357-1_39.

[12] B. Goethals and J. Van den Bussche, Relational Association Rules: Getting Warmer, in: *Pattern Detection and Discovery*, Springer, 2002, pp. 125–139, https://doi.org/10.1007/3-540-45728-3_10.

[13] J. Józefowska, A. Ławrynowicz and T. Łukaszewski, The role of semantics in mining frequent patterns from knowledge bases in description logics with rules, *Theory and Practice of Logic Programming* **10**(3) (2010), 251–289, https://doi.org/10.1017/S1471068410000098.

[14] A. Ławrynowicz and J. Potoniec, Fr-ONT: An Algorithm for Frequent Concept Mining with Formal Ontologies, in: *International Symposium on Methodologies for Intelligent Systems*, Springer, 2011, pp. 428–437, https://doi.org/10.1007/978-3-642-21916-0_46.

[15] A. Ławrynowicz and J. Potoniec, Pattern Based Feature Construction in Semantic Data Mining, *International Journal on Semantic Web and Information Systems (IJSWIS)* **10**(1) (2014), 27–65, http://doi.org/10.4018/ijswis.2014010102.

[16] M. Morzy, A. Ławrynowicz and M. Zozuliński, Using Substitutive Itemset Mining Framework for Finding Synonymous Properties in Linked Data, in: *International Symposium on Rules and Rule Markup Languages for the Semantic Web*, Springer, 2015, pp. 422–430, https://doi.org/10.1007/978-3-319-21542-6_27.

[17] J. Potoniec, P. Jakubowski and A. Lawrynowicz, Swift Linked Data Miner: Mining OWL 2 EL class expressions directly from online RDF datasets, *Journal of Web Semantics First Look* (2017), https://doi.org/10.1016/j.websem.2017.08.001.

[18] H. Paulheim and C. Bizer, Type Inference on Noisy RDF Data, in: *International semantic web conference*, Springer, 2013, pp. 510–525, https://doi.org/10.1007/978-3-642-41335-3_32.

[19] C. Meilicke, M.W. Chekol, D. Ruffinelli and H. Stuckenschmidt, Anytime Bottom-Up Rule Learning for Knowledge Graph Completion (2019), 3137–3143, https://doi.org/10.24963/ijcai.2019/435.

[20] M. Svatoš, S. Schockaert, J. Davis and O. Kuzelka, STRiKE: Rule-Driven Relational Learning Using Stratified k-Entailment, in: *ECAI 2020: 24th European Conference on Artificial Intelligence*, 2020.

[21] R. Agrawal, T. Imieliński and A. Swami, Mining association rules between sets of items in large databases, in: *ACM SIGMOD record*, Vol. 22, ACM, 1993, pp. 207–216, https://doi.org/10.1145/170036.170072.

[22] J. Lehmann, G. Sejdiu, L. Bühmann, P. Westphal, C. Stadler, I. Ermilov, S. Bin, N. Chakraborty, M. Saleem, A.-C.N. Ngomo et al., Distributed Semantic Analytics Using the SANSA Stack, in: *International Semantic Web Conference*, Springer, 2017, pp. 147–155, https://doi.org/10.1007/978-3-319-68204-4_15.

[23] J. Rabatel, M. Croitoru, D. Ienco and P. Poncelet, Contextual Itemset Mining in DBpedia, in: *LD4KD: Linked Data for Knowledge Discovery*, Vol. 1232, CEUR, 2014, p. http–ceur.

[24] J. Kim, E.-K. Kim, Y. Won, S. Nam and K.-S. Choi, The Association Rule Mining System for Acquiring Knowledge of DBpedia from Wikipedia Categories., in: *NLP-DBPEDIA@ISWC*, 2015, pp. 68–80.

[25] V. Nebot and R. Berlanga, Finding association rules in semantic web data, *Knowledge-Based Systems* **25**(1) (2012), 51–62, https://doi.org/10.1016/j.knosys.2011.05.009.

[26] C. d'Amato, A.G. Tettamanzi and T.D. Minh, Evolutionary Discovery of Multi-relational Association Rules from Ontological Knowledge Bases, in: *European knowledge acquisition workshop*, Springer, 2016, pp. 113–128, https://doi.org/10.1007/978-3-319-49004-5_8.

[27] C. d'Amato, S. Staab, A.G. Tettamanzi, T.D. Minh and F. Gandon, Ontology Enrichment by Discovering Multi-Relational Association Rules from Ontological Knowledge Bases, in: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 333–338, http://doi.org/10.1145/2851613.2851842.

[28] M. Nickel, V. Tresp and H.-P. Kriegel, A Three-Way Model for Collective Learning on Multi-Relational Data, in: *ICML*, Vol. 11, 2011, pp. 809–816.

[29] M. Nickel, L. Rosasco and T. Poggio, Holographic embeddings of knowledge graphs, in: *Thirtieth AAAI conference on artificial intelligence*, 2016.

[30] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston and O. Yakhnenko, Translating embeddings for modeling multi-relational data, in: *Advances in neural information processing systems*, 2013, pp. 2787–2795.

[31] P.G. Omran, K. Wang and Z. Wang, Scalable Rule Learning via Learning Representation, in: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, AAAI Press, 2018, pp. 2149–2155. ISBN 978-0-9992411-2-7.

[32] V.T. Ho, D. Stepanova, M.H. Gad-Elrab, E. Kharlamov and G. Weikum, Rule Learning from Knowledge Graphs Guided by Embedding Models, in: *International Semantic Web Conference*, Springer, 2018, pp. 72–90, https://doi.org/10.1007/978-3-030-00671-6_5.

[33] H. Xiao, M. Huang, L. Meng and X. Zhu, SSP: Semantic Space Projection for Knowledge Graph Embedding with Text Descriptions, in: *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[34] W. Zhang, B. Paudel, L. Wang, J. Chen, H. Zhu, W. Zhang, A. Bernstein and H. Chen, Iteratively Learning Embeddings and Rules for Knowledge Graph Reasoning, in: *The World Wide Web Conference*, ACM, 2019, pp. 2366–2377, https://doi.org/10.1145/3308558.3313612.

[35] Z. Yin and Y. Shen, On the Dimensionality of Word Embedding, in: *Advances in Neural Information Processing Systems*, 2018, pp. 887–898.

[36] C. Meilicke, M. Fink, Y. Wang, D. Ruffinelli, R. Gemulla and H. Stuckenschmidt, Fine-Grained Evaluation of Rule and Embedding-Based Systems for Knowledge Graph Completion, in: *International Semantic Web Conference*, Springer, 2018, pp. 3–20, https://doi.org/10.1007/978-3-030-00671-6_1.

[37] R. Das, A. Godbole, N. Monath, M. Zaheer and A. McCallum, Probabilistic Case-based Reasoning for Open-World Knowledge Graph Completion, *arXiv preprint arXiv:2010.03548* (2020).

[38] A.A. Freitas, Automated Machine Learning for Studying the Trade-Off Between Predictive Accuracy and Interpretability, in: *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, Springer, 2019, pp. 48–66, https://doi.org/10.1007/978-3-030-29726-8_4.

[39] M. Fernández-Delgado, E. Cernadas, S. Barro and D. Amorim, Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?, *Journal of Machine Learning Research* **15**(1) (2014), 3133–3181.

[40] J. Fürnkranz, T. Kliegr and H. Paulheim, On Cognitive Preferences and the Plausibility of Rule-based Models, *Machine Learning* **109**(4) (2020), 853–898, https://doi.org/10.1007/s10994-019-05856-5.

[41] T. Kliegr, Š. Bahník and J. Fürnkranz, A review of possible effects of cognitive biases on interpretation of rule-based machine learning models, *arXiv preprint arXiv:1804.02969* (2018).

[42] J. Lajus, L. Galárraga and F. Suchanek, Fast and Exact Rule Mining with AMIE 3, in: *European Semantic Web Conference*, Springer, 2020, pp. 36–52, https://doi.org/10.1007/978-3-030-49461-2_3.

[43] S. Ceri, G. Gottlob and L. Tanca, *Logic Programming and Databases*, Springer Science & Business Media, 2012,

https://doi.org/10.1007/978-3-642-83952-8. ISBN 978-3-642-83952-8.

[44] M. Kifer and H. Boley, RIF Overview, 2010. http://www.w3.org/TR/rif-overview/.

[45] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosofand and M. Dean, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, 2004. http://www.w3.org/Submission/SWRL/.

[46] L.A. Galárraga, N. Preda and F.M. Suchanek, Mining rules to align knowledge bases, in: *Proceedings of the 2013 workshop on Automated knowledge base construction*, ACM, 2013, pp. 43–48, https://doi.org/10.1145/2509558.2509566.

[47] T. Kliegr and J. Kuchař, Tuning Hyperparameters of Classification Based on Associations (CBA), in: *Proceedings of ITAT 2019*, CEUR-WS, 2019.

[48] A.V. Carreiro, A.J. Ferreira, M.A. Figueiredo and S.C. Madeira, Prognostic Prediction Using Clinical Expression Time Series: Towards a Supervised Learning Approach Based on Meta-biclusters, in: *6th International Conference on Practical Applications of Computational Biology & Bioinformatics*, Springer, 2012, pp. 11–20, https://doi.org/10.1007/978-3-642-28839-5_2.

[49] G.I. Webb, Filtered-top-k association discovery, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **1**(3) (2011), 183–192, https://doi.org/10.1002/widm.28.

[50] L. Bustio-Martínez, M. Letras-Luna, R. Cumplido, R. Hernández-León, C. Feregrino-Uribe and J.M. Bande-Serrano, Using hashing and lexicographic order for Frequent Itemsets Mining on data streams, *Journal of Parallel and Distributed Computing* **125** (2019), 58–71, https://doi.org/10.1016/j.jpdc.2018.11.002.

[51] J. Wang, J. Han, Y. Lu and P. Tzvetkov, TFP: An efficient algorithm for mining top-k frequent closed itemsets, *IEEE Transactions on Knowledge and Data Engineering* **17**(5) (2005), 652–663, https://doi.org/10.1109/TKDE.2005.81.

[52] J. Fürnkranz, D. Gamberger and N. Lavrač, *Foundations of Rule Learning*, Springer Science & Business Media, 2012, https://doi.org/10.1007/978-3-540-75197-7. ISBN 978-3-540-75196-0.

[53] P. Hájek, M. Holeňa and J. Rauch, The GUHA method and its meaning for data mining, *Journal of Computer and System Sciences* **76**(1) (2010), 34–48, https://doi.org/10.1016/j.jcss.2009.05.004.

[54] K. Vanhoof and B. Depaire, Structure of association rule classifiers: a review, in: *2010 IEEE International Conference on Intelligent Systems and Knowledge Engineering*, IEEE, 2010, pp. 9–12, https://doi.org/10.1109/ISKE.2010.5680784.

[55] B. Liu, W. Hsu and Y. Ma, Integrating Classification and Association Rule Mining, in: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD'98, AAAI Press, 1998, pp. 80–86.

[56] M. Ester, H.-P. Kriegel, J. Sander, X. Xu et al., A density-based algorithm for discovering clusters in large spatial databases with noise, in: *KDD*, Vol. 96, 1996, pp. 226–231.

[57] P.J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, *Journal of computational and applied mathematics* **20** (1987), 53–65, https://doi.org/10.1016/0377-0427(87)90125-7.

[58] A. Cropper, S. Dumančić and S.H. Muggleton, Turning 30: New Ideas in Inductive Logic Programming, *arXiv preprint arXiv:2002.11002* (2020), https://doi.org/10.24963/ijcai.2020/673.

[59] T. Kliegr and J. Kuchar, Tuning Hyperparameters of Classification Based on Associations (CBA)., in: *Proceedings of ITAT*, 2019.

[60] A. Cropper and S.H. Muggleton, Learning Higher-Order Logic Programs through Abstraction and Invention, in: *IJCAI*, 2016, pp. 1418–1424.

[61] C. Hocquette, Can Meta-Interpretive Learning outperform Deep Reinforcement Learning of Evaluable Game strategies? (2019), 6440–6441, https://doi.org/10.24963/ijcai.2019/909.