AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases

Luis Galárraga[1], Christina Teflioudi[1], Katja Hose[2], Fabian M. Suchanek[1]
[1]Max-Planck Institute for Informatics, Saarbrücken, Germany
[2]Aalborg University, Aalborg, Denmark
[1]{lgalarra, chteflio, suchanek}@mpi-inf.mpg.de, [2]{khose}@cs.aau.dk

ABSTRACT

Recent advances in information extraction have led to huge knowledge bases (KBs), which capture knowledge in a machine-readable format. Inductive Logic Programming (ILP) can be used to mine logical rules from the KB. These rules can help deduce and add missing knowledge to the KB. While ILP is a mature field, mining logical rules from KBs is different in two aspects: First, current rule mining systems are easily overwhelmed by the amount of data (state-of-the art systems cannot even run on today's KBs). Second, ILP usually requires counterexamples. KBs, however, implement the open world assumption (OWA), meaning that absent data cannot be used as counterexamples. In this paper, we develop a rule mining model that is explicitly tailored to support the OWA scenario. It is inspired by association rule mining and introduces a novel measure for confidence. Our extensive experiments show that our approach outperforms state-of-the-art approaches in terms of precision and coverage. Furthermore, our system, AMIE, mines rules orders of magnitude faster than state-of-the-art approaches.

Categories and Subject Descriptors

H2.8 [**Information Systems**]: Database Applications

General Terms

Algorithms

Keywords

Rule Mining, Inductive Logic Programming, ILP

1. INTRODUCTION

In recent years, we have experienced the rise of large knowledge bases (KBs), such as Cyc [23], YAGO [35], DBpedia [5], and Freebase[1]. These KBs provide information about a great variety of entities, such as people, countries, rivers, cities, universities, movies, animals, etc. Moreover, KBs also contain facts relating these entities, e.g., who was born where, which actor acted in which movie, or which city is located in which country. Today's KBs contain millions of entities and hundreds of millions of facts.

Yet, even these large KBs are not complete. Some of them are extracted from natural language resources that inevitably exhibit gaps. Others are created and extended manually. Making these KBs complete requires great effort to extract facts, check them for correctness, and add them to the KB. However, KBs themselves often already contain enough information to derive and add new facts. If, for instance, a KB contains the fact that a child has a mother, then the mother's husband is most likely the father:

$$motherOf(m, c) \wedge marriedTo(m, f) \Rightarrow fatherOf(f, c)$$

As for any rule, there can be exceptions, but in the vast majority of cases, the rule will hold. Finding such rules can serve four purposes: First, by applying such rules on the data, new facts can be derived that make the KB more complete. Second, such rules can identify potential errors in the knowledge base. If, for instance, the KB contains the statement that a totally unrelated person is the father of a child, then maybe this statement is wrong. Third, the rules can be used for reasoning. Many reasoning approaches rely on other parties to provide rules (e.g., [27, 31]). Last, rules describing general regularities can help us understand the data better. We can, e.g., find out that countries often trade with countries speaking the same language, that marriage is a symmetric relationship, that musicians who influence each other often play the same instrument, and so on.

The goal of this paper is to mine such rules from KBs. We focus on RDF-style KBs in the spirit of the Semantic Web, such as YAGO [35], Freebase[1], and DBpedia [5]. These KBs provide binary relationships in the form of RDF triples[2]. Since RDF has only positive inference rules, these KBs contain only positive statements and no negations. Furthermore, they operate under the *Open World Assumption* (OWA). Under the OWA, a statement that is not contained in the KB is not necessarily false; it is just *unknown*. This is a crucial difference to many standard database settings that operate under the *Closed World Assumption* (CWA). Consider an example KB that does not contain the information that a particular person is married. Under CWA we can conclude that the person is not married. Under OWA, however, the person could be either married or single.

Mining rules from a given dataset is a problem that has a long history. It has been studied in the context of association rule mining and inductive logic programming (ILP). Association rule mining [3] is well-known in the context of sales databases. It can find rules such as "If a client bought beer and wine, then he also bought aspirin". The confidence of such a rule is the ratio of cases where beer and wine was actually bought together with aspirin. Association rule mining inherently implements a closed world assumption: A rule

[1]http://freebase.com

WWW, '13 Rio de Janeiro, Brazil
ACM 978-1-4503-2035-1/13/05.

[2]http://www.w3.org/TR/rdf-primer/

that predicts new items that are not in the database has a low confidence. It cannot be used to (and is not intended to be used to) add new items to the database.

ILP approaches deduce logical rules from ground facts. Yet, current ILP systems cannot be applied to semantic KBs for two reasons: First, they usually require negative statements as counter-examples. Semantic KBs, however, usually do not contain negative statements. The semantics of RDF are too weak to deduce negative evidence from the facts in a KB.[3] Because of the OWA, absent statements cannot serve as counter-evidence either. Second, today's ILP systems are slow and cannot handle the huge amount of data that KBs provide. In our experiments, we ran state-of-the-art approaches on YAGO2 for a couple of days without obtaining any results.

In this paper, we propose a rule mining system that is inherently designed to work under the OWA, and efficient enough to handle the size of today's KBs. More precisely, our contributions are as follows:
(1) A method to simulate negative examples for positive KBs (the Partial Completeness Assumption)
(2) An algorithm for the efficient mining of rules.
(3) A system, AMIE, that mines rules on millions of facts in a few minutes without the need for parameter tuning or expert input.
The rest of this paper is structured as follows. Section 2 discusses related work and Section 3 introduces preliminaries. Sections 4 and 5 are the main part of the paper, presenting our mining model and its implementation. Section 6 presents our experiments before Section 7 concludes.

2. RELATED WORK

We aim to mine rules of the form

$$motherOf(m, c) \wedge marriedTo(m, f) \Rightarrow fatherOf(f, c)$$

Technically, these are Horn rules on binary predicates. Rule mining has been an area of active research for the past couple of years. Some approaches mine association rules, some mine logical rules, others mine a schema for the KB, and again others use rule mining for application purposes.

Association Rule Mining. Association rules [3] are mined on a list of *transactions*. A transaction is a set of items. For example, in the context of sales analysis, a transaction is the set of products bought together by a customer in a specific event. The mined rules are of the form $\{ElvisCD, ElvisBook\} \Rightarrow ElvisCostume$, meaning that people who bought an Elvis CD and an Elvis book usually also bought an Elvis costume. However, these are not the kind of rules that we aim to mine in this paper. We aim to mine Horn rules.

One problem for association rule mining is that for some applications the standard measurements for support and confidence do not produce good results. [36] discusses a number of alternatives to measure the interestingness of a rule in general. Our approach is inspired by this work and we also make use of a language bias [2] to reduce the search space.

Logical Rule Mining. Sherlock [32] is an unsupervised ILP method to learn first-order Horn clauses from a set of extracted facts for a given target relation. It uses probabilistic graphical models (PGMs) to infer new facts. It tackles the noise of the extracted facts by extensive filtering in a preprocessing step and by penalizing longer rules in the inference part. For mining the rules, Sherlock uses 2 heuristics: statistical significance and statistical relevance.

The WARMR system [11,12] mines patterns in databases that correspond to conjunctive queries. It uses a declarative language bias to reduce the search space. An extension of the system, WARMER [13], modified the approach to support a broader range of conjunctive queries and increase efficiency of search space exploration.

ALEPH[4] is a general purpose ILP system, which implements Muggleton's Inverse Entailment algorithm [25] in Prolog. It employs a variety of evaluation functions for the rules, and a variety of search strategies.

These approaches are not tailored to deal with large KBs under the Open World Assumption. We compare our system, AMIE, to WARMR and ALEPH, which are the only ones available for download. Our experiments do not only show that these systems mine less sensible rules than AMIE, but also that it takes them much longer to do so.

Expert Rule Mining. Another rule mining approach over RDF data [28] was proposed to discover causal relations in RDF-based medical data. It requires a domain expert who defines targets and contexts of the mining process, so that the correct transactions are generated. Our approach, in contrast, does not rely on the user to define any context or target. It works out-of-the-box.

Generating Schemas. In this paper, we aim to generate Horn rules on a KB. Other approaches use rule mining to generate the schema or taxonomy of a KB. [7] applies clustering techniques based on context vectors and formal concept analysis to construct taxonomies. Other approaches use clustering [21] and ILP-based approaches [9]. For the friend-of-a-friend network on the Semantic Web, [14] applies clustering to identify classes of people and ILP to learn descriptions of these groups. Another example of an ILP-based approach is the DL-Learner [19], which has successfully been applied [15] to generate OWL class expressions from YAGO [35]. As an alternative to ILP techniques, [37] propose a statistical method that does not require negative examples. In contrast to our approach, these techniques aim at generating a schema for a given RDF repository, not logical rules in general.

Learning Rules From Hybrid Sources. [8] proposes to learn association rules from hybrid sources (RDBMS and Ontologies) under the OWA. For this purpose, the definition of frequency (and thus of support and confidence) is changed so that unknown statements contribute with half of the weight of the true statements. Another approach [20] makes use of an ontology and a constraint Datalog program. The goal is to learn association rules at different levels of granularity w.r.t. the type hierarchy of the ontology. While these approaches focus more on the benefits of combining hybrid sources, our approach focuses on pure RDFS KBs.

Further Applications of Rule Mining. [17] proposes an algorithm for frequent pattern mining in KBs that use DL-safe rules. Such KBs can be transformed into a disjunctive Datalog program, which allows seeing patterns as queries. This approach does not mine the Horn rules that we aim at.

Some approaches use rule mining for ontology merging and alignment [10, 24, 30]. The AROMA system [10], e.g.,

[3] RDF has only positive rules and no disjointness constraints or similar concepts.

[4] http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph_toc.html

uses association rules on extracted terms to find subsumption relations between classes and properties of different ontologies. Again, these systems do not mine the kind of rules we are interested in.

In [1] association rules and frequency analysis are used to identify and classify common misusage patterns for relations in DBpedia. In contrast to our work, this approach does not mine logical rules, but association rules on the co-occurrence of values. Since RDF data can be seen as a graph, mining frequent subtrees [6, 18] is another related field of research. However, as the URIs of resources in knowledge bases are unique, these techniques are limited to mining frequent combinations of classes.

Several approaches, such as Markov Logic [31] or URDF [27] use Horn rules to perform reasoning. These approaches can be consumers of the rules we mine with AMIE.

3. PRELIMINARIES

RDF KBs. In this paper, we focus on RDF knowledge bases[5]. An RDF KB can be considered a set of facts, where each fact is a triple of the form $\langle x, r, y\rangle$ with x denoting the subject, r the relation (or predicate), and y the object of the fact. There are several equivalent alternative representations of facts; in this paper we use a logical notation and represent a fact as $r(x, y)$. For example, we write *father(Elvis,Lisa)*.

The facts of an RDF KB can usually be divided into an *A-Box* and a *T-Box*. While the A-Box contains instance data, the T-Box is the subset of facts that define classes, domains, ranges for predicates, and the class hierarchy. Although T-Box information can also be used by our mining approach, we are mainly concerned with the A-Box, i.e., the set of facts relating one particular entity to another.

In the following, we assume a given KB $\mathcal{K}$ as input. Let $\mathcal{R} = \pi_{relation}(\mathcal{K})$ denote the set of relations contained in $\mathcal{K}$ and $\mathcal{E} = \pi_{subject}(\mathcal{K}) \cup \pi_{object}(\mathcal{K})$ the set of entities.

Functions. A *function* is a relation r that has at most one object for every subject, i.e., $\forall x : |\{y : r(x, y)\}| \leq 1$. A relation is an *inverse function* if each of its objects has at most one subject. Since RDF KBs are usually noisy, even relations that should be functions (such as *hasBirthdate*) may exhibit two objects for the same subject. Therefore, we use the notion of *functionality* [33]. The functionality of a relation r is a value between 0 and 1, that is 1 if r is a function:

$$fun(r) := \frac{\#x : \exists y : r(x, y)}{\#(x, y) : r(x, y)}$$

with $\#x : X$ as an abbreviation for $|\{x : X \in \mathcal{K}\}|$. The inverse functionality is defined accordingly as $ifun(r) := fun(r^{-1})$. Without loss of generality, we assume that $\forall r \in \mathcal{R} : fun(r) \geq ifun(r)$ (*FUN-Property*). If that is not the case for a relation r, we can replace all facts $r(x, y)$ with the inverse relation, $r^-(y, x)$, which entails $fun(r^-) \geq ifun(r^-)$. For example, if the KB contains the inverse functional relation *directed(person,movie)*, we can create the functional relation *isDirectedBy(movie,person)* and use only that one in the rule mining process. Manual inspection shows, however, that relations in semantic KBs tend to be more functional than inverse functional. Intuitively, this allows us to consider a fact $r(x, y)$ as a fact about x.

[5] http://www.w3.org/TR/rdf-primer/

Rules. An *atom* is a fact that can have variables at the subject and/or object position. A *(Horn) rule* consists of a head and a body, where the head is a single atom and the body is a set of atoms. We denote a rule with head $r(x, y)$ and body $\{B_1, ..., B_n\}$ by an implication

$$B_1 \wedge B_2 \wedge ... \wedge B_n \Rightarrow r(x, y)$$

which we abbreviate as $\vec{B} \Rightarrow r(x, y)$. One example of such a rule is

$$hasChild(p, c) \wedge isCitizenOf(p, s) \Rightarrow isCitizenOf(c, s)$$

An *instantiation* of a rule is a copy of the rule, where all variables have been substituted by entities. A *prediction* of a rule is the head atom of an instantiated rule if all body atoms of the instantiated rule appear in the KB. For example, the above rule can predict *isCitizenOf(Lisa,USA)* if the KB knows a parent of Lisa (*hasChild(Elvis,Lisa)*) who is American (*isCitizenOf(Elvis,USA)*).

Language Bias. As most ILP systems, AMIE uses a language bias to restrict the search space. We say that two atoms in a rule are *connected* if they share a variable or an entity. A rule is *connected* if every atom is connected transitively to every other atom of the rule. AMIE mines only connected rules, i.e., it avoids constructing rules that contain unrelated atoms. We say that a rule is *closed* if every variable in the rule appears at least twice. Such rules do not predict merely the existence of a fact (e.g. $diedIn(x, y) \Rightarrow \exists z : wasBornIn(x, z)$), but also concrete arguments for it (e.g. $diedIn(x, y) \Rightarrow wasBornIn(x, y)$). AMIE mines only closed rules. We allow *recursive rules* that contain the head relation in the body.

Parallels to Association Rule Mining. Association Rule Mining discovers correlations in shopping transactions. Thus, association rules are different in nature from the Horn rules we aim at. Still, we can show some similarities between the two approaches. Let us define one transaction for every set of n entities that are connected in the KB. For example, in Figure 1, we will define a transaction for the entities *Elvis*, *Lisa* and *Priscilla*, because they are connected through the facts *mother(Priscilla,Lisa)*, *father(Elvis,Lisa)*, *marr(Elvis, Priscilla)*. We label the transaction with the set of these entities. Each atom $r(x_i, x_j)$ on variables indexed by $1 \leq i, j \leq n$ corresponds to an item. A transaction with label $\langle C_1, \ldots, C_n\rangle$ contains an item $r(x_i, x_j)$ if $r(C_i, C_j)$ is in the KB. For example, the transaction $\langle$*Elvis, Lisa, Priscilla*$\rangle$ contains the items {*mother*(x_3,x_2), *father*(x_1,x_2), *marr*(x_1,x_3)}, since the ground atoms *mother(Priscilla,Lisa)*, *father(Elvis,Lisa)* and *marr(Elvis, Priscilla)* are in the KB. In this representation, association rules are Horn rules. In the example, we can mine the association rule

$$\{mother(x_3, x_2), marr(x_1, x_3)\} \Rightarrow \{father(x_1, x_2)\}$$

which corresponds to the Horn rule

$$mother(x_3, x_2) \wedge marr(x_1, x_3) \Rightarrow father(x_1, x_2)$$

Transaction Label	Transaction Items
⟨Elvis,Lisa,Priscilla⟩	{mother(x_3,x_2),father(x_1,x_2),marr(x_1,x_3)}
⟨Barack,Mali,Mich.⟩	{mother(x_3,x_2),father(x_1,x_2),marr(x_1,x_3)}
⟨François,Flora,Ségo⟩	{mother(x_3,x_2),father(x_1,x_2)}

Figure 1: Mining Rules with 3 Variables

Constructing such a table with all possible combinations of entities is practically not very viable. Apart from that,

it faces a number of design issues (e.g., how to deal with transactions that contain the same entities in different orderings). Therefore, association rule mining cannot be used directly to mine Horn rules. However, we take inspiration from the parallels between the two types of mining for our system, AMIE.

4. MINING MODEL

Model. Let us consider a given Horn rule $\vec{B} \Rightarrow r(x, y)$. Let us look at all facts with relation r (Figure 2). We distinguish 4 types of facts: True facts that are known to the KB (KBtrue), true facts that are unknown to the KB (NEWtrue), facts that are known to be false in the KB (KBfalse), and facts that are false but unknown to the KB (NEWfalse). The rule will make certain predictions (blue circle). These predictions can be known to be true (A), known to be false (C), or unknown (B and D). When they are unknown to the KB, they can still be true (B) or false (D) with respect to the real world.

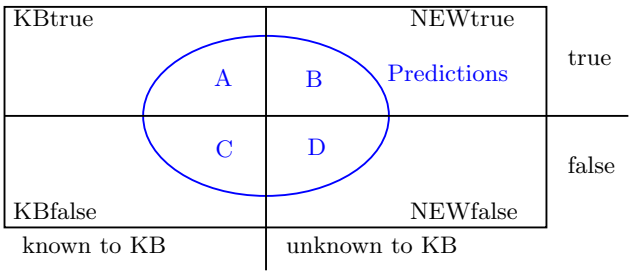

Figure 2: Prediction under Incompleteness

Goal. Our goal is to find rules that make true predictions that go beyond the current KB. In the figure, we wish maximize the area B, and to minimize the area D. There are two obvious challenges in our context: First, the areas NEWtrue and NEWfalse are unknown. So if we wish to maximize B at the expense of D, we are operating in an area outside our KB. We would want to use the areas KBtrue and KBfalse to estimate the unknown area. This, however, leads to the second challenge: Semantic KBs do not contain negative evidence. Thus, the area KBfalse is empty. We will now present different measures that address these challenges.

Support. The *support* of a rule quantifies the number of correct predictions, i.e., the size of A. There are several ways to define the support: It can be the number of instantiations of the rule that appear in the KB. This is what our analogy to association rule mining [3] suggests (Section 3). This measure, however, is not monotonic if we add atoms to the body. Consider, for example, the rule

$$marriedTo(x, y) \Rightarrow marriedTo(y, x)$$

If we add *hasGender(x,male)* to the body, the number of instantiations that are in the KB decreases. If we add an atom with a fresh variable, e.g., *hasFriend(x,z)*, to the body, the number of instantiations increases for every friend of x. This is true even if we add another atom with z to make the rule closed. Alternatively, we can count the number of facts in one particular body atom. This definition, however, depends on the choice of the body atom, so that the same rule can have different supports. We can also count the number of facts of the head atom. This measure decreases monotonically if more body atoms are added and avoids equivalent rules with different support values. With this in mind, we define the support of a rule as the number of distinct pairs of subjects and objects in the head of all instantiations that appear in the KB:

$$supp(\vec{B} \Rightarrow r(x, y)) := \#(x, y) : \exists z_1, ..., z_m : \vec{B} \wedge r(x, y)$$

where $z_1, ..., z_m$ are the variables of the rule apart from x and y.

Head Coverage. Support is an absolute number. This means that a user who thresholds on support has to know the absolute size of the KB to give meaningful values. To avoid this, we also define a proportional version of support. A naive way would be to use the absolute number of support, as defined in the previous paragraph, over the size of the KB. In this case, however, relations that do not have many facts (either because of the incompleteness of the KB or because of their nature), will not be considered in the head of rules, i.e. we will not learn rules predicting such relations. Therefore, we propose to use the notion of *head coverage*. This is the proportion of pairs from the head relation that are covered by the predictions of the rule

$$hc(\vec{B} \Rightarrow r(x, y)) := \frac{supp(\vec{B} \Rightarrow r(x, y))}{\#(x', y') : r(x', y')}$$

Negative Examples. The central challenge of our setting is to provide counter-examples for the rule mining. These can take the role of KBfalse, so that we can estimate the areas NEWtrue and NEWfalse. There are several approaches to this problem: The standard confidence, the standard positive-only learning evaluation score of ILP, and our new partial completeness assumption.

Standard Confidence. The standard confidence measure takes all facts that are not in the KB (i.e., NEWtrue and NEWfalse) as negative evidence. Thus, the standard confidence of a rule is the ratio of its predictions that are in the KB, i.e., the share of A in the set of predictions:

$$conf(\vec{B} \Rightarrow r(x, y)) := \frac{supp(\vec{B} \Rightarrow r(x, y))}{\#(x, y) : \exists z_1, ..., z_m : \vec{B}}$$

The standard confidence is blind to the distinction between "false" and "unknown". Thus, it implements a closed world setting. It mainly describes the known data and penalizes rules that make a large number of predictions in the unknown region. We, in contrast, aim to maximize the number of true predictions that go beyond the current knowledge. We do not want to describe data, but to predict data.

Positive-Only Learning. For cases where the KB does not contain negative examples, Muggleton has developed a *positive-only learning evaluation score* for ILP [26], [22]. It takes random facts as negative evidence:

$$Score = log(P) - log\frac{R + 1}{Rsize + 2} - \frac{L}{P}$$

Here, P is the number of known true facts covered (A in the figure), R is the number of randoms covered, $Rsize$ is the total number of randoms and L is the number of atoms in the hypothesis. The intuition is that a good rule should cover many positive examples, and few or no randomly generated examples. This ensures that the rule is not overly general. Furthermore, the rule should use as few atoms as possible, and thus achieve a high compression. This measure is implemented (among others) in the ALEPH system.

Partial Completeness. We propose to generate negative evidence by the *partial completeness assumption* (PCA). This is the assumption that if $r(x, y) \in KBtrue$ for some x, y, then

$$\forall y' : r(x, y') \in KBtrue \cup NEWtrue \Rightarrow r(x, y') \in KBtrue$$

In other words, we assume that if the database knows some r-attribute of x, then it knows all r-attributes of x. This assumption is certainly true for functional relations r, such as birth dates, capitals, etc. Thanks to the FUN-Property (see Section 4), it is also true for inverse-functional relations, such as *owns*, *created*, etc. The assumption is also true in the vast majority of cases for relations that are not functional, but that have a high functionality. Even for other relations, the PCA is still reasonable for knowledge bases that have been extracted from a single source (such as DBpedia and YAGO). These usually contain either all r-values or none for a given entity.

PCA Confidence. Under the PCA, we normalize the confidence not by the entire set of facts, but by the set of facts of which we know that they are true, together with the facts of which we assume that they are false. If the head atom of the rule is $r(x, y)$, then this set is just the set of facts $\{r(x, y') : r(x, y') \in \mathcal{K}\}$. Thanks to the FUN-Property, the PCA is always applied to the first argument of the head atom:

$$pcaconf(\vec{B} \Rightarrow r(x, y)) := \frac{supp(\vec{B} \Rightarrow r(x, y))}{\#(x, y) : \exists z_1, ..., z_m, y' : \vec{B} \wedge r(x, y')}$$

We show in our experiments that the PCA confidence identifies much more productive rules than the other measures.

5. AMIE

After having outlined the basic definitions and the mining model in Sections 3 and 4, we now outline the core algorithm of our framework and its implementation.

5.1 Algorithm

Goal. Our goal is to mine rules of the form defined in Section 3. One of the main problems of any mining approach is to find an efficient way to explore the search space. The naive algorithm of enumerating all possible rules is infeasible for large KBs. Hence, we explore the search space by iteratively extending rules by *mining operators*.

Mining Operators. We see a rule as a sequence of atoms. The first atom is the head atom and the others are the body atoms. In the process of traversing the search space, we can extend a rule by using one of the following operators:

1. **Add Dangling Atom ($\mathcal{O}_D$)**
   This operator adds a new atom to a rule. The new atom uses a fresh variable for one of its two arguments. The other argument (variable or entity) is shared with the rule, i.e., it occurs in some other atom of the rule.

2. **Add Instantiated Atom ($\mathcal{O}_I$)**
   This operator adds a new atom to a rule that uses an entity for one argument and shares the other argument (variable or entity) with the rule.

3. **Add Closing Atom ($\mathcal{O}_C$)**
   This operator adds a new atom to a rule so that both of its arguments are shared with the rule.

By repeated application of these operators, we can generate the entire space of rules as defined in Section 3. The operators generate even more rules than those we are interested in, because they also produce rules that are not closed. An alternative set of operators could consist of $\mathcal{O}_D$ and an operator for instantiation. But these operators would not be monotonic, in the sense that an atom generated by one operator can be modified in the next step by the other operator. Therefore, we chose the above 3 operators as a canonic set.

Algorithm. We mine rules with Algorithm 1. The algorithm maintains a queue of rules, which initially just contains the empty rule. The algorithm iteratively dequeues a rule from the queue. If the rule is closed (see Section 3), the rule is output, otherwise, it is not. Then, the algorithm applies all operators to the rule and adds the resulting rules to the queue (unless they are pruned out, s.b.). This process is repeated until the queue is empty. We parallelize this process by maintaining a centralized queue, from which the threads dequeue and enqueue. We do not feed predictions of the rules back into the KB. All measures (such as confidence and support) are always computed on the original KB.

Algorithm 1 Rule Mining

```
1: function AMIE(KB K)
2:     q = ⟨[]⟩
3:     Execute in parallel:
4:     while ¬q.isEmpty() do
5:         r = q.dequeue()
6:         if r is closed ∧ r is not pruned for output then
7:             Output r
8:         end if
9:         for all operators o do
10:            for all rules r' ∈ o(r) do
11:                if r' is not pruned then
12:                    q.enqueue(r')
13:                end if
14:            end for
15:        end for
16:    end while
17: end function
```

Pruning. If executed naively, our algorithm will have prohibitively high run-times. The instantiation operator $\mathcal{O}_I$, in particular, generates atoms in the order of $|\mathcal{R}| \times |\mathcal{E}|$. We first observe that we are usually not interested in rules that cover only very few facts of the head relation. Rules that cover, for example, less than 1% of the facts of the head relation can safely assumed to be marginal. Therefore, we set $\theta = 0.01$ as a lower bound for the head coverage. We observe that head coverage decreases monotonically as we add more atoms. This allows us to safely discard any rule that trespasses the threshold (Lines 11 and 12).

The monotonicity of head coverage gives us another opportunity to prune: If a rule $B_1 \wedge ... \wedge B_n \wedge B_{n+1} \Rightarrow H$ does not have larger confidence than the rule $B_1 \wedge ... \wedge B_n \Rightarrow H$, then we do not output the longer rule. This is because both the confidence and the head coverage of the longer rule are necessarily dominated by the shorter rule. This way, we can reduce the number of produced rules (Lines 6 and 7).

Last, we never enqueue a rule that is already in the queue. It is expensive to check two rules for equality. However, it is easy to compute measures such as head coverage, confidence, and PCA confidence for each rule. Two rules can

only be equal if they have the same values for these measures. This restricts the rules that have to be checked. If a rule is duplicate, we do not enqueue it (Lines 11 and 12). We can be sure that any potential duplicates will still be in the queue. This is because the length of the rules increases monotonically: When we dequeue a rule with n atoms, no rule with $n+1$ atoms has ever been dequeued. Thus, when we apply the operators to the rule with n atoms, and generate a rule with $n+1$ atoms, any potential duplicate of that new rule must be in the queue.

Projection Queries. No matter what operator is applied in particular, the algorithm needs to choose a relation for the new atom that is added to the rule. In addition, the instantiation operator $\mathcal{O}_\mathcal{I}$ also allows the choice of an entity. In order to select only relations and entities that will fulfill the head coverage constraint, we rely on the KB to answer *projection queries*. These are queries of the form

SELECT $?x$ WHERE $H \wedge B_1 \wedge ... \wedge B_n$
HAVING COUNT(H)$\geq k$

where $B_1, ..., B_n$ are atoms and k is a natural number. H is the *projection atom* on which we project. $?x$ is the *selection variable*. It is a variable that appears in one or more atoms at the position of one of the arguments or at the position of the relation (as it is common in SPARQL[6]). Such queries select an entity or relation x such that the result of the query $H \wedge B_1 \wedge ... \wedge B_n$ on the KB contains more than k distinct query answers for H.

Using Projection Queries. Projection queries allow us to select the relationship for the operators $\mathcal{O}_\mathcal{D}$, $\mathcal{O}_\mathcal{I}$, and $\mathcal{O}_\mathcal{C}$ in such a way that the head coverage of the resulting rule is above θ. This works by firing a projection query of the form

SELECT $?r$ WHERE $H \wedge B_1 \wedge ... \wedge B_n \wedge ?r(X,Y)$
HAVING COUNT(H)$\geq k$

where X and Y are variables or constants, depending on the type of atoms that the operator generates. The results for $?r$ will be the relations that, once bound in the query, ensure that the support of the rule $B_1 \wedge ... \wedge B_n \wedge ?r(X,Y) \Rightarrow H$ is greater than k. If we choose k equal to θ times the number of facts of the relation of H, then the head coverage of the resulting rules will be greater than θ – which is what we want. For the instantiation operator $\mathcal{O}_\mathcal{I}$, we first fire a projection query to select relations, and then fire projection queries to retrieve entities. This way, projection queries allow us to choose the relationships and entities for the operators in such a way that the head coverage for the new rules is guaranteed to be above θ. Next, we discuss how to implement projection queries efficiently.

5.2 Implementation

SQL and SPARQL. Projection queries are essential for the efficiency of our system. Yet, standard database implementations do not provide special support for these types of queries. Assuming that the KB $\mathcal{K}$ is stored as a three-column table (i.e., each fact is a row with three elements), the projection query template in SQL would be:

SELECT $?x$
FROM $\mathcal{K}$ AS H, $\mathcal{K}$ AS $B_1, \ldots B_n$
WHERE $H.x_i = B_j.x_m, \ldots$
GROUP BY($H.x_1$, $H.x_r$, $H.x_2$)
HAVING COUNT(*) $\geq k$

where $?x$ is replaced with a reference to any of the introduced columns. The WHERE clause lists all variables that are shared between any two atoms in the rule, i.e., all join columns and conditions between atom tables. Since SELECT can only select variables that appear in the GROUP BY statement, the above template is for the case where $?x$ appears in H. The case where $?x$ does not appear in H will require a nested query. Our experience shows that already running the non-nested query on a database of a few million facts can easily take several minutes on an off-the-shelf RDBMS. Hence, efficient SPARQL engines such as RDF-3X [29] are an alternative option. In SPARQL 1.1, the projection query template is:

SELECT $?x$
WHERE {
  $H.x_1$, $H.x_r$, $H.x_2$.
  $B_1.x_1$, $B_1.x_r$, $B_1.x_2$.
  ...
  $B_n.x_1$, $B_n.x_r$, $B_n.x_2$.
}
GROUP BY $H.x_1$ $H.x_r$ $H.x_2$
HAVING COUNT(*) $\geq k$

Again, this is only for the case where $?x$ appears in H. RDF-3X does not support aggregate functions in this way. Thus, we would need extensive postprocessing of query results to compute a projection query – already in the case where $?x$ is in H.

In-Memory Database. We have implemented a vanilla in-memory database for semantic KBs. Our implementation indexes the facts aggressively with one index for each permutation of subject, relation, and object. Each index is a map from the first item to a map from the second item to a set of third items (e.g., a map from relations to a map from subjects to a set of objects). This allows retrieving the instantiations of a single atom in constant time. The existence of a query answer can be checked naively by selecting the atom with fewest instantiations, running through all of its instantiations, instantiating the remaining atoms accordingly, and repeating this process recursively until we find an instantiation of the query that appears in the KB. Select queries are similar.

Projection Queries. Algorithm 2 shows how we answer projection queries. The algorithm takes as input a selection variable $?x$, a projection atom $H = R(X,Y)$, remaining atoms $B_1, ...B_n$, a constant k, and a KB $\mathcal{K}$. We first check whether $?x$ appears in the projection atom. If that is the case, we run through all instantiations of the projection atom, instantiate the query accordingly, and check for existence. Each existing instantiation increases the counter for the respective value of $?x$. We return all values whose counter exceeds k. If the selection variable does not appear in the projection atom, we iterate through all instantiations of the projection atom. We instantiate the query accordingly, and fire a SELECT query for $?x$. We increase the counter for each value of $?x$. We report all values whose counter exceeds k.

Summary. We have identified projection queries as the crucial type of queries for rule mining. Since standard database systems and standard SPARQL systems provide no specifically tuned support for these queries, we have implemented

[6] http://www.w3.org/TR/rdf-sparql-query/

a vanilla in-memory database, which has specific support for projection queries. Our entire implementation is in Java.

Algorithm 2 Answering Projection Queries

```
function SELECT(?x, R(X,Y) ∧ B1 ∧ ... ∧ Bn, k, K)
    map = ∅
    if R ≡ ?x ∨ X ≡ ?x ∨ Y ≡ ?x then
        for all instantiations r(x,y) of R(X,Y) ∈ K do
            q = B1 ∧ ... ∧ Bn
            In q, replace R by r, X by x, Y by y
            if exists instantiation q ∈ K then
                map(value of ?x) + +
            end if
        end for
    else
        for all instantiations r(x,y) of R(X,Y) ∈ K do
            q = B1 ∧ ... ∧ Bn
            In q, replace R by r, X by x, Y by y
            for all x ∈ SELECT ?x FROM K WHERE q do
                map(x) + +
            end for
        end for
    end if
    return {x : map(x) ≥ k}
end function
```

6. EXPERIMENTS

6.1 Overview

Experiments. We conducted 3 groups of experiments: In the first group, we compare AMIE to two popular, state-of-the-art systems that are publicly available, WARMR [11, 12] and ALEPH[4]. In the second group of experiments, we compare the standard confidence to the novel PCA confidence that we have introduced in this paper (Section 4). In the third group of experiments, we run AMIE on different datasets to show the applicability of the system.

Settings. By default, AMIE finds all rules whose head coverage exceeds the default threshold of $\theta = 1\%$. AMIE ranks the resulting rules by decreasing PCA confidence. There is no need to deviate from this default configuration when a user runs AMIE. There are no parameters to tune. All experiments with AMIE on all datasets are run in this setting, unless otherwise mentioned.

For some experiments, we want to compare AMIE's runtime with other systems. To have an equal basis, we make AMIE simulate the metrics of the competitor systems. AMIE can threshold on support, head coverage, confidence, and PCA confidence, and she can rank by any of these. AMIE can also count the support not on two variables, but on a single variable. AMIE can also output non-closed rules. Since this is just a choice of what to output, it does not influence runtime. All experiments with all systems are run on a server with 48GB RAM and 8 CPUs. We always mine rules without constants (i.e., without the instantiation operator), unless otherwise mentioned.

Knowledge Bases. We run our experiments on different KBs. In all cases, we removed the *rdf:type* relationship, because it inflates the size of the KBs. We are aware that the *rdf:type* relationship can be very helpful for rule mining. However, currently no approach (including ours) makes specific use of it. We plan to make use of it in future work. Furthermore, we removed all facts with literals (numbers and strings) from the KBs. Literal values (such as geographical coordinates) are shared by only very few entities, which makes them less interesting for rule mining.

Evaluations. In all experiments, our goal is twofold: First, we want to produce as many predictions as possible beyond the current KB. Second, the percentage of correct predictions shall be as large as possible. The particular challenge is that we want to evaluate *predictions that go beyond the current KB*. We are not interested in describing the existing data, but in generating new data. Therefore, we proceed as follows: We run the systems on an older dataset (YAGO2 [16]). We generate all predictions, i.e., the head atoms of the instantiated rules (see Section 3). We remove all predictions that are in the old KB. Then we compare the remaining predicted facts to the successor of that dataset (YAGO2s [34]). A prediction is "correct" if it appears in the newer KB. A prediction is "incorrect" if it has a highly functional or highly inverse functional relation and contradicts an existing fact in the newer KB, e.g., a different birth place. For all other predictions, we manually validated the facts by checking a sample of 30 of them against Wikipedia pages. This classifies the remaining predictions as "correct" or "incorrect" – except for a few cases where the fact is "unknown", such as the death place of a person that is still alive. The ratio of correct predictions out of the correct and incorrect predictions yields the *precision* of the rule.

Outlook. We note that with the project of predicting beyond current knowledge, we are entering a new, and very risky area of research. We do not expect Horn rules to have extraordinary precisions in the unknown region. Rules can only yield hypotheses about possible facts.

6.2 AMIE vs. WARMR and ALEPH

In this section, we compare AMIE to WARMR and ALEPH. For each system, we conduct 3 experiments: We first compare the usability of the competitor system to AMIE. Then, we compare their runtimes. Last, we compare their outputs.

6.2.1 AMIE vs. WARMR

Usability. WARMR is a system that unifies ILP and association rule mining. Similar to APRIORI algorithms [4], it performs a breadth-first search in order to find frequent patterns. WARMR generates Datalog queries of the form "$? - A_1, A_2, ..., A_n$", where A_i are logical atoms.

To discover frequent patterns (as in association rule mining), we need to have a notion of frequency. Given that WARMR considers queries as patterns and that queries can have variables, it is not immediately obvious what the frequency of a given query is. Therefore, the user needs to specify the predicate that is being counted by the system (the *key predicate*). In the usual scenario of market basket analysis, e.g., the system counts customer transactions. In a scenario in which the database is a KB, one solution is to count entities. Since the key predicate determines what is counted, it is necessary that it is contained in all queries. Therefore, we add a predicate $entity(x)$, which we fill with all entities of the KB. AMIE does not require such a choice.

For WARMR, the user needs to provide specific information about which predicates can be added to a query, which of their variables can be fresh, and which arguments of predicates are allowed to be unified (type declarations). In contrast, AMIE requires none of these. AMIE simply takes as input the KB in triple format.

WARMR is also able to mine rules with constants. The user can define which predicates and arguments should be instantiated with constants (we call this mode MODE1). WARMR then checks all the constants appearing in the facts of that specific predicate and argument and afterwards uses them in the queries. MODE1 naturally entails an increase of the branching factor in the search space and an explosion in the number of candidates that need to be evaluated. Alternatively, WARMR allows the user to set a maximum number of constants to be used for each predicate and argument (MODE2). Unfortunately, though, it does not provide a way for the user to influence the selection of these constants. In other words, there is no guarantee that the constants that WARMR will use are the most promising ones.

WARMR produces rules as output. These rules are not necessarily connected. For example, WARMR mines

$$isMarriedTo(\text{B,C}), \wedge\ isLeaderOf(\text{A,D}) \Rightarrow hasAcademicAdvisor(\text{C,E})$$

This rule is not only nonsensical from a semantic perspective, but also redundant, because the second atom does not influence the implication. Therefore, the user has to filter out these rules from the output.

Thus, we conclude that the broader mission and the broader applicability of WARMR entails that much more configuration, acquaintance, and expert knowledge is needed to make it mine Horn rules on semantic KBs.

Runtime. YAGO2 [16] contains around 940K facts about 470K entities. WARMR was not able to terminate on this data in a time period of 1 day. Therefore, we created a sample of YAGO2. Randomly selecting a number of facts from the initial dataset could break the interesting links between the entities. Therefore, we randomly selected 10,000 seed entities and included their 3-hop neighborhood. This yielded 14K entities and 47K facts. This sample contains all available information in a radius of 3 hops around the seed entities, but much less information about the entities at the periphery of the subgraph. Therefore, we restricted the values for the key predicate to the seed entities only.

Since the sample is much smaller than the original KB, we lowered the support threshold to 5 entities. We ran AMIE with these parameters on the sample. AMIE mined her rules in 3.90 seconds. WARMR, in contrast, took 18 hours. We also ran both systems allowing them to mine rules with constants. AMIE completed the task in 1.53 minutes. WARMR in MODE1 for all relations did not terminate in 3 days. Therefore, we ran it also only for the relations *diedIn*, *livesIn*, *wasBornIn*, for which it took 48h. We also ran WARMR in MODE2. To have reasonable runtimes, we allowed WARMR to find constants only for one predicate (*diedIn*). We also restricted it to find only 20 constants. WARMR ran 19 hours. Table 3 summarizes the runtime results. We conclude that AMIE is better suited for large KBs than WARMR. This is because WARMR is an ILP algorithm written in a logic programming environment, which makes the evaluation of all candidate queries inefficient.

Constants	WARMR	AMIE
no	18h	3.90s
yes	(48h) / (19.3h)	1.53min

Table 3: Runtimes on YAGO2 Sample

Results. After filtering out non-connected rules, WARMR mined 41 closed rules. AMIE, in contrast, mined 207 closed rules, which included the ones mined by WARMR. We checked back with the WARMR team and learned that for a given set of atoms $B_1, ...B_n$, WARMR will mine only one rule, picking one of the atoms as head atom (e.g., $B_1 \wedge ... \wedge B_{n-1} \Rightarrow B_n$). AMIE, in contrast, will mine one rule for each possible choice of head atom (as long as the thresholds are met). In other words, AMIE with the standard support and confidence measures simulates WARMR, but mines more rules. Furthermore, it runs orders of magnitude faster. Especially for large datasets for which the user would have needed to use complicated sampling schemes in order to use WARMR, AMIE can be a very attractive alternative. Even for smaller datasets with rules with constants, AMIE can provide results while WARMR cannot. Moreover, AMIE comes with metrics that go beyond the standard confidence and the standard support. We will show later that these improve the quality of the results.

6.2.2 AMIE vs. ALEPH

Usability. ALEPH can be run with different commands that influence the search strategy. We chose the *induce* command, which runs fastest. For running ALEPH, the user has to specify the target predicate for learning (the head predicate of the rules). In the following, we ran ALEPH successively with all predicates of the KB as targets. In addition, the user has to specify a series of type and mode declarations (similar to WARMR), which will be used as a language bias in order to restrict the search space. In addition, the user needs to provide ALEPH with files containing the background knowledge and positive examples for the target predicate. In contrast, AMIE requires no such input. It will run on a KB without any prespecified choices of predicates.

KB	Facts	ALEPH	AMIE
YAGO2 full	948k	4.96s to > 1 day	3.62min
YAGO2 Sample	47k	0.05s to > 1 day	5.41s

Table 4: Runtimes ALEPH vs. AMIE

Relations	Runtime
isPoliticianOf, hasCapital, hasCurrency	< 5min
dealsWith, hasOfficialLanguage, imports	< 5min
isInterested, hasMusicalRole	<19min
hasAcademicAdvisor, hasChild	> 1 day
isMarriedTo, livesIn, worksAt, isLocatedIn	> 1 day

Table 5: Runtimes of ALEPH on YAGO2

Relations	Runtime
diedIn, directed, hasAcademicAdvisor	< 2min
graduatedFrom, isPoliticianOf, playsFor	< 2min
wasBornIn, worksAt, isLeaderOf	< 2min
exports, livesIn, isCitizenOf	< 1.4h
actedIn, produced, hasChild, isMarriedTo	> 1 day

Table 6: Runtimes of ALEPH on YAGO2 Sample

Runtime. We ran AMIE and ALEPH on YAGO2 [16]. For ALEPH, we used the positive-only evaluation function with $Rsize = 50$ and we considered only clauses that were able to explain at least 2 positive examples, so that we will not get grounded facts as rules in the output. For a fair comparison, we also instructed AMIE to run with a support threshold of 2 facts. AMIE terminated in 3.62 minutes, and found rules for all relations. ALEPH ran for one head relation at a time. For some relations (e.g. $isPoliticianOf$), it terminated in a few seconds. For others, however, we had to

abort the system after 1 day without results (Tables 4 and 5). For each relation, ALEPH treats one positive example at a time. Some examples need little processing time, others block the system for hours. We could not figure out a way to choose examples in such a way that ALEPH runs faster. Hence, we used the sample of YAGO2 that we created for WARMR. Again, runtimes varied widely between relations (Table 6). Some relations ran in a few seconds, others did not terminate in a day. The runtimes with constants are similarly heterogenous, with at least 7 relations not terminating in 1 day.

Results. We compared the output of ALEPH on the head relations for which it terminated to the output of AMIE on these head relations, on the sample dataset. ALEPH mined 56 rules, while AMIE mined 335 rules. We order the rules by decreasing score (ALEPH) and decreasing PCA confidence (AMIE). Table 7 shows the number of predictions, and their total precision as described in Section 6.1. We show the aggregated values at the points where both approaches have produced around 3K, 5K and 8K predictions. AMIE's PCA confidence succeeds in sorting the rules roughly by descending precision, so that the initial rules have an extraordinary precision compared to ALEPH's. AMIE needs more rules to produce the same number of predictions as ALEPH (but she also mines more). We suspect that ALEPH's positives-only evaluation function manages to filter out overly general rules only to some extent. ALEPH will mine, e.g, $livesIn(A, C)$,$isLocatedIn(C, B) \Rightarrow isPoliticianOf(A, B)$. The problem is that ALEPH generates counterexamples by randomly using valid constants for variables A and B. This means that the probability of creating a random example in which B is the place of residence of the specific person A is very low.

System	Top n	Predictions	Precision
ALEPH	7	2997	27%
AMIE	13	3180	66%
ALEPH	9	5031	26%
AMIE	29	5003	47%
ALEPH	17	8457	30%
AMIE	52	8686	45%

Table 7: Top Rules of ALEPH vs. AMIE

6.3 AMIE with Different Metrics

In this section, we compare the standard confidence measure to the PCA confidence measure. We ran AMIE with the default head coverage threshold on the YAGO2 dataset. It contains nearly 500K entities and 948K facts. We sort the rules first by descending PCA confidence, and then by descending standard confidence, and look at the top rules. For each rule, we evaluated the predictions beyond YAGO2 as described in Section 6.1. Figure 8 uses aggregated predictions and aggregated precision to illustrate the results. The n-th dot from the left tells us the total number of predictions and the total precision of these predictions, aggregated over the first n rules. As we see, ranking the rules by standard confidence is a very conservative approach: It identifies rules with reasonable precision, but these do not produce many predictions. Going down in the list of ranked rules, the rules produce more predictions – but at lower precision. The top 30 rules produce 113K predictions at an aggregated precision of 32%. If we rank the rules by PCA confidence, in contrast, we quickly get large numbers of predictions. The top 10 rules already produce 135K predictions – at a precision of 39%. The top 30 rules produce 3 times more predictions than the top 30 rules by standard confidence – at comparable precision. This is because the PCA confidence is less conservative than the standard confidence.

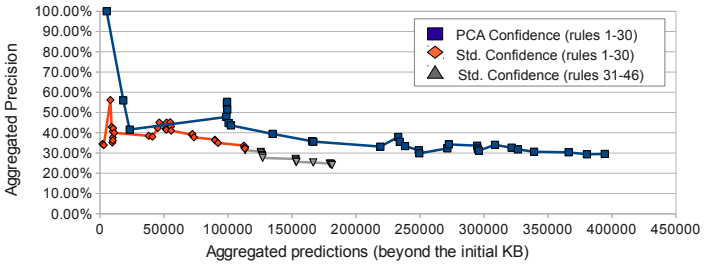

Figure 8: Std. Confidence vs. PCA Confidence

Discussion. The precision of the rules is in the range of 30%-40%. Thus, only a third of the predictions in the unknown region will be correct. Still, even imperfect rules can be useful: If, e.g., a human checks the facts before they are added, then reducing the number of false predictions is a great advantage. If games with a purpose are employed, then the rules can help pre-select candidate facts. If multiple sources are combined, then rules can contribute evidence. If reasoning approaches are used, then rules can be taken into consideration according to their estimated performance. Finally, the precision is better if standard confidence is used.

Predicting Precision. The confidence measures can serve to estimate the actual precision of a rule. In Table 9, we rank the mined rules by their precision and report the average absolute error of the standard and PCA confidence weighted by the number of predictions produced by the rules. We can observe that, on average, the PCA confidence estimates the precision of the rules better than the normal confidence. Thus, reasoning approaches can use the PCA confidence as a weight for the rule.

	Top 20 rules	Top 30 rules	All rules
Confidence	0.76	0.63	0.33
PCA Confidence	0.32	0.29	0.29

Table 9: Average Absolute Error to Precision

We also note that our rules are insightful. Table 10 shows some of the rules we mined. Being able to mine reasonable rules on semantic KBs of this size is an achievement beyond the current state of the art.

$isMarriedTo(x, y) \wedge livesIn(x, z) \Rightarrow livesIn(y, z)$
$isCitizenOf(x, y) \Rightarrow livesIn(x, y)$
$hasAdvisor(x, y) \wedge graduatedFrom(x, z) \Rightarrow worksAt(y, z)$
$wasBornIn(x, y) \wedge isLocatedIn(y, z) \Rightarrow isCitizenOf(x, z)$
$hasWonPrize(x, G.\ W.\ Leibniz) \Rightarrow livesIn(x, Germany)$
$hasWonPrize(x, Grammy) \Rightarrow hasMusicalRole(x, Guitar)$

Table 10: Some Rules by AMIE

6.4 AMIE on Different Datasets

As a proof of concept, we ran AMIE on YAGO2 [16], YAGO2 with constants, and DBpedia [5]. We chose an older version of DBpedia (2.0), so that we can evaluate the output to a newer version of DBpedia (3.8). Due to the large number of relations in DBpedia 2.0, there is an enormous number of rules to be found. We show the time taken to

mine rules with 2 atoms. We provide also the number of predicted facts that are in the newer version of the KB but not in the old one (hits). As Table 11 shows, AMIE can produce rules with or without constants in a reasonable time.

Dataset	Entities	Facts	Runtime	Rules	Hits
YAGO2	470475	948044	3.62min	138	74K
YAGO2 const	470475	948044	17.76min	18886	159K
DBpedia	1376877	6704524	2.89min	6963	122K

Table 11: AMIE on Different Datasets

All rules and results are available at http://www.mpi-inf.mpg.de/departments/ontologies/projects/amie/.

7. CONCLUSION

In this paper, we have presented an approach for mining Horn rules on large RDF knowledge bases. We have introduced a formal model for rule mining under the Open World Assumption, a novel measure to simulate counter-examples, and a scalable algorithm for the mining. In contrast to state-of-the-art approaches, our system (AMIE) requires no input other than the KB, and does not need configurations or parameter tuning. As our extensive experiments have shown, AMIE runs on millions of facts in only a few minutes and outperforms state-of-the-art approaches not only in terms of runtime, but also in terms of the number and quality of the output rules. Our confidence measure can reasonably predict the precision of the rules. In our future work, we plan to consider also the T-Box of the KB in order to produce more precise rules. We also aim to explore the synergies when several rules predict the same fact, and extend the set of rules beyond Horn rules, so that even more complex facts and hidden knowledge can be predicted.

8. REFERENCES

[1] Z. Abedjan, J. Lorey, and F. Naumann. Reconciling ontologies and the web of data. In *CIKM*, 2012.

[2] H. Adé, L. Raedt, and M. Bruynooghe. Declarative bias for specific-to-general ilp systems. *Machine Learning*, 20, 1995.

[3] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, 1993.

[4] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in knowledge discovery and data mining*. 1996.

[5] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. DBpedia: A nucleus for a Web of open data. In *ISWC*, 2007.

[6] Y. Chi, R. R. Muntz, S. Nijssen, and J. N. Kok. Frequent Subtree Mining - An Overview. *Fundam. Inf.*, 66(1-2), 2004.

[7] P. Cimiano, A. Hotho, and S. Staab. Comparing Conceptual, Divisive and Agglomerative Clustering for Learning Taxonomies from Text. In *ECAI*, 2004.

[8] C. d'Amato, V. Bryl, and L. Serafini. Data-driven logical reasoning. In *URSW*, 2012.

[9] C. d'Amato, N. Fanizzi, and F. Esposito. Inductive learning for the Semantic Web: What does it buy? *Semant. web*, 1(1,2), Apr. 2010.

[10] J. David, F. Guillet, and H. Briand. Association Rule Ontology Matching Approach. *Int. J. Semantic Web Inf. Syst.*, 3(2), 2007.

[11] L. Dehaspe and H. Toironen. Discovery of relational association rules. In *Relational Data Mining*. Springer-Verlag New York, Inc., 2000.

[12] L. Dehaspe and H. Toivonen. Discovery of frequent DATALOG patterns. *Data Min. Knowl. Discov.*, 3(1), Mar. 1999.

[13] B. Goethals and J. Van den Bussche. Relational Association Rules: Getting WARMER. In *Pattern Detection and Discovery*, volume 2447. Springer Berlin / Heidelberg, 2002.

[14] G. A. Grimnes, P. Edwards, and A. D. Preece. Learning Meta-descriptions of the FOAF Network. In *ISWC*, 2004.

[15] S. Hellmann, J. Lehmann, and S. Auer. Learning of OWL Class Descriptions on Very Large Knowledge Bases. *Int. J. Semantic Web Inf. Syst.*, 5(2), 2009.

[16] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence Journal*, 2013.

[17] J. Jozefowska, A. Lawrynowicz, and T. Lukaszewski. The role of semantics in mining frequent patterns from knowledge bases in description logics with rules. *Theory Pract. Log. Program.*, 10(3), 2010.

[18] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. In *ICDM*. IEEE Computer Society, 2001.

[19] J. Lehmann. DL-Learner: Learning Concepts in Description Logics. *Journal of Machine Learning Research (JMLR)*, 10, 2009.

[20] F. A. Lisi. Building rules on top of ontologies for the semantic web with inductive logic programming. *TPLP*, 8(3):271–300, 2008.

[21] A. Maedche and V. Zacharias. Clustering Ontology-Based Metadata in the Semantic Web. In *PKDD*, 2002.

[22] T. Mamer, C. Bryant, and J. McCall. L-modified ilp evaluation functions for positive-only biological grammar learning. In *Inductive logic programming*, number 5194 in LNAI. Springer-Verlag, 2008.

[23] C. Matuszek, J. Cabral, M. Witbrock, and J. Deoliveira. An introduction to the syntax and content of Cyc. In *AAAI Spring Symposium*, 2006.

[24] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An Environment for Merging and Testing Large Ontologies. In *KR*, 2000.

[25] S. Muggleton. Inverse entailment and progol. *New Generation Comput.*, 13(3&4), 1995.

[26] S. Muggleton. Learning from positive data. In *ILP*. Springer-Verlag, 1997.

[27] N. Nakashole, M. Sozio, F. Suchanek, and M. Theobald. Query-time reasoning in uncertain rdf knowledge bases with soft and hard rules. In *Workshop on Very Large Data Search (VLDS) at VLDB*, 2012.

[28] V. Nebot and R. Berlanga. Finding association rules in semantic web data. *Knowl.-Based Syst.*, 25(1), 2012.

[29] T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. *Proc. VLDB Endow.*, 1(1), Aug. 2008.

[30] N. F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *AAAI/IAAI*. AAAI Press, 2000.

[31] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2), 2006.

[32] S. Schoenmackers, O. Etzioni, D. S. Weld, and J. Davis. Learning first-order Horn clauses from web text. In *EMNLP*, 2010.

[33] F. M. Suchanek, S. Abiteboul, and P. Senellart. PARIS: Probabilistic Alignment of Relations, Instances, and Schema. *PVLDB*, 5(3), 2011.

[34] F. M. Suchanek, J. Hoffart, E. Kuzey, and E. Lewis-Kelham. YAGO2s: Modular High-Quality Information Extraction. In *German Database Symposium (BTW)*, 2013.

[35] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, 2007.

[36] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *KDD*, 2002.

[37] J. Völker and M. Niepert. Statistical schema induction. In *ESWC*, 2011.