RDFRules: Making RDF Rule Mining Easier and Even More Efficient

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

Abstract. AMIE+ (Galárraga et al., 2015) 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 more practical to use. The main contributions are related to performance improvement: the top-*k* approach, which addresses the problem of combinatorial explosion often resulting from a hand-set minimum support threshold, a grammar that allows to define fine-grained patterns reducing the size of the search space, and the faster projection binding reducing the number of repetitive calculations. Other enhancements include the possibility to mine across multiple graphs, lift as a new rule interest measure adapted to RDF KGs, the support for discretization of continuous values, and the selection of the most representative rules using proven rule pruning and clustering algorithms. Benchmarks show considerable improvements compared to AMIE+ on some problems. An open-source reference implementation is available under the name 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 modern 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 knowledge graphs, 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 KGs operate under the *Open World Assumption* (OWA). It means that the KG is regarded as potentially 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[1] 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

*Corresponding author. E-mail: vaclav.zeman@vse.cz.

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

right side, called head, and conjunction of atoms on the left side, called body:

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

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

(?a <hasChild> ?c) ∧ (?b <hasChild> ?c)

⇒ (?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.,

(?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 a precise user-set support threshold, fine-grained search space restrictions, avoidance of some repetitive calculations, and the ability to process numerical attributes.

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.

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 provided 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 presented approach has a more than an order of magnitude shorter mining time than AMIE+. 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.

The main contributions of this paper are:

- A review of recent developments in RDF rule mining, with a focus on AMIE+ and its limitations.
- A collection of optimisation steps that improve performance of AMIE+, while providing the same results, and a collection of pre-processing and post-processing algorithms for rule learning from RDF.
- A software framework including a web-based GUI and an API designed to cover all steps of rule learning from RDF data.
- Experiments evaluating the proposed algorithms, showing considerable improvements over AMIE+.

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, under the title "RdfRules Preview: Towards an Analytics Engine for Rule Mining in RDF Knowledge Graphs" [10].

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

2. Related Work

Approaches applicable for rule learning from RDF-style 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 downward closure property has been introduced to prune the space of hypotheses.

Inductive Logical Programming Algorithms based on the principles of ILP learn Horn rules on binary predicates. Examples of applicable ILP approaches include ALEPH[3] or WARMeR [11]. 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 semantic web 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 [12].
- ILP systems have been reported 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.

Association Rule Mining Some algorithms for rule mining from RDF KGs use adaptations of the seminal *apriori* algorithm [13] 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:

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

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

The AMIE+ algorithm uses the downward 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 [14] and is also used for rule learning in SANSA-Stack [15], which is a general-purpose toolbox for distributed data processing of large-scale RDF KGs based on Apache Spark.

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:

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

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 [16] or Wikipedia categories [17]. Nebot and Berlanga [18], 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.

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 [19], HolE [20] or TransE [21], 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 [22] 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

[3] http://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph_toc.html

on learning rules for a specific predicate and cannot discover rules with constants.

Another rule mining system using embeddings is RuLES[4] [23]. 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 [24].

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 [25]. 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 [25, 26]. 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 completion indicated superior results of AMIE+ in terms of precision compared to multiple algorithms based on embeddings, including TransE, HolE, or RESCAL [27], on some completion tasks.

3. A Review of AMIE+ of Galárraga et al, 2015

The following paragraphs describe the basic features of the AMIE+ algorithm on which our approach builds.

3.1. Specifics of RDF Knowledge Graph Mining

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 described by an IRI[5] 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.

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

[5] Internationalized Resource Identifier

3.2. Rules

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

3.2.1. Atom

An AMIE+ atom is a statement which is further indivisible and contains a *constant* at the predicate position and at least one *variable* at the subject and/or object position. E.g., 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.

Notation The AMIE+ literature describes rules using the Datalog notation [28] 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.[6] The same rule is then written as

(?a <wasBornIn> ?b) ⇒ (?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 (ground fact) is enclosed in angle brackets to be distinguished from non-ground atoms, e.g.,

⟨<John>, <livesIn>, <Prague>⟩.

Coverage Let the symbol $\prec$ denote the coverage of a triple (or a conjunction of triples) by an atom (or a conjunction of atoms). For example:

⟨<John>, <livesIn>, <Prague>⟩ ≺ (?a <livesIn> ?b).

A triple t is covered by an atom A 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 $A\theta$ on the atom A produces the triple t. For instance, the particular substitution and its application of the previous coverage example are:

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

(?a <livesIn> ?b)θ = ⟨<John>, <livesIn>, <Prague>⟩

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

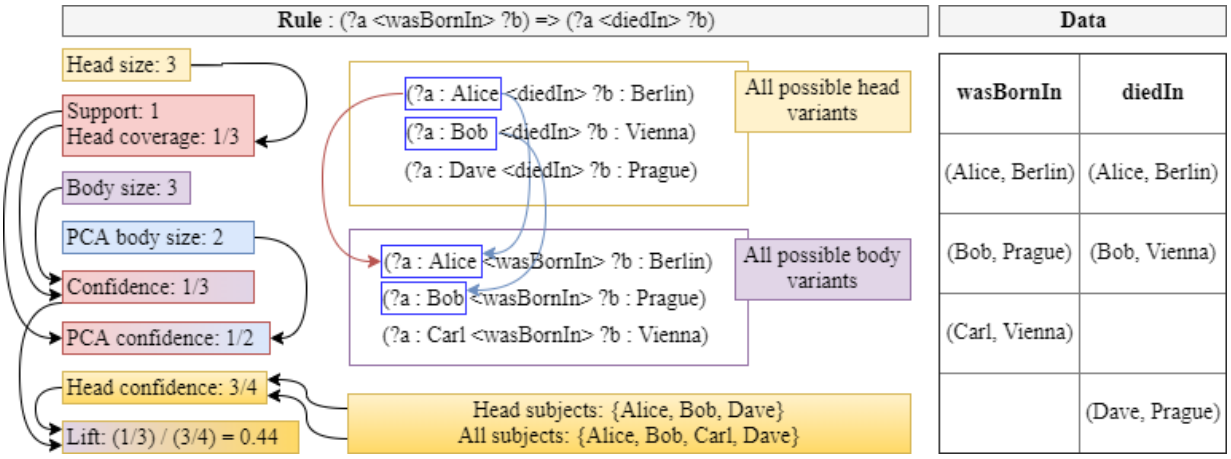


Figure 1. Computation of measures of significance in AMIE+ with the additional lift measure.

The application of a substitution can also be used on a conjunction of atoms to obtain a conjunction of triples.

3.2.2. *Form 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) ∧ (?c <directed> ?b)

⇒ (?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) ⇒ (?a <wasBornIn> ?b),

(an open rule)

(?a <livesIn> ?c) ⇒ (?a <wasBornIn> ?b).

Finally, the rule atoms must not be reflexive: an atom cannot contain the same variable at both object and subject positions.

3.3. *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 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.

3.3.1. *Atom Size*

One of the key functions used by AMIE+ is $size(A)$, which counts the number of triples from the KG covered by 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 is three.

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

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

3.3.2. *Support and Head Coverage*

Support is the 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 support has the *anti-monotone* property [29] (also referred to as downward closure, or apriori property). 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*. Hence, a whole branch of infrequent extensions can be skipped.

In AMIE+, the support is defined as the number of correctly predicted triples covered by the head atom H given at least one conjunction of triples covered by the body $\vec{B}$:

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

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

⟨<Alice>, <diedIn>, <Berlin>⟩,

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

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:

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

3.3.3. Confidence

The support of a rule only indicates the number of correct predictions, 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 the support and the *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 all distinct subject-object pairs derived from the variables in the body that also occur in the head:

$$bsize(\vec{B} \Rightarrow H) = \#\langle s,o\rangle : \exists\{?a/s, ?b/o\} \subseteq \theta \wedge \vec{B}\theta = (t_1 \wedge \cdots \wedge t_n) \wedge t_1,\ldots,t_n \in KG \wedge H = (?a\ p\ ?b)$$

For each pair $\langle s,o\rangle$ at least one conjunction of triples $(t_1 \wedge \cdots \wedge t_n)$ covered by $\vec{B}$ must exist. This conjunction contains constants s and o substituted from variables ?a and ?b. These variables have to be included in body $\vec{B}$ and also in head H.

The standard confidence operates under the CWA. That means if some statement is missing, then it can be considered a negative example. 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 PCA confidence. If some predicate p is completely absent for some subject s then it is not considered, unlike in the standard confidence, a negative example. For instance, if the subject s does not appear with the <isMarriedTo> predicate in any triple, s may not necessarily be unmarried. However, once we know of a triple with subject s and predicate p, we can assume that the KG involves *all* facts associated with this subject and predicate. As a consequence, all other missing statements related to this predicate and subject are regarded as negative examples:

$$bsize_{pca}(\vec{B} \Rightarrow H) = \#\langle s,o\rangle : \exists\{?a/s, ?b/o\} \subseteq \theta \wedge ((?a\ p\ o') \wedge \vec{B})\theta = (t_1 \wedge \cdots \wedge t_n) \wedge t_1,\ldots,t_n \in KG \wedge H = (?a\ p\ ?b)$$

where o' is a dangling variable which is not connected to the body.

For example, in Figure 1, while there are three triples covered by the body (the body size is three), only for two subjects of these triples, there exists at least one triple containing the head predicate <diedIn>. About the third subject *Carl* we have no record relating to his death; therefore, it cannot be considered a negative example and the PCA body size is two.

3.4. 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 pre-set 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 indexed KG, from 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)

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

(closing atom added)

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

(dangling atom added)

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

(instantiated atom added)

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

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 p : ⟨ s, p, o ⟩ ∈ KG do
5      if size((?a p ?b)) ⩾ minHS then
6        queue.enqueue(∅ ⇒ (?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) ⩾ minHC ∧ newRule ∉ queue then
18           queue.enqueue(newRule)
19         end if
20       end for
21     end if
22   end while
23   return out
24 end function
```

3.5. Index

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 indexes*: 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 a subject s points to a subset of predicates P where each predicate $p \in P$ points to a subset of objects O:

$$s \in S \mapsto \{p \in P' \mapsto O' : P' \subseteq P, O' \subseteq O\}.$$

Based on this structure, the data have to be replicated six times and still have to fit into the main memory.

4. Limitations of AMIE+

As noted earlier, AMIE [30] 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. 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 preprocessing 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 [31].

An integrated approach is even more necessary in the linked data context as general algorithms for data pre-processing are difficult to apply to linked datasets due to the different structure of inputs as well as outputs. Also, in the tabular or transactional context, it is typically computationally feasible to pre-process all available data. However, for linked data such an indiscriminative 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 linked datasets 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 consist of the lack of various features which were found useful for mining rules from transactional or tabular data, such as the support for additional interest measures or the selection of most representative rules, but are missing in AMIE+, or 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 *anti-monotone* property, which requires that not only the complete rule but also each subset of atoms composing it meets the minimum support threshold. Since numeric attributes have typically many values, a single distinct value may not have 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 numeric value at the object position related to a specific contract, e.g.,

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

If every contract has just one price, which is unique in the dataset, the instantiated atom (?a <hasPrice> C), where C is a numeric constant, has the size of at most one. In AMIE+, the user sets the minimum size of the head atom – Galárraga et al. [2] suggest 100 as the default value. For this threshold, the above-mentioned instantiated atom 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 instantiated atom is in the body, e.g.,

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

(?a <authority> <MinistryOfDefense>),

the atom size must be greater than or equal to the minimum support threshold depending on the rule head size and the minimum head coverage threshold (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 by one broader nominal value. This process is called *discretization*, *binning* or *quantization*.

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 [32, 33]. 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 situa-

tion. 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 in the linked data context, where data may be very large and not known beforehand.

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 [32, 34].

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[7] 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 containing only rules that contain a triple with a specific value in the antecedent, such as:

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

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+.

4.4. Repetitive Calculations

During the refinement process, AMIE+ binds variables to constants in order to count the support for each fresh[8] 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:

(?a <livesIn> ?b) ⇒ (?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. It slows down the overall mining process.

4.5. Exhaustive Calculations

AMIE+ first computes the value of support for each refined rule, and only afterward, 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 paths covered by 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. Limited Choice of Measures of Significance

While minimum support and confidence thresholds are an integral part of association rule mining, later

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

[8] "Fresh" is a term used in [2] to denote a newly added atom.

research has shown the utility of involving thresholds on a range of additional measures of significance into association rule learning algorithms [35]. Out of the dozens of proposed measures, the one most frequently adopted seems to be the *lift* measure. Lift is supported in most association rule learning systems, including the open source (R language) *arules* package, but it is not included in AMIE+.

4.7. Lack of Support for Multiple Graphs

In semantic web terms, a KG identified by a specific IRI is called a *named graph*. 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 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, <dbpedia>⟩,

⟨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⟩.

4.8. 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* [36]. While pruning may not be applicable for explorative association rule learning[9], 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 [38].

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 knowledge base. 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.

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 $(x\ ?r\ y) \in A_n$ 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) recursively 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).

[9] E.g., the motto of the GUHA rule learning framework [37] is "GUHA offers everything interesting", which translates as "all hypotheses of the given form true in the data".

Algorithm 2. *The AMIE+ count projection query*

```
1  function countProjections(A_n, B⃗ ⇒ H, k, KG)
2    out = {}
3    for all (x ?r y) ∈ A_n do
4      map = {}
5      q = B⃗ + H + (x ?r y)
6      for all ⟨s, p, o⟩ ≺ H : ⟨s, p, o⟩ ∈ KG do
7        q' = bind(q, ⟨s, p, o⟩, H)
8        χ = select(?r, q')
9        for all r ∈ χ do
10         if map[r] = ∅ then map[r] = 1 else map[r]++
11       end for
12     end for
13     map = {⟨r → n⟩ ∈ map : n ⩾ k}
14     for all r ∈ map do
15       out += ((x r y) ∧ B⃗ ⇒ H)
16     end for
17   end for
18   return out
19 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' and returns all possible bindings for variable ?r. 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). 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(B⃗ ⇒ H, k, KG)
2    map = {}
3    q = B⃗ + H
4    maxSupp = 0
5    remainingSteps = size(H)
6    A_n = newAtomVariants(B⃗ ⇒ H)
7    for all ⟨s, p, o⟩ ≺ H : ⟨s, p, o⟩ ∈ KG do
8      A_r = bindProjections(A_n, B⃗, bind(q, ⟨s, p, o⟩, H))
9      for all x ∈ A_r do
10       if map[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   map = {⟨x → n⟩ ∈ map : n ⩾ k}
19   return map
20 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 items 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).

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 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, the *best* unbound atom with the smallest size is selected from $\vec{B}$ (line 18). 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 19 to 28). In the last 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 30 to 32). All resolved atoms are saved into A_r and returned on line 33.

Algorithm 4. *The RDFRules bind projections query*

```
function bindProjections(A_n, B⃗, q, KG)
 A_r, A_c, A_b = {}
 // PHASE I
 for all (x ?r y) ∈ A_n do
  if z ∈ {x, y} : z ∈ q ∨ isDangling(z) then
   A_c += (x ?r y)
  else
   A_b += (x ?r y)
  end if
 end for
 // PHASE II
 if ¬isEmpty(A_c) ∧ exists(q) then
  for all (x ?r y) ∈ A_c do
   A_r ++= bindFreshAtom((x ?r y), q)
  end for
 end if
 // PHASE III
 best = argmin_i(size(B_i ∈ B⃗))  // B_i is the i-th atom of B⃗
 for all (x ?r y) ∈ A_b do
  if size((x ?r y)) ⩽ size(best) then
   χ = bindFreshAtom((x ?r y), q)
   for all (x p y) ∈ χ do
    q' = bind(q, p, (x ?r y))
    if exists(q') then A_r += (x p y)
   end for
   A_b -= (x ?r y)
  end if
 end for
 // PHASE IV
 for all ⟨s, p, o⟩ ≺ best : ⟨s, p, o⟩ ∈ KG do
  A_r ++= bindProjections(A_b, B⃗ - best, bind(q, ⟨s, p, o⟩, best))
 end for
 return A_r
end function
```

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 of working with numerical data in association rule mining is to perform 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.

By default, the mining process starts with some user-defined minimum head size threshold *minHS* and minimum head coverage *minHC*. Let an instantiated atom $A_i = (?x\ p\ C)$, containing a numerical literal C, be derived from atom $A = (?x\ p\ ?y)$. To ensure that atom A_i is included in the consequent of a rule, the following inequality must be true:

$$size(A_i) \geqslant size(A) \cdot minHC, \tag{1}$$

where $size(A) \geqslant minHS$. This observation can help to merge numerical (or even nominal) values. This will create bigger groups (bins) with size satisfying the *minHC* threshold.

Data Type Determination As a first step, all predicates that have a numeric range have to be found. Most RDF serializations directly encode data types. Data types can also be determined from associated RDF vocabularies, which contain information about the range of the predicates.

Discretization in the Rule Head For each predicate p with a numeric range and its corresponding list of numerical literals $L_p \in L$, where L is a set of all numerical lists in the KG, we apply the *equal-frequency* discretization. This process is performed on the sorted list L_p. Intervals are gradually created by merging the numbers until the frequency of the currently expanded interval reaches a predefined threshold. 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:

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

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

In our approach, a constructed interval I is broadened until the atom (?x p I) satisfies Condition 1. All atoms constructed by this process can be used as the head of a rule since they exceed the minimum head size and head coverage thresholds.

Discretization in the Rule Body The head coverage of a rule is calculated from the head size. Therefore, according to Condition 2, the discretization threshold for generating intervals is dependent on the head of the rule. Let A_i be an atom considered to be added into the body of a rule $\vec{B} \Rightarrow H$. Then the atom A_i can only be added if the following condition is met:

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

For instance, this rule:

$$(?a \text{ <hasNumberOfPeople> "[1M;5M]"}) \Rightarrow$$
$$(?a \text{ <hasInflation> "[0.01;0.015]"}),$$

containing intervals in all atoms at the object position, is accepted for output only if

$$size(?a \text{ <hasNumberOfPeople> "[1M;5M]"}) \geqslant$$
$$size(?a \text{ <hasInflation> ?b}) \cdot minHC.$$

However, the previous discretization step, focused only on the head of the rule, only ensured that

$$size(\text{?a <hasInflation> "[0.01;0.015]"}) \geqslant$$
$$minHC \cdot size(\text{?a <hasInflation> ?b}).$$

The solution is to perform the discretization process for all numerical lists in L separately with respect to the current head of the rule.

5.3. Multiple Graphs

Efficiently working with multiple graphs requires data structures supporting *quads* (subject, predicate, object, and graph assignment) 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>) ⇒

(?a dbo:deathPlace ?b <DBpedia>).

The existence of such a rule depends on a same description of identical resources in all 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). This inconsistency can be resolved by *owl:sameAs* predicate, which would join these two descriptions:

<Prague> owl:sameAs dbr:Prague .

Support for named graphs should also be reflected in the user-set pattern for rule learning, introduced in detail in the next subsection. Before mining, the user can decide which atoms will be tightly bound to a graph or whether to enable *graph-aware* mining at all. For example, the user should be able to define a pattern of a mining task where all rules have to consist of parts belonging to various graphs, e.g.,

(?a ? ?b <YAGO>) ⇒ (?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 inclusion of graph information in the rule mining process also requires an extension of fact indexes, described in 3.5, to: PG, PSG, POG, PSOG. These indexes 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 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 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 rules applicable in this grammar, and S is the start symbol representing the whole item pattern.

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

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 ¬, e.g, ¬[<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:

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

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)

(?a <wasBornIn> ?b) ∧
(?b $?_c$ $?_c$ <DBpedia>) ⇒
(?a [<livesIn>, <deadIn>] ?b),

(a matching rule)

(?a <wasBornIn> ?b) ∧
(?b dbo:isCityOf <USA> <DBpedia>) ⇒
(?a <deadIn> ?b).

During the mining process, rules are pruned based on all input 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 of *RP*. 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 of *RP*.

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. [34] 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 2). The support threshold is continuously increasing during mining and the result set always contains at most *k* rules with the highest head coverage.

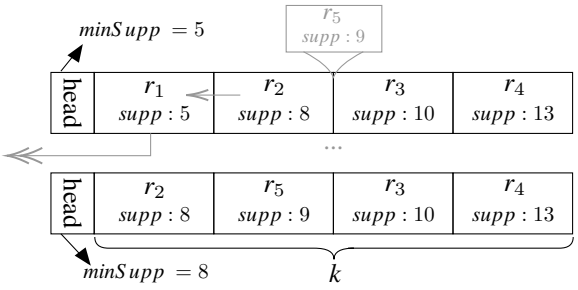


Figure 2. The top-*k* strategy using a priority queue with modification of the minimum support threshold.

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. The standard confidence formula is:

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

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. Support for the Lift Measure

When the user wants to express the sought rules in terms of the quantitative relationship between the antecedent and consequent of the rules, AMIE+ only provides a single option: the confidence (or, the PCA confidence). We suggest to complement the confidence with the *lift*, which is a measure often regarded as of equal importance as the confidence [39, 40].

Intuitively, the lift should reflect an increase in the probability of the head occurrence given the validity of the conditions expressed in the body of the rule (which corresponds to the confidence measure) compared to the probability of the head occurrence under a random choice across the complete dataset. The result is a value expressing the relative increase in the probability of the head occurrence caused by the addition of the body:

$$lift(\vec{B} \Rightarrow H) = \frac{conf(\vec{B} \Rightarrow H)}{hconf(H)}.$$

The lift relies on the computation of the probability of the head occurrence, the so-called *head confidence* (*hconf*). The proposed formula depends on the given variant of a head atom H. There are three atom variants:

(?a p ?b) (two variables),

(?a p C) (one variable at the subject position),

(C p ?a) (one variable at the object position).

For the head atom (?a p ?b) with a specified predicate p, we need to determine the probability of binding any resource x with the predicate p. It is the ratio between the number of distinct subjects bound with the predicate p and the number of distinct subjects in the whole KG (exemplified in Figure 1). Similarly, it is also computed for head atoms with constants, but the variable is always bound with the predicate p:

if $H = (?a\ p\ ?b)$ then

$$hconf(\vec{B} \Rightarrow H) = \frac{\#s : \exists\langle s,p,o\rangle \prec (?a\ p\ ?b)}{\#s : \exists\langle s,p,o\rangle \prec (?a\ ?r\ ?b)},$$

if $H = (?a\ p\ C)$ then

$$hconf(\vec{B} \Rightarrow H) = \frac{\#s : \exists\langle s,p,o\rangle \prec (?a\ p\ C)}{\#s : \exists\langle s,p,o\rangle \prec (?a\ p\ ?b)},$$

if $H = (C\ p\ ?a)$ then

$$hconf(\vec{B} \Rightarrow H) = \frac{\#o : \exists\langle s,p,o\rangle \prec (C\ p\ ?a)}{\#o : \exists\langle s,p,o\rangle \prec (?a\ p\ ?b)}.$$

Notice that the definition of lift refers to the standard confidence that is defined under the CWA. For the head confidence computation, we assume that the data is complete. Hence, if some resource is not bound with a predicate, then it is considered a negative example. E.g., for the head atom (?a <livesIn> ?b), negative examples are such resources that are not in the domain of the <livesIn> predicate.

The lift value is a non-negative real number with the following semantics:

- if $lift(\vec{B} \Rightarrow H) > 1$ the body of the rule is able to predict the head better than a random choice,
- if $lift(\vec{B} \Rightarrow H) < 1$ the body is not able to predict the head better than a random choice,
- if $lift(\vec{B} \Rightarrow H) = 1$ the body and the head are independent of each other.

This means that the closer the lift is to 1 the less relevant is the body of the in the context of prediction the head. Therefore, high confidence of a rule does not necessarily mean a better prediction of the head given the body than a random choice.

5.7. 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 partial 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 trivial 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 consists 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) returns one of the three pre-defined values:

$$sim_t(\langle s_1, p_1\rangle, \langle s_2, p_2\rangle) = \begin{cases} 1 & \text{if } s_1 = s_2, \\ 0.5 & \text{if } s_1 \neq s_2 \wedge p_1 = p_2 \wedge \\ & \exists x \in \{s_1, s_2\} : IsVar(x), \\ 0 & \text{otherwise.} \end{cases}$$

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 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 be assigned to 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.

$$sim_a(A_1, A_2) = \frac{1}{3}[sim_t(\langle s_1, p_1\rangle, \langle s_2, p_2\rangle) + sim_t(\langle o_1, p_1\rangle, \langle o_2, p_2\rangle) + sim_p(p_1, p_2)].$$

Suppose two rules r_1 and r_2, where r_1 has the length greater than or equal to the length of r_2, $|r_1| \geqslant |r_2|$ (the rule length is the number of atoms in a rule). Then the similarity function sim_r between two rules is defined as the sum of atom similarities between the atoms from r_1 and atoms from r_2, normalized using the maximum similarity:

$$sim_r(r_1, r_2) = \frac{1}{|r_1|} \sum_{i=1}^{|r_1|} max(sim_a(A_i^{r_1}, A_1^{r_2}), \ldots, sim_a(A_i^{r_1}, A_{|r_2|}^{r_2}))$$

5.8. 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 [38].

This technique, described in Algorithm 5, processes input rules in the order specified in Figure 3. For each rule, the algorithm checks whether the rule correctly classifies 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.

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.

Figure 3. Rule ranking criteria for rule pruning.

Algorithm 5. *Rule data coverage pruning*

```
1  function dataCoveragePruning(rules, KG)
2    KG' = KG
3    rules = sort rules according to criteria in Fig. 3
4    prunedRules = ()
5    for each (B⃗ ⇒ H) ∈ rules do
6      ruleCoversTriples = false
7      for each θ : B⃗θ = (t_1 ∧ ... ∧ t_n) ∧ t_1,...,t_n ∈ KG do
8        // where θ is the substitution defined in Sec. 3.2.1
9        t_h = Hθ
10       if t_h ∈ KG' then
11         KG' -= t_h
12         ruleCoversTriples = true
13       end if
14     end for
15     if ruleCoversTriples then prunedRules += (B⃗ ⇒ H)
16   end for
17   return prunedRules
18 end function
```

In the list of rules mined by AMIE+, it is often the case that a single triple is covered by multiple rules. After data coverage pruning, many rules are removed, but it is still ensured that the new set of rules covers the same triples as the original rule set. Also, on transactional data, it has been empirically shown that the data coverage pruning (in combination with default rule pruning[10]) reduces the number of input rules by

[10]The well-known Classification Based on Associations (CBA) algorithm [41] essentially corresponds to data coverage pruning

up to two magnitudes, while maintaining good classification performance. For example, experiments performed on 26 datasets, reported in [41], showed that, on average, 35.140 input rules were pruned into 69 final rules.

6. RDFRules: Reference Implementation

While AMIE+ constitutes a breakthrough approach for rule learning from RDF-style 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.[11]

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).

RDFRules is freely available under the GPLv3[12] open-source license and is hosted on GitHub[13].

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 4.

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.

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

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

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

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:

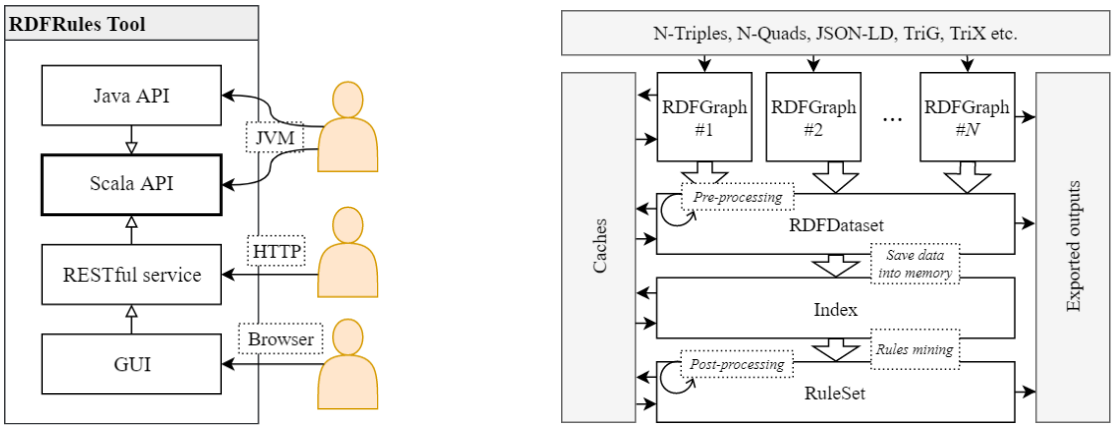
$$\textit{RDFGraph}^{*} \rightarrow \textit{RDFDataset} \rightarrow \textit{Index} \rightarrow \textit{RuleSet}$$

All the transformations are lazy[14] 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 *strict* ones. The streaming actions process data as small chunks (e.g., triples or rules) without large memory requirements, while the strict 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.

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



(a) Available interfaces (b) Main data structures and processes

Figure 4. Architecture of RDFRules.

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[15] 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 saving/caching its data structures and transformations on 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 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. The operations implemented over the *RDFDataset* structure are the same as for the *RDFGraph* structure, aside a few extensions.

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 indexes described in Section 3.5. These indexes 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 indexes 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.

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

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, or lift measures. These additional measures of significance can only be calculated after the main mining phase within the *RuleSet* structure.

MinHeadSize	Minimum number of triples covered by 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.

Table 1
Available user-defined pruning thresholds used in the mining process in RDFRules.

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*.
- *WithInstances*: it allows to mine rules with constants in the subject or object position.
- *WithObjectInstances*: it allows to mine rules with constants in the object position only.
- *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 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*, *PCA confidence*, *head confidence* and *lift*. The rules can be filtered and sorted according to all these measures.

RDFRules supports rule clustering with the DBScan algorithm [42], using similarity functions proposed in Section 5.7. 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.

	Triples	Subjects	Predicates	Objects
YAGO2 core	948 358	470 483	36	400 341

Table 2
YAGO2 core dataset properties.

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.

7. Experiments

We performed two kinds of experiments. The first compare our proposed enhancements presented in Section 5 and implemented within our RDFRules framework presented in Section 6 with the original implementation of the AMIE+ algorithm. The second is focused on some additional methods, such as the top-k approach and rule patterns, which are compared with common mining approaches only employing basic thresholds.

7.1. Setup

For all experiments, we used the YAGO2 core dataset available from the Max Planck Institute website.[16] The number of triples and their unique elements are shown in Table 2.

All the mining tasks are set with the minimum head size threshold 100 and maximum rule length 3. Each experiment is composed of a set of mining thresholds, constraints, patterns, the input dataset, the number of threads 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[17]. 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:

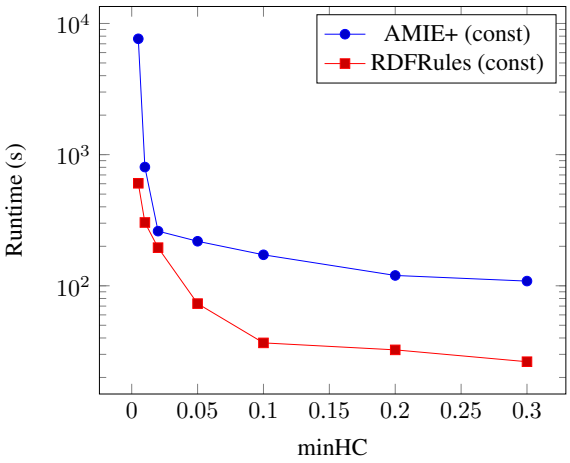


Figure 5. AMIE+ vs RDFRules: Rule mining with constants.

- 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,[18], 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.[19] Some selected tasks, their settings, and results are shown in Table 3. We performed two experiment types: 1) mining logical rules only with variables – rules only with variables at the subject and object positions – and 2) mining rules with 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 thresholds (MinConf+PCA) and the minimum head coverage threshold (MinHC). The Diff column contains the difference between the runtimes of AMIE+ and RDFRules. The Rules column contains the number of rules returned by RDFRules.

The results are also visualized in Figure 6 for mining without constants, and in Figure 5 for mining with

[16] https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/downloads/
[17] https://metavo.metacentrum.cz/en/index.html
[18] https://github.com/propi/rdfrules/tree/master/experiments
[19] https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/amie/

Task	Cores	MinHC	MinConf +PCA	Runtime		Diff	Rules
				AMIE+	RDFRules		
only with variables	8	0.005	0.1	54.8 s ± 2.49 s	18.72 s ± 1.04 s	-36.08 s	48
		0.01		15.32 s ± 222.1 ms	15.93 s ± 84.21 ms	+613.3 ms	31
		0.1		9.91 s ± 102.1 ms	11.98 s ± 56.4 ms	+2.07 s	10
with constants	8	0.005	0.1	2.12 h ± 3.96 min	10.08 min ± 13.38 s	-1.96 h	1,043,539
		0.01		13.40 min ± 9.22 s	5.06 min ± 11.56 s	-8.33 min	123,877
		0.1		2.87 min ± 5.76 s	36.7 s ± 1.63 s	-2.26 min	28
	1	0.01		27.94 min ± 15.47 s	15.47 min ± 12.51 s	-12.47 min	123,877

Table 3
The mining runtime of AMIE+ and RDFRules with various settings.

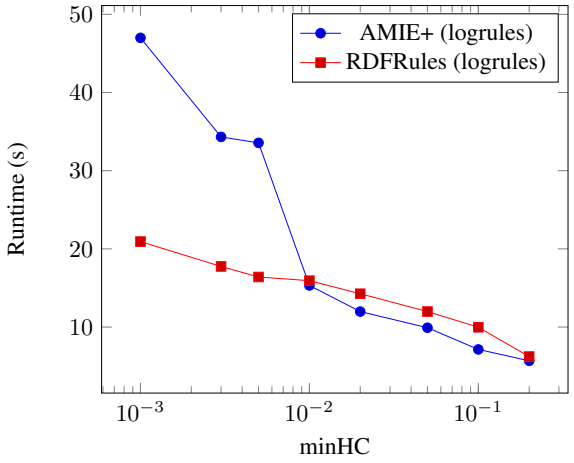


Figure 6. AMIE+ vs RDFRules: Rule mining without constants.

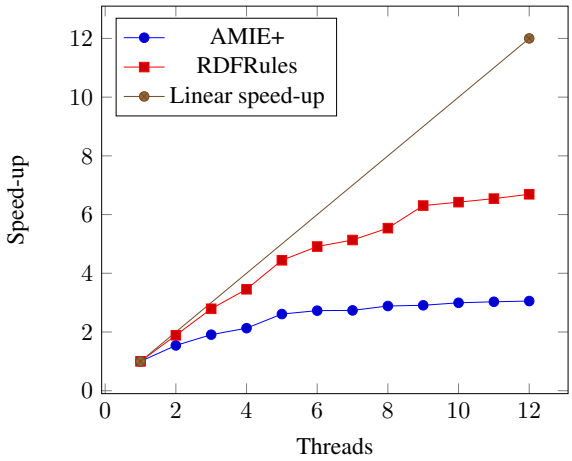


Figure 7. Scalability of RDFRules and AMIE+. Mining with constants and with $minHC = 0.01$.

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 3). The degree to which the two systems scale when mining with constants is depicted in Figure 7.

7.3. 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 show some rules mined with rule patterns, comparing the mining time with that of mining all rules followed by subsequent filtering by a particular pattern.

Figure 8 and Table 4 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.

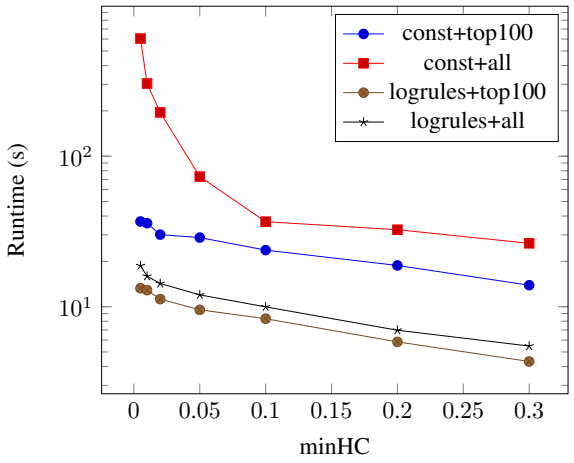


Figure 8. RDFRules rule mining, only with variables (logrules) and instantiated (const), with vs. without the top-*k* approach.

Task	MinHC	Runtime
only with variables	0.005	13.26 s ± 746.2 ms
	0.01	12.87 s ± 718.9 ms
	0.1	8.320 s ± 709.4 ms
with constants	0.005	36.80 s ± 2.42 s
	0.01	35.85 s ± 3.07 s
	0.1	23.76 s ± 1.06 s

Table 4

The mining runtime of the RDFRules top-*k* approach.

Rules	Runtime	
	All	Top-100
10,000	1.69 s ± 68.6 ms	195.3 ms ± 18.66 ms

Table 5

Confidence values for 10,000 rules compared with the top-*k* approach. Settings are: $minConf = 0.1$, num of cores is 8.

As described in Section 5.5, the top-*k* approach can also be used to speed-up confidence computation (standard or PCA), and subsequently also for computation of the lift measure. Table 5 contains results for confidence computation (both standard and PCA) of 10,000 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:

$$(? \; ? \; ?) \Rightarrow (? \; \text{<hasAcademicAdvisor>} \; ?). \quad (3)$$

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

$$(? \; \text{<hasWonPrize>} \; ?) \Rightarrow (? \; ? \; ?). \quad (4)$$

Both tasks were launched with $minHC = 0.01$, 8 cores, without confidence counting, and with constants. Table 6 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. 4 is slower than with mining with the pattern depicted in Eq. 3. Figure 9 shows three examples of rules generated in these two experiments.

Task	Runtime	Rules
Mine all + filter by patterns	5.03 min ± 10.47 s	137,595
Mine by pattern 3	216 ms ± 186 ms	13
Mine by pattern 4	14.29 s ± 1.1 s	13

Table 6

Mining with and without rule patterns.

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

Figure 9. The example of rules mined by RDFRules with patterns.

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 dis-

covered. 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 at https://github.com/propi/rdfrules. This software integrates several APIs (Java API, Scala API, and RESTful API) with a graphical user interface. 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 adding support for RDF schemas and ontologies, which would involve resource types with hierarchies into the mining process. 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, AMIE+ produces logical rules with a possibly complex structure, which may be found difficult to understand by some users. From the user perspective, research into human-perceived interpretability of logical rules is urgently needed. In terms of applications, we consider investigating to what extent RDFRules can complement the recent generation of ILP systems such as Metagol [43], or MIGO [44] 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

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 University of Economics, Prague by long term institutional support of research activities.

References

[1] R. Agrawal, R. Srikant et al., Fast algorithms for mining association rules, 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.

[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.

[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.

[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.

[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.

[8] M. Barati, Q. Bai and Q. Liu, Mining semantic association rules from RDF data, *Knowledge-Based Systems* **133** (2017), 183–196.

[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*.

[11] B. Goethals and J. Van den Bussche, Relational association rules: getting Warmer, in: *Pattern Detection and Discovery*, Springer, 2002, pp. 125–139.

[12] H. Paulheim and C. Bizer, Type inference on noisy RDF data, in: *International semantic web conference*, Springer, 2013, pp. 510–525.

[13] 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.

[14] C. Meilicke, M.W. Chekol, D. Ruffinelli and H. Stuckenschmidt, Anytime bottom-up rule learning for knowledge graph completion (2019).

[15] 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.

[16] 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.

[17] 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.

[18] V. Nebot and R. Berlanga, Finding association rules in semantic web data, *Knowledge-Based Systems* **25**(1) (2012), 51–62.
[19] 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.
[20] M. Nickel, L. Rosasco and T. Poggio, Holographic embeddings of knowledge graphs, in: *Thirtieth AAAI conference on artificial intelligence*, 2016.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] 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.
[26] Z. Yin and Y. Shen, On the dimensionality of word embedding, in: *Advances in Neural Information Processing Systems*, 2018, pp. 887–898.
[27] 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.
[28] S. Ceri, G. Gottlob and L. Tanca, *Logic programming and databases*, Springer Science & Business Media, 2012.
[29] C.K.-S. Leung, Anti-monotone constraints, *Encyclopedia of Database Systems* (2009), 98–98.
[30] 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.
[31] T. Kliegr and J. Kuchař, Tuning Hyperparameters of Classification Based on Associations (CBA), in: *Proceedings of ITAT 2019*, CEUR-WS, 2019.
[32] G.I. Webb, Filtered-top-k association discovery, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **1**(3) (2011), 183–192.
[33] 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.
[34] 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.
[35] E.R. Omiecinski, Alternative interest measures for mining associations in databases, *IEEE Transactions on Knowledge and Data Engineering* **15**(1) (2003), 57–69.
[36] J. Fürnkranz, D. Gamberger and N. Lavrač, *Foundations of rule learning*, Springer Science & Business Media, 2012.
[37] 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.
[38] 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.
[39] Z. Zheng, R. Kohavi and L. Mason, Real world performance of association rule algorithms, in: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2001, pp. 401–406.
[40] P.D. McNicholas, T.B. Murphy and M. O'Regan, Standardising the lift of an association rule, *Computational Statistics & Data Analysis* **52**(10) (2008), 4712–4721.
[41] 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.
[42] 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.
[43] A. Cropper and S.H. Muggleton, Learning Higher-Order Logic Programs through Abstraction and Invention, in: *IJCAI*, 2016, pp. 1418–1424.
[44] C. Hocquette and S.H. Muggleton, Can Meta-Interpretive Learning outperform Deep Reinforcement Learning of Evaluable Game strategies?, *arXiv preprint arXiv:1902.09835* (2019).