Fast Rule Mining in Ontological Knowledge Bases with AMIE+

Luis Galárraga, Christina Teflioudi, Katja Hose, Fabian Suchanek

Noname manuscript No.
(will be inserted by the editor)

Fast Rule Mining in Ontological Knowledge Bases with AMIE+

Luis Galárraga · Christina Teflioudi · Katja Hose · Fabian M. Suchanek

Received: date / Accepted: date

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 these KBs, such as "If two persons are married, then they (usually) live in the same city". While ILP is a mature field, mining logical rules from KBs is difficult, because KBs make an open world assumption. This means that absent information cannot be taken as counterexamples. Our approach AMIE [16] has shown how rules can be mined effectively from KBs even in the absence of counterexamples. In this paper, we show how this approach can be optimized to mine even larger KBs with more than 12M statements. Extensive experiments show how our new approach, AMIE+, extends to areas of mining that were previously beyond reach.

Keywords Rule Mining · Inductive Logic Programming · ILP · Knowledge Bases

1 Introduction

Recent advances in information extraction have led to the creation of large knowledge bases (KBs). These KBs contain facts such as "London is the capital of the United Kingdom", "Elvis was born in Tupelo", or "Every singer is a person". Some of the most prominent projects in this direction are NELL [6], YAGO [39], DBpedia [5], and Freebase [41]. These KBs provide information about a great variety of entities, such as people, countries, rivers, cities, universities, movies, animals, etc. The KBs know, 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.

These KBs have been constructed by mining the Web for information. In recent years, however, the KBs have become so large that they can themselves be mined for information. It is possible to find *rules* in the KBs that describe common correlations in the data. For example, we can mine the rule

$$livesIn(h, p) \wedge marriedTo(h, w) \Rightarrow livesIn(w, p)$$

This rule captures the fact that, very often, the spouse of a person lives in the same place as the person. 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. For example, if we know where Barack Obama lives, and if we know that Michelle Obama is his wife, then we can deduce (with high probability) where Michelle Obama lives. Second, such rules can identify potential errors in the knowledge base. If, for instance, the KB contains the statement that Michelle Obama lives in a completely different place, 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., [32, 36]). Last, rules describing general regularities can help us understand the data better. We can, e.g., find out that countries often trade with countries

L. Galárraga and F. M. Suchanek
Télécom ParisTech
Paris, France
E-mail: {galarrag, suchanek}@enst.fr

C. Teflioudi
Max Planck Institute for Informatics
Saarbrücken, Germany
E-mail: chteflio@mpi-inf.mpg.de

K. Hose
Aalborg University
Aalborg, Denmark
E-mail: khose@cs.aau.dk

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 [39], Wikidata[1], and DBpedia [5]. These KBs provide binary relationships in the form of RDF triples [43]. 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 the CWA we can conclude that the person is not married. Under the OWA, however, the person could be either married or single.

Mining rules from a data set is the central task of Inductive Logic Programming (ILP). ILP approaches induce logical rules from ground facts. Yet, classical ILP systems cannot be applied to semantic KBs for two reasons: First, they usually require negative statements as counterexamples. Semantic KBs, however, usually do not contain negative statements. The semantics of RDF Schema are too weak to deduce negative evidence from the facts in a KB[2]. 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.

With the AMIE project [16], we have shown how to mine logical rules from KBs despite the absence of explicit counter-examples. The key technique was the Partial Completeness Assumption (PCA). It allowed AMIE to "guess" counterexamples for rules, and thus estimate their quality even under the OWA. We have shown that our approach outperformed other rule mining systems both in terms of the quality and the quantity of the mined rules. AMIE could already run on KBs with up to one million statements – a size that was beyond the reach of any previous ILP-based rule mining system. AMIE achieved this without any need for parameter tuning or expert input.

With the present paper, we develop AMIE even further. We present pruning strategies and approximations that allow the system to explore the search space much more efficiently. This allows us to find Horn rules on KBs with several millions of statements in a matter of hours or minutes. Such large KBs were previously out of reach even for AMIE. We also show how the precision of the predictions can be increased to up to 70% by using type information and joint reasoning. In addition, we provide a thorough investigation of the metrics we use, thus giving a more complete picture of rule mining on large-scale knowledge bases.

More precisely, our contributions are as follows:

- A comprehensive investigation and description of the AMIE approach, including a description of our in-memory database and an evaluation of AMIE's fundamental assumption, the PCA.
- A suite of optimization steps that allow a much more efficient exploration of the search space.
- Extensive experiments that show the competitiveness of our approach, including techniques to increase the precision of our predictions to 70%.

The rest of this paper is structured as follows: Section 2 discusses related work and Section 3 introduces preliminaries. In Section 4, we introduce the Partial Completeness Assumption (PCA) and, based on it, the PCA confidence measure. Section 5 recaptures the AMIE approach from [16], extending it by a description of our in-memory database. Section 6 is the main part of the paper: It presents the pruning strategies that optimize the performance of AMIE. Section 7 presents our experiments before Section 8 concludes.

2 Related Work

Technically speaking, we aim to mine Horn rules on binary predicates. Rule mining has been an area of active research during the past 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. In the following, we survey the most pertinent related work along these lines.

2.1 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 at mining Horn rules. We show in [16] that Horn rule mining

[1] http://www.wikidata.org

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

corresponds to association rule mining on a database that is exponentially large in the maximal number of variables of the rules. One problem for association rule mining is that for some applications the standard measurements for support and confidence do not produce good results. [40] discusses a number of alternatives to measure the interestingness of a rule in general. Our approach is inspired by this work and also makes use of a language bias [2] to reduce the search space.

2.2 Logical Rule Mining

Sherlock [37] is an unsupervised ILP method to learn first-order Horn clauses from open domain facts. Sherlock 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 two heuristics: statistical significance and statistical relevance. Unlike AMIE, it works on facts extracted from free text that are not mapped to crisp relations. QuickFOIL [44] is a standard ILP system based on a generic top-down greedy algorithm and implemented on top of the QuickStep in-memory storage engine [7]. It learns a set of hypotheses (Horn rules) from positive and negative examples of a target relation and a collection of background facts. When refining a rule, the QuickFOIL algorithm greedily picks the clause that maximizes a scoring function depending on the support and the confidence gain of the new rule. Once a rule is mined, the algorithm removes the positive examples covered by the rule and starts the induction process on the remaining facts. QuickFOIL can scale to problem instances with millions of background facts thanks to a set of aggressive pruning heuristics and multiple database optimizations. However, it is not suitable for mining rules under the Open World Assumption, since it requires explicit negative examples. The WARMR system [13, 14] 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 [17], modified the approach to support a broader range of conjunctive queries and increase efficiency of search space exploration. ALEPH[3] is a general purpose ILP system that implements Muggleton's Inverse Entailment algorithm [30] in Prolog. It employs a variety of evaluation functions for the rules as well as a variety of search strategies. These approaches are not tailored to deal with large KBs under the Open World Assumption. We compare our system 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 our approach, but also that they take more time to do so.

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

2.3 Expert Rule Mining

Another rule mining approach over RDF data [33] 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.

2.4 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. [9] applies clustering techniques based on context vectors and formal concept analysis to construct taxonomies. Other approaches use clustering [26] and ILP-based approaches [11]. For the friend-of-a-friend network on the Semantic Web, [19] 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 [24], which has successfully been applied [20] to generate OWL class expressions from YAGO [39]. As an alternative to ILP techniques, [42] proposes 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.

2.5 Relational Machine Learning

Some approaches learn new associations from semantic data without mining explicit logical rules. For example, relational machine learning methods propose a holistic statistical approach that considers both the attribute information and the relationships between entities to learn new links and concepts. [34] applies tensor factorization methods to predict new triples on the YAGO2 ontology by representing the KB as a three-dimensional tensor. In a similar fashion, [21] uses multivariate prediction techniques to learn new links on a social graph. In both approaches, however, the predictions are *opaque*. It is possible to generate predictions, but not to derive general structural knowledge about

the data that can explain the reasons why the predictions were made. For example, these approaches will tell us that Michelle Obama most likely lives in Washington, but they will not tell us that this is because her husband lives in Washington and people tend to live in same place as their spouses. Our approach, in contrast, aims at mining explicit logical rules that capture the correlations in the data. These can then be used to derive new facts and also to explain why these facts were derived.

2.6 Learning Rules From Hybrid Sources

[10] 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 [25] 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 RDF KBs.

2.7 Further Applications of Rule Mining

[22] proposes an algorithm for frequent pattern mining in KBs that uses 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 [12,29,35]. The AROMA system [12], for instance, 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 the same fashion, [45] applies association rules to find synonym predicates in DBpedia. The matched synonyms are then used for predicate expansion in the spirit of data integration. This is a vital task in manually populated KBs where the users may not use canonical names for relations, or for cases when the data is produced by independent providers. In contrast to our work, these approaches do 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 [8,23] 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 [36] or URDF [32] use Horn rules to perform reasoning. These approaches can be consumers of the rules we mine with AMIE.

3 Preliminaries

3.1 RDF KBs

In this paper, we focus on RDF [43] knowledge bases. We follow here the introduction of the preliminaries from [16]. 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 borrow the notation from Datalog 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.

3.2 Functions

A *function* is a relation r that has at most one object for every subject, i.e.,

$$\forall x : |\{y : r(x, y)\}| \leq 1$$

Similarly, 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. Vice versa, there are relations that are not functions in the strict sense, but that exhibit a similar behavior. For example, *hasNationality* can give several nationalities to a person, but the vast majority of people only have one nationality. Therefore, we use the notion of *functionality* [38]. The functionality of a

relation r is a value between 0 and 1, which is 1 if r is a function:

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

where $\#x : X$ is an abbreviation for $|\{x : X \in \mathcal{K}\}|$. The inverse functionality is defined accordingly as $ifun(r) := fun(r^{-1})$, where r^{-1} denotes the inverse relation of r, that is, the relation defined by swapping the arguments of r, e.g., $actedIn^{-1} = hasActor$, therefore $ifun(actedIn) := fun(hasActor)$.

Some relations have roughly the same degree of functionality and of inverse functionality. Bijections are an example. Usually, however, fun and $ifun$ are different. Manual inspection shows that in web-extracted common sense KBs (e.g., YAGO, DBpedia) the functionality is usually higher than the inverse functionality. For example, a KB is more likely to specify *isCitizenOf* than *hasCitizen*. Intuitively, this allows us to consider a fact $r(x,y)$ as a fact about x. In the following, we will assume that for all relations r, $fun(r) \geq ifun(r)$. Whenever this is not the case, r can be replaced by its inverse relation r^{-1}. Then, $fun(r^{-1}) \geq ifun(r^{-1})$. In the following, we assume that all relations have been substituted with their inverses if their inverse functionality is larger than their functionality. This will simplify the analysis without affecting the generality of our approach.

3.3 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 $\overrightarrow{B} \Rightarrow r(x,y)$.

An *instantiation* of a rule is a copy of the rule, where all variables have been substituted by constants. 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 *citizenOf(Lisa,USA)* if the KB knows a parent of Lisa, e.g., *hasChild(Elvis,Lisa)*, who is American, e.g.,*citizenOf(Elvis,USA)*.

AMIE, like other ILP systems, does not mine general Horn Clauses, but uses a language bias (constraints to the form of the mined rules) in order to restrict the size of the search space. Language biases offer a trade-off between the expressiveness of the mined rules and the speed of the mining process. As an example, rules with 3 atoms can capture more complicated correlations than rules with 2 atoms, but come with a larger search space and thus with a much slower performance. The less restrictive the language bias is, the more expressive the rules can potentially be, the larger the search space grows, and the less tractable the search becomes.

AMIE's language bias requires rules to be *connected*. 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. The restriction to connected rules avoids mining rules with completely unrelated atoms, such as $diedIn(x,y) \Rightarrow wasBornIn(w,z)$.

AMIE also requires the rules to be *closed*. A variable in a rule is *closed* if it appears at least twice in the rule. A rule is *closed* if all its variables are *closed*. The restriction to closed rules avoids mining rules that predict merely the existence of a fact, as in $diedIn(x,y) \Rightarrow \exists z : wasBornIn(x,z)$.

AMIE omits also *reflexive rules*, i.e., rules with atoms of the form $r(x,x)$, as they are typically of less interest in real world KBs. However, unlike some other ILP systems, AMIE allows mining *recursive rules*. These are rules that contain the head relation in the body, as e.g., $isMarriedTo(x,z) \wedge hasChild(z,y) \Rightarrow hasChild(x,y)$.

3.4 Measures of Significance

Normally, data mining systems define a notion of significance or *support* for rules, which quantifies the amount of evidence for the rule in the data. If a rule applies only to a few instances, it is too risky to use it to draw conclusions. For this reason, data mining systems frequently report only rules above a given support threshold. In the following, we define this metric for AMIE's setting and introduce another notion of significance, the *head coverage*.

Support. In our context, the support of a rule quantifies the number of correct predictions in the existing data. One desired property for support is *monotonicity*, that is, the addition of more atoms and constraints to the rule should always decrease its support. As we will show in Section 5.1, such property is crucial for pruning. There are several ways to define the support: it can be the number of instantiations of a rule that appear in the KB. This measure, however, is not monotonic if we add atoms to the body. Consider, for example, the rule

$$R: livesIn(x,y) \Rightarrow wasBornIn(x,y)$$

If we add the atom $hasGender(x, male)$ to the body, the number of instantiations x, y in the KB decreases.

In contrast, if we add an atom with a fresh variable, e.g., $hasFriend(x, z)$, to the body, the number of instantiations x, y, z increases for every friend of x. This is true even if we add another atom with z to obtain a closed rule. Alternatively, we can count the number of facts in one particular body atom. Under this definition, however, the same rule can have different support values depending on the selected body atom. 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:

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

where $z_1, ..., z_m$ are the variables of the rule apart from x and y. Table 1 shows an example KB that contains only 2 relations and 5 facts. For this KB, our example rule R has support 1, because of the facts $livesIn(Adam, Paris)$ and $wasBornIn(Adam, Paris)$.

Note that the support is defined even for rules that are not yet closed. This allows for early pruning of unpromising candidate rules. For example, consider the rule

$$R\text{'}: livesIn(x, y) \Rightarrow wasBornIn(y, z)$$

This rule is obviously unpromising, because it postulates a birth place for y, which is not a person. The rule is not yet closed (x and z appear only once). Yet, it has support 0. Thus, it can be pruned away and does not need further refinement.

Head Coverage. Support is an absolute number. This means that a user defining thresholds on support has to know the absolute size of the KB to give meaningful values. Moreover, if the support threshold is higher than the size of some relation, this relation will be disregarded as head relation for rule mining. To avoid this, we propose 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. This definition, however, does not solve the problem for small relations. Therefore, we propose to use the notion of *head coverage*:

$$hc(\overrightarrow{B} \Rightarrow r(x, y)) := \frac{supp(\overrightarrow{B} \Rightarrow r(x, y))}{size(r)}$$

with $size(r) := \#(x', y') : r(x', y')$ denoting the number of facts in relation r. Head coverage quantifies the ratio of the known true facts that are implied by the rule. For the example presented in Table 1, $hc(R) = 1/2$.

livesIn	*wasBornIn*
(Adam, Paris)	(Adam, Paris)
(Adam, Rome)	(Carl, Rome)
(Bob, Zurich)	

Table 1: An example KB containing two relations between people and cities.

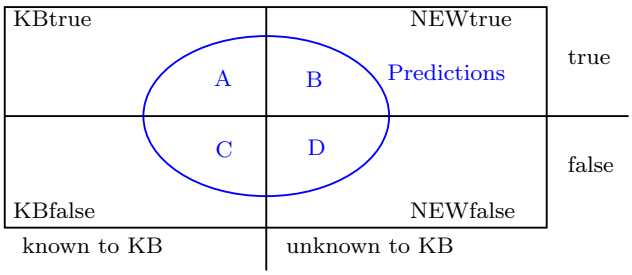
4 Confidence Measures

The support of a rule quantifies the number of known correct predictions of the rule. However, it does not take into account the false predictions of the rule. Following [16], we will now describe measures that judge the *quality* of a rule. We first describe the challenges in defining such a measure in our setting and discuss the most common way to measure the rule quality, which we call the standard confidence. Then, we introduce our own measure: the confidence under the assumption of partial completeness.

4.1 Challenges

Let us consider a given Horn rule $\overrightarrow{B} \Rightarrow r(x, y)$. Let us look at all facts with relation r (Figure 1). 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 about relation r (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.

Fig. 1: Prediction under Incompleteness



Our goal is to find rules that make true predictions that go beyond the current KB. In Figure 1, we wish to maximize the area B and to minimize the area D. There

are two obvious challenges in this 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. This is the central challenge of our setting: to provide counterexamples for the rule mining. These can take the role of KBfalse so that we can estimate the areas NEWtrue and NEWfalse. We describe two approaches to this problem: Creating counterexamples according to the Closed World Assumption (CWA) that traditional association rule mining systems apply and according to the Partial Completeness Assumption (PCA) that we propose. We will now present these approaches in detail.

4.2 The CWA and 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 (KBtrue) in the set of predictions:

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

For example, consider again the rule

$$R : livesIn(x,y) \Rightarrow wasBornIn(x,y)$$

together with the KB given in Table 1. In this case, $conf(R) = 1/3$, because (a) there is one positive example for the rule, $wasBornIn(Adam, Paris)$, and (b) the predictions $wasBornIn(Adam, Rome)$ and $wasBorn(Bob, Zurich)$ are counted as negative examples since they do not appear in the KB.

Standard confidence is the measure traditionally used in association rule mining and market basket analysis, where the Closed World Assumption (CWA) is used: if there is no evidence in any of the transactions of the database that a user bought a specific product, then this user did not buy the product. Albeit natural for the market basket analysis scenario, standard confidence fails to distinguish between "false" and "unknown" facts, which makes it inappropriate for a scenario with Open World semantics like ours. Moreover, we also pursue a different goal than market basket analysis: we aim to maximize the number of true predictions that go beyond the current knowledge, whereas market basket analysis usually tries to mine rules that can describe data that is already known.

4.3 The PCA and the PCA-Confidence

In AMIE, we generate negative examples for a rule by means of the *Partial Completeness Assumption* (PCA). The PCA 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 we know one y for a given x and r, then we know all y for that x and r. This assumption allow us to generate counter-examples in a way that is less restrictive than the standard confidence. In our example from Table 1, the PCA will assume that any other place of birth for Adam and Carl is false. Conversely, the PCA will not assume anything about the places of birth of Bob, because the KB does not know any. With this notion in mind, we redefine the definition of confidence for rules. Under the PCA, the denominator of the confidence formula is not the size of the entire set of conclusions derived from the body of the rule, but the number of facts that we know to be true together with the facts that we assume to be false.

$$conf_{pca}(\overrightarrow{B} \Rightarrow r(x,y)) := \frac{supp(\overrightarrow{B} \Rightarrow r(x,y))}{\#(x,y) : \exists z_1, ..., z_m, y' : \overrightarrow{B} \wedge r(x,y')} \quad (1)$$

This formula normalizes the support by the number of pairs (x,y) for which there exists a y' with $r(x,y')$. Consider again the KB given in Table 1 and the rule $R : livesIn(x,y) \Rightarrow wasBornIn(x,y)$. In this case, $conf_{pca}(R) = 1/2$. This is because (a) there is one positive example for the rule, $wasBornIn(Adam, Paris)$, and (b) the prediction $wasBornIn(Adam, Rome)$ is counted as negative example, because we already know a different place of birth for Adam. The prediction $wasBorn(Bob, Zurich)$ is completely disregarded as evidence, because we neither know where Bob was born nor where he was not born.

Notice that Eq. 1 fixes x and r and implies that rules will try to predict values for y. AMIE always predicts in the most functional direction. To see this, recall that it is more intuitive to predict the birthplace of a specific person than predict all the people that were born in a specific city. Since in Sec. 3.2 we re-write all relations so that their functionality is larger than their inverse functionality, the most functional direction will be always to predict y given x.

In spite of being an assumption, the PCA is certainly true for functions, such as *birthdate* and *capital*. The PCA also holds for relations that are not functions but that have a high functionality, as we shall see in our qualitative analysis of the PCA in Section 7.4. The PCA has been applied in the Google Knowledge Vault under the name "local completeness assumption" [15].

5 AMIE

We now outline the core algorithm of AMIE and its implementation. We follow the description in [16] and extend it with further explanations and details.

5.1 Algorithm

Algorithm. Algorithm 1 shows our approach to mine rules. It takes as input a KB $\mathcal{K}$, a threshold $minHC$ on the head coverage of the mined rules, a maximum rule length $maxLen$ and a threshold $minConf$ on the confidence. We discuss the choice of parameter values later in this section. The algorithm maintains a queue of rules (line 1), which initially contains all possible head atoms, that is, all rules of size 1. It then iteratively dequeues a rule from this queue. If the rule meets certain criteria (line 6), it is pushed to the output. If the rule does not exceed the maximum number of atoms $maxLen$ (line 9), it goes through a refinement process (described below) which expands the rule (the parent) to produce a set of new rules (the children). These new rules, if neither duplicates nor pruned by the head coverage threshold (line 12), are also pushed into the queue. This process is repeated until the queue is empty. In the following, we will see in more detail the different phases of the algorithm.

Refinement. One of the major challenges of rule mining is to find an efficient way to explore the search space. The naive algorithm of enumerating all possible combinations of conjunctions of atoms is infeasible for large KBs. Hence, we explore the search space by iteratively extending rules using a set of *mining operators* (line 10 of Alg. 1). 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 is a variable that 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) 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.

Note that all above operators create connected rules. 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 that 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. We will describe in Section 5.2 how these operators are executed on the KB.

Algorithm 1 Rule Mining

```
1: function AMIE(KB K, minHC, maxLen, minConf)
2:     q = [r1(x, y), r2(x, y) ... rm(x, y)]
3:     out = ⟨⟩
4:     while ¬q.isEmpty() do
5:         r = q.dequeue()
6:         if AcceptedForOutput(r, out, minConf) then
7:             out.add(r)
8:         end if
9:         if length(r) < maxLen then
10:            R(r) = Refine(r)
11:            for all rules rc ∈ R(r) do
12:                if hc(rc) ≥ minHC & rc ∉ q then
13:                    q.enqueue(rc)
14:                end if
15:            end for
16:        end if
17:    end while
18:    return out
19: end function
```

Algorithm 2 Decide whether to output a rule

```
1: function AcceptedForOutput(rule r, out, minConf)
2:     if r is not closed ∨ conf_pca(r) < minConf then
3:         return false
4:     end if
5:     parents = parentsOfRule(r, out)
6:     for all rp ∈ parents do
7:         if conf_pca(r) ≤ conf_pca(rp) then
8:             return false
9:         end if
10:    end for
11:    return true
12: end function
```

When to Output. Not every rule that the mining algorithm dequeues is output. This is because some rules may not be closed, or may not be better than rules that have already been output. Algorithm 2 explains how we decide if a rule should be output or not once it has been dequeued. The algorithm first checks if the rule is of the form described in Section 3 (i.e., closed and connected). The refinement operators used by AMIE (see Section 5.1) always produce connected rules. So, at this point, the algorithm only checks if the rule is closed. Then, the algorithm calculates the confidence of the rule and performs a quality check. The rule should have a confidence value that (i) passes the confidence threshold (line 1) and (ii) improves over the confidence of all its parents (line 7). The latter condition implies that the refinements of a rule $(B_1 \wedge ... \wedge B_n \wedge B_{n+1} \Rightarrow H)$ must bring some confidence gain with respect to the parent rule $(B_1 \wedge ... \wedge B_n \Rightarrow H)$. Since support and head coverage are monotonic metrics, we know that the child rule will never have a higher score than its parent rule. If the child rule has also lower confidence, then its quality is worse in all aspects than the parent rule. Hence, there is no reason to output it.

A rule can have several parents. For example, the rule $actedIn(x, y) \wedge directed(x, y) \Rightarrow created(x, y)$ can be derived by either adding $directed(x, y)$ to $actedIn(x, y) \Rightarrow created(x, y)$ or by adding $actedIn(x, y)$ to $directed(x, y) \Rightarrow created(x, y)$. AMIE requires a confidence gain over all parents of a rule.

Note that the decisions made at this point affect only the output. They do not influence the refinement process. i.e., a rule with low confidence can still be refined to obtain new rules. This is because confidence is a non-monotonic measure, i.e., we might get good rules with further refinement of bad rules.

Parameters and Pruning. If executed naively, Algorithm 1 will have prohibitively high runtimes. The instantiation operator $\mathcal{O}_I$, in particular, generates atoms in the order of $|\mathcal{R}| \times |\mathcal{E}|$. For this reason the algorithm defines some parameters that determine when to stop with the exploration of the space. These are the minimal head coverage $minHC$, the maximal length $maxLen$ and the minimal confidence $minConf$. Choosing larger thresholds on head coverage, and choosing a shorter maximum rule length will make the algorithm stop earlier and output fewer rules. Relaxing the values will make the algorithm output the very same rules as before, and find also rules with a smaller head coverage or a larger number of atoms. Thus, these parameters define a trade-off between the runtime and the number of rules.

Interestingly, a larger number of rules is not necessarily a good thing. For instance, a rule that covers only 1% or less of the instances of a relation is probably not interesting. It simply lacks statistical significance. Assuming that a user is not interested in such spurious rules, we set $minHC = 0.01$ by default.

Additionally, we show in our experiments that rules with more than 3 atoms tend to be very convoluted and not insightful. Hence, we set $maxLen = 3$ by default.

Likewise, rules with low confidence will not be of much use to the application. For example, a rule with confidence 10% will make correct predictions in only one out of ten cases. Assuming that a user is not interested in such kind of rules, we set $minConf = 0.1$ by default.

That being said, if the user is interested in less confident, more complex, or less supported rules, she can change these thresholds. However, we believe that there is no good reason to deviate from the default values. In particular, relaxing these values will not output better rules. This makes AMIE a system that can be run off the shelf, without the need for parameter tuning.

Duplicate Elimination. As mentioned in Section 5.1 a rule can be derived in multiple ways. For example, the rule $actedIn(x, y) \wedge directed(x, y) \Rightarrow created(x, y)$ can result from the application of the operator $\mathcal{O}_C$ to both $actedIn(x, y) \Rightarrow created(x, y)$ and $directed(x, y) \Rightarrow created(x, y)$. For this reason, AMIE checks for the existence of duplicate rules (line 12) in order to avoid queuing the same rule multiple times. While checking two rules for equality is expensive (it is a graph isomorphism verification task), we observe that two rules can only be equal if they have the same head relation, the same number of atoms and the same head coverage (or support). This reduces drastically the set of rules that have to be checked and therefore the time invested in this task.

Multithreading. To speed up the process, our implementation parallelizes Algorithm 1, that is, the main loop (lines 4 to 17) runs in multiple threads. This is achieved by synchronizing the access to the centralized queue from which the threads dequeue and enqueue and the access to the output.

5.2 Count Projection Queries

AMIE tries to expand a given rule by applying all mining operators defined in the last section (one each time). We now explain how the operators are implemented and executed on a KB.

Count Projection Queries. Assume that AMIE needs to add the atom $r(x, y)$ to a rule. For efficiency reasons, we do not blindly try all possible relations in the place of r. Instead, we first find all relations that

lead to a new rule that passes the head-coverage threshold. In other words, we first fire a *count projection query* of the form

$$\text{SELECT } \boldsymbol{r}, \text{ COUNT}(H)$$
$$\text{WHERE } H \wedge B_1 \wedge ... \wedge B_{n-1} \wedge \boldsymbol{r}(\boldsymbol{X}, \boldsymbol{Y})$$
$$\text{SUCH THAT COUNT}(H) \geq k$$

where $k := minHC \times size(H)$ (see Section 3.4) is the translation of the head coverage threshold into an absolute support threshold and the expression COUNT$(\cdot)$ has COUNT(DISTINCT $\cdot$) semantics (also for the rest of this section). $\boldsymbol{X}$ and $\boldsymbol{Y}$ represent variables that are either fresh or already present in the rule. The results for $\boldsymbol{r}$ are the relations that, once bound in the query, ensure that the head coverage of the rule $B_1 \wedge ... \wedge B_{n-1} \wedge \boldsymbol{r}(\boldsymbol{X}, \boldsymbol{Y}) \Rightarrow H$ is greater or equal than $minHC$. Notice also that for each value of $\boldsymbol{r}$, the expression COUNT(H) gives us the support of the new rule. We now discuss the instantiation of this query for all three operators.

Dangling Atom Operator. As an example, assume that Algorithm 1 dequeues the following intermediate non-closed rule for further specialization:

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

The application of the operator $\mathcal{O}_{\mathcal{D}}$ will fire queries of the form:

$$\text{SELECT } \boldsymbol{r}, \text{ COUNT}(livesIn(x, y)) \text{ WHERE}$$
$$livesIn(x, y) \wedge marriedTo(x, z) \wedge \boldsymbol{r}(\boldsymbol{X}, \boldsymbol{Y})$$
$$\text{SUCH THAT COUNT}(livesIn(x, y)) \geq k$$

with

$$\boldsymbol{r}(\boldsymbol{X}, \boldsymbol{Y}) \in \{\boldsymbol{r}(x, w), \boldsymbol{r}(z, w), \boldsymbol{r}(w, x), \boldsymbol{r}(w, z)\}$$

That is, $\boldsymbol{r}(\boldsymbol{X}, \boldsymbol{Y})$ binds to each possible join combination of a new dangling atom, where w is an arbitrary fresh variable. For intermediate rules, dangling atoms are joined on the non-closed variables; z and y in this example. If the rule is closed, dangling atoms are joined on all the variables appearing in the rule.

Closed Atom Operator. The $\mathcal{O}_{\mathcal{C}}$ operator works in the same fashion. In our example, the atom $\boldsymbol{r}(\boldsymbol{X}, \boldsymbol{Y})$ can take values in $\{\boldsymbol{r}(z, y), \boldsymbol{r}(y, z)\}$. The method will produce new atoms so that all open variables are closed. In this example, the method produces the minimum number of specializations required to close the variables y and z. If there is only one closed variable, the method will produce atoms between the open variable and all the other variables. If the rule is already closed, the operator tries with all possible pairs of variables in the rule.

Instantiated Atom Operator. The operator $\mathcal{O}_{\mathcal{I}}$ is implemented in two steps. We first apply the operator $\mathcal{O}_{\mathcal{D}}$ to produce a set of intermediate rules with a new dangling atom and a new fresh variable. Then for each rule, we fire a count-projection query on the fresh variable. This step provides bindings for one of the arguments of the relation. For instance, the application of the $\mathcal{O}_{\mathcal{I}}$ operator to our example rule

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

will first add all possible dangling atoms to the rule. Let us consider one group of such atoms, e.g., those of the form $\boldsymbol{r}(x, w)$. Then for each value of $\boldsymbol{r}$ that keeps the rule above the head coverage threshold $minHC$, the algorithm tries to find the best bindings for w. For example, imagine we bind $\boldsymbol{r}$ to the relation $citizenOf$. The second step will fire a query of the form:

$$\text{SELECT } \boldsymbol{w}, \text{ COUNT}(livesIn(x, y)) \text{ WHERE}$$
$$livesIn(x, y) \wedge marriedTo(x, z) \wedge citizenOf(x, \boldsymbol{w})$$
$$\text{SUCH THAT COUNT}(livesIn(x, y)) \geq k$$

Each binding of $\boldsymbol{w}$ forms a new rule that will be enqueued and later evaluated for output.

Count-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 above $minHC$. We discuss how to implement count projection queries efficiently in Section 5.3.

5.3 Query Implementation Details

In-Memory Database. We have shown [16] that count projection queries translate into very inefficient queries in both SPARQL and SQL. Therefore, we have implemented an in-memory database that is specifically geared towards this type of queries. Our implementation indexes the facts aggressively with one index for each permutation of the columns subject (`S`), relation (`R`), and object (`O`). This means that there are six indexes, namely `SRO`, `SOR`, `RSO`, `ROS`, `OSR` and `ORS`. We call them *fact indexes*. Each fact index is a hash table, which maps elements of the first column to a nested hash table. This nested hash table maps elements of the second column to a set of elements of the third column. For example, the index `ORS` has as keys the objects of all triples in the KB. It maps each object o to a hash table. This hash table has as keys all possible relations of the KB. It maps each relation r to a set of subjects $\{s_1, ..., s_n\}$, such that $r(s_i, o)$ for $i = 1...n$. Fact indexes allow us to check the existence of a triple in constant time. They also allow us to efficiently fetch the instantiations of an atom.

In addition to the fact indexes, our database relies on three *aggregated indexes* `S`, `P`, `O`. These store the aggregated number of facts for each key of the fact indexes. For example, the aggregated index `P` stores the

number of triples for each relation in the KB, whereas the aggregated index `S` stores the number of triples where each entity appears as subject.

Size Queries. Fact indexes in combination with aggregated indexes can be used to determine the *size of an atom* ($size(a, \mathcal{K})$), i.e., its number of bindings in the KB $\mathcal{K}$. For example, the size of the atom $livesIn(x, y)$ can be retrieved by a simple look-up in the aggregated index `P`. The size of the atom $livesIn(x, USA)$ requires two lookups in the fact index `ROS`: the first lookup to get the object values of $livesIn$ and the second to retrieve the list of subjects for the object value USA.

Existence Queries. One of the central tasks of the in-memory database is to determine whether there exists a binding for a conjunctive query. Algorithm 3 shows how this can be implemented. The algorithm requires as input a conjunctive query and a KB $\mathcal{K}$. If the query is a single atom (Line 3), we can directly verify if its size is greater than zero using the indexes (Line 4). Otherwise, we select the atom B_s with fewest instantiations using the indexes (Line 6), and run through all of its instantiations (Lines 8 to 13). We apply such instantiations to the remaining atoms (Line 9) and repeat this process recursively (Line 10) until we end up with a single atom. Since rules are connected query patterns, the atom B_s must share at least one variable with the remaining atoms. This means that by instantiating B_s, some variables in the remaining atoms become instantiated, making the atoms more selective with every recursive step.

Algorithm 3 Existence Queries

```
1: function EXISTS(B_1 ∧ ... ∧ B_n, K)
2:     q := B_1 ∧ ... ∧ B_n
3:     if n = 1 then
4:         return size(B_1, K) > 0
5:     else
6:         s := argmin_i {size(B_i, K)}
7:         q := q \ {B_s}
8:         for all instantiations b_s ∈ B_s do
9:             q' := q instantiated with bindings from b_s
10:            if Exists(q', K) then
11:                return true
12:            end if
13:        end for
14:    end if
15:    return false
16: end function
```

Select Queries. Algorithm 4 describes the implementation of SELECT DISTINCT queries on one projection variable for a conjunction of atoms. The algorithm starts finding the atom with the fewest number of instantiations B_s. If the projection variable x is in B_s (Lines 5 to 11), the algorithm goes through all the instantiations $\hat{x}$ of x, instantiates the query accordingly and checks whether there exists a solution for the instantiated query pattern in the KB (Line 8). If there is, the solution $\hat{x}$ is added to the result set. In contrast, if the projection variable is not in the most restrictive atom B_s (Lines 13 to 17), the algorithm iterates through the instantiations of B_s and recursively selects the distinct bindings of x in the remaining atoms (Line 16).

Algorithm 4 Select Distinct Queries

```
1: function SELECT(x, B_1 ∧ ... ∧ B_n, K)
2:     q := B_1 ∧ ... ∧ B_n
3:     s := argmin_i {size(B_i, K)}
4:     result := {}
5:     if x ∈ B_s then
6:         for all instantiations x̂ ∈ x do
7:             q' := q instantiated with x̂ for x
8:             if Exists(q', K) then
9:                 result.add(x̂)
10:            end if
11:        end for
12:    else
13:        q := q \ {B_s}
14:        for all instantiations b_s ∈ B_s do
15:            q' := q instantiated with bindings from b_s
16:            result.add(Select(x, q', K))
17:        end for
18:    end if
19:    return result
20: end function
```

Count Queries. To compute the confidence of a rule $\overrightarrow{B} \Rightarrow r(x, y)$, AMIE must fire a *count query* to estimate the denominator of the confidence formula. For the PCA confidence, such queries have the form:

$$\text{SELECT COUNT}(x, y) \text{ WHERE } r(x, y') \wedge \overrightarrow{B}$$

where x, y are the variables in the head atom of the rule, $\overrightarrow{B} = B_1, \ldots, B_n$ are the body atoms, and $r(x, y')$ is a variant of the head atom where the least-functional variable has been replaced by a fresh variable y' (see Section 4.3). These queries return the number of distinct bindings of the head variables that fulfill the pattern $r(x, y') \wedge \overrightarrow{B}$. They are used to calculate the confidence of rules. The in-memory database first fires a SELECT query on variable x:

$$\text{SELECT DISTINCT } x \text{ WHERE } r(x, y') \wedge \overrightarrow{B}$$

Then, for each binding of x, it instantiates the query and fires another select query on variable y, adding up the number of instantiations.

Count Projection Queries. Count projection queries take the form

$$\text{SELECT } \boldsymbol{x}, \text{ COUNT}(H) \text{ WHERE } H \wedge B_1 \wedge \ldots \wedge B_n \text{ SUCH THAT COUNT}(H) \geq k$$

These are the types of queries used to determine the relations and instances for new atoms in the refinement phase of AMIE. Algorithm 5 shows how we answer these queries. The algorithm takes as input a selection variable $\boldsymbol{x}$, a projection atom $H := R(\boldsymbol{X}, \boldsymbol{Y})$, remaining atoms $B_1, ...B_n$, the threshold k, and a KB $\mathcal{K}$. The algorithm returns a hash table with each instantiation of the selection variable $\boldsymbol{x}$ as key and the number of distinct bindings of the projection atom H as value.

We first check whether $\boldsymbol{x}$ appears in the projection atom (Line 3). If that is the case (Lines 4 to 10), we run through all instantiations of the projection atom, instantiate the query accordingly (Line 6), and check for existence (Line 7). Each existing instantiation increases the counter for the respective value of the selection variable $\boldsymbol{x}$ (Line 8). If the selection variable does not appear in the projection atom (Lines 12 to 18), we iterate through all instantiations of the projection atom. We instantiate the query accordingly, and fire a SELECT DISTINCT query for $\boldsymbol{x}$ (Line 14). We then increase the counter for each value of $\boldsymbol{x}$ (Line 16).

Algorithm 5 Count Projection Queries

```
1: function SELECT(x, R(X,Y) ∧ B1 ∧ ... ∧ Bn, k, K)
2:     map = {}
3:     q = B1 ∧ ... ∧ Bn
4:     if x ∈ {R, X, Y} then
5:         for all instantiations r(x,y) ∈ R(X,Y) do
6:             q' := q, replace R by r, X by x, Y by y
7:             if Exists(q', K) then
8:                 map[x] + +
9:             end if
10:        end for
11:    else
12:        for all instantiations r(x,y) ∈ R(X,Y) do
13:            q' := q, replace R by r, X by x, Y by y
14:            X := Select(x, q', K)
15:            for all x ∈ X do
16:                map[x] + +
17:            end for
18:        end for
19:    end if
20:    map := {⟨x → n⟩ ∈ map : n ≥ k}
21:    return map
22: end function
```

6 Scalability Improvements: AMIE+

Since the publication of the original AMIE framework [16], we have extended it with a series of improvements that allow the system to run over very large KBs. In the following, we will introduce and discuss these extensions and refer to this new version of AMIE as AMIE+. Our extensions aim to speed up 2 different parts of the main rule-mining algorithm: (i) the refinement phase and (ii) the confidence evaluation.

6.1 Speeding Up Rule Refinement

In this section, we will discuss how AMIE+ speeds up the rule refinement phase for specific kinds of rules. We emphasize that the techniques described below do not alter AMIE's output in any way.

Maximum Rule Length. The maximum rule length $maxLen$ is an input parameter for our system. AMIE stops exploring the search space as soon as all rules with a length of at most $maxLen$ have been produced. During the mining process, AMIE creates connected rules by applying all possible mining operators (line 10 in Algorithm 1) on previously created rules. Given a maximum rule length $maxLen$ and a non-closed Horn rule of length $maxLen - 1$, AMIE+ will refine it only if it is possible to close it before exceeding the length constraint. This means that for a not-yet-closed rule of length $maxLen - 1$, AMIE+ will not apply the add-dangling-atom operator $\mathcal{O}_D$, because this results in a non-closed rule, which will be neither output nor refined. In the same spirit, if the same rule contains more than two non-closed variables (see Section 3.3), AMIE+ will skip the application of the add-closing atom operator $\mathcal{O}_C$. This happens because an application of the operator $\mathcal{O}_C$ can close at most two variables with one atom. This reasoning also applies to the instantiation operator $\mathcal{O}_I$: rules with more than one non-closed variable are not refined with instantiated atoms, because the addition of an instantiated atom can close at most one variable.

Perfect Rules. By definition, a rule cannot achieve a PCA confidence that is higher than 100%. Thus, once a rule has achieved 100% PCA confidence, we can stop adding new atoms. This is because the confidence cannot increase and the support can only decrease. Hence, any refinement is futile and will be discarded by the output routine described in Algorithm 2. We call rules with 100% PCA confidence *perfect rules*.

Simplifying Projection Queries. Support is monotonically decreasing with the length of the rule (Section. 3.4). Hence, whenever we apply an add-dangling-atom operator to a rule R_p (the parent rule) to produce a new rule R_c (the child rule), the support of R_c will likely be smaller than the support of R_p. However, there is one case in which the addition of a dangling atom cannot reduce the support. This happens when R_c (i) already contains atoms with the same relation as the dangling atom and (ii) these atoms have a variable in common with the dangling atom. An example is the parent rule $R_p : livesIn(x, y) \Rightarrow citizenOf(x, y)$ and

the child rule R_c : $citizenOf(z, y) \wedge livesIn(x, y) \Rightarrow citizenOf(x, y)$. Intuitively, the addition of the dangling atom $citizenOf(z, y)$ cannot further restrict the support of R_p because the new atom is a less restrictive version of the atom $citizenOf(x, y)$. This means that z will always bind to the same values as x. From this observation, it follows that the support of R_c can be rewritten as

$$supp(R_c) = \#(x, y) : citizenOf(x, y) \wedge livesIn(x, y) \wedge citizenOf(x, y)$$

$$supp(R_c) = \#(x, y) : livesIn(x, y) \wedge citizenOf(x, y)$$

which is the same as $supp(R_p)$. Thus both R_p and R_c have the same support. This observation can be leveraged to speed up projection queries. The query for $supp(R_p)$ has one fewer join and thus executes faster.

6.2 Speeding up Confidence Evaluation

Confidence Scores. A significant part of the runtime of our algorithm is spent on computing confidence scores (up to 35% in our experiments). The reason is that the calculation of confidence (both PCA and standard) requires the calculation of the number of instantiations of the rule body. If the body contains atoms with many instantiations, the joins can be very expensive to compute.

At the same time, we will not output rules with a confidence below the threshold $minConf$ (Section 5.1). This means that the system might spend a significant amount of time evaluating expensive confidence queries only to find out that the rule was of low confidence and will not be output. An example of such a rule (which we will also use later in this section) is:

$$directed(x, z) \wedge hasActor(z, y) \Rightarrow married(x, y)$$

This rule concludes that a director is married to all the actors that acted in his/her movies, producing a total of 74249 married couples in YAGO2. AMIE needs more than 500ms (more than twice the average cost: 200ms) to calculate the confidence of this intuitively bad rule.

Approximation. We have developed a method to approximate the confidence value of such a rule very quickly. Our approximation is based on statistics, such as the functionalities of the atoms, or the size of the joins between two relations. We pre-compute these quantities, so that they can be accessed in constant time. As a result, AMIE+ prunes the example-rule above in less than 1ms.

Our approximation is designed such that it is more likely to overestimate confidence than to underestimate it. This is important, because we use it to prune rules, and we want to avoid pruning rules that have a higher confidence in reality. Our experiments (see Section 7.2) show that this technique prunes only 4% of the rules erroneously. In return, it makes AMIE+ run in the range of minutes instead of days. It is thus one of the main techniques that allow AMIE+ to run on large-scale KBs.

In Section 6.2.1, we give an overview of the confidence approximation and we explain for which form of rules we use it. Section 6.2.2 describes how the size of the rule's body is approximated. Section 6.2.3 discusses the underlying assumptions made by our approximation and explains how it is used within AMIE+. Finally, Section 6.2.4 derives upper bounds for the confidence of a particular class of rules.

6.2.1 Confidence Approximation

Computing Confidence. Recall that confidence and PCA confidence (see Sections 4.2 and 4.3) are defined as:

$$conf(\overrightarrow{B} \Rightarrow r_h(x, y)) := \frac{supp(\overrightarrow{B} \Rightarrow r_h(x, y))}{\#(x, y) : \exists z_1, \ldots, z_m : \overrightarrow{B}}$$

and

$$conf_{pca}(\overrightarrow{B} \Rightarrow r_h(x, y)) := \frac{supp(\overrightarrow{B} \Rightarrow r_h(x, y))}{\#(x, y) : \exists z_1, \ldots, z_m, y' : \overrightarrow{B} \wedge r_h(x, y')}$$

By the time AMIE has to calculate the confidence of a rule, the system already knows the support of the rule. Hence, the remaining step is to fire the queries for the denominators of the confidence expressions (see Sections 4.2 and 4.3). We denote them by d_{std} and d_{pca}:

$$d_{std}(\overrightarrow{B} \Rightarrow r_h(x, y)) := \#(x, y) : \exists z_1, \ldots, z_m : \overrightarrow{B} \quad (2)$$

$$d_{pca}(\overrightarrow{B} \Rightarrow r_h(x, y)) := \#(x, y) : \exists z_1, \ldots, z_m, y' : \overrightarrow{B} \wedge r_h(x, y') \quad (3)$$

Our aim is to derive a conservative approximation for d_{pca} (or d_{std}) denoted by $\widehat{d}_{pca}$. By plugging this expression into the confidence formula, we get

$$\widehat{conf}_{pca}(R) := \frac{supp(R)}{\widehat{d}_{pca}(R)} \quad (4)$$

Let us reconsider Eq. 3 and rewrite it as follows:

$$d_{pca}(\overrightarrow{B}(x, y) \Rightarrow r_h(x, y)) := \#(x, y) : \exists z_1, \ldots, z_m, y' : \overrightarrow{B}(x, y) \wedge r_h(x, y')$$

Here, we resort to an abstraction that treats the body of the rule $\overrightarrow{B}(x, y)$ as a relation on the head variables. If $\overrightarrow{B}$ has functionality $fun(\overrightarrow{B})$, this means that, on average, each entity in variable x relates to $\#y_{per\ x} = 1/fun(\overrightarrow{B})$ bindings in y. If we denote the domain and range of a relation r as $dom(r)$ and $rng(r)$ respectively, the expression $\#y_{per\ x} \cdot |dom(\overrightarrow{B})|$ gives us an estimate for the body size of the rule, i.e., $d_{std}(\overrightarrow{B} \Rightarrow r_h(x, y))$. However, for the PCA confidence, the denominator is restricted also by the entities in the domain of the head relation. This consideration leads us to the expression:

$$\widehat{d}_{pca}(R) := |dom(\overrightarrow{B}) \cap dom(r_h)| \cdot \#y_{per\ x} \quad (5)$$

In the following, we first describe for which kind of rules it makes sense to use this approximation and then, in Section. 6.2.2, we discuss how to calculate the terms of Equation 5 in an efficient way.

When to Use the Approximation. Using any form of confidence approximation always involves the risk of pruning a good rule. At the same time, if the exact confidence value is cheap to compute, the potential gain of using an approximation is small. For this reason, we only use the confidence approximation for rules whose exact confidence is relatively "expensive" to compute. These rules typically have a large number of bindings in the body because of the presence of *intermediate variables*. This translates into higher runtimes and memory usage. An example is the rule we saw before:

$$directed(x, z) \wedge hasActor(z, y) \Rightarrow married(x, y)$$

In this example, a director x is related to many movies z (the intermediate variable) that have different actors y. Hence, we consider a rule *expensive* if its body (i) contains variables other than the variables appearing in the head atom (z in our example) and (ii) if these additional variables define a single path between the head variables ($x \rightarrow z \rightarrow y$ in our example). Note that rules without intermediate variables (such as $livesIn(x, y) \wedge bornIn(x, y) \Rightarrow diedIn(x, y)$) or that contain multiple paths between the head variables (such as $livesIn(x, z_1) \wedge locatedIn(z_1, y) \wedge bornIn(x, z_2) \wedge locatedIn(z_2, y) \Rightarrow isCitizenOf(x, y)$) are usually associated with more selective queries. In these examples, both $livesIn$ and $bornIn$ join on x in the body and restrict the size of the result.

We therefore use the confidence approximation only for rules where the head variables x, y are connected through a single chain of existentially quantified variables $z_1, \ldots, z_{n-1}$. These rules have the form:

$$r_1(x, z_1) \wedge r_2(z_1, z_2) \wedge \ldots \wedge r_n(z_{n-1}, y) \Rightarrow r_h(x, y)$$

In order to write a rule in this canonical form, we may have to replace some relations by their inverses (i.e., substitute $r_2(z_2, z_1)$ with $r_2^{-1}(z_1, z_2)$) and change the order of the atoms.

We will now see how to compute the approximation for this type of rules.

6.2.2 Computing the Approximation

In the following, we denote the domain and range of a relation r by $dom(r)$ and $rng(r)$, respectively. In addition, we use the shortcut notations $ov_{dr}(r_1, r_2)$, $ov_{rd}(r_1, r_2)$, $ov_{dd}(r_1, r_2)$, $ov_{rr}(r_1, r_2)$ for the size of the overlap sets between the domains and ranges of pairs of relations. For example,

$$ov_{dr}(r_1, r_2) := |dom(r_1) \cap rng(r_2)|$$

Let us now consider again the rule

$$directed(x, z) \wedge hasActor(z, y) \Rightarrow married(x, y)$$

which implies that a director is married to all actors that acted in his movies. In this case, $d_{pca}(R)$ is defined as

$$d_{pca}(R) := \#(x, y) : \exists\, z, y' : directed(x, z) \wedge hasActor(z, y) \wedge isMarried(x, y')$$

Here $\overrightarrow{B}(x, y) = directed(\boldsymbol{x}, z) \wedge hasActor(z, \boldsymbol{y})$. To calculate the approximation defined in Equation 5, we need to calculate the number of directors in $\overrightarrow{B}$ that are married, i.e., $|dom(\overrightarrow{B}) \cap dom(isMarried)|$ and the number of actors y associated to each director x, i.e., $\#y_{per\ x}$. We focus on the latter term. This requires us to walk from the most to the least functional variable, i.e., through the path $x \rightarrow z \rightarrow y$, connecting a director to his potential actors. If $fun(r)$ and $ifun(r)$ denote the functionality and inverse functionality of the relation r, respectively, then walking through this path involves the following steps:

1. For each director x, the relation *directed* will produce on average $\frac{1}{fun(directed)}$ movies z.
2. Some or all of these movies z will find join partners in the first argument of *hasActor*.
3. For each movie z, *hasActor* will produce on average $\frac{1}{fun(hasActor)}$ actors y.
4. Each of these actors in y acted on average in $\frac{1}{ifun(hasActor)}$ movies of the *hasActor* relation.

Up to step 2, we can approximate the number of distinct movies that bind to the variable z for each director in the variable x as:

$$\#z_{per\ x} := \frac{ov_{rd}(directed, hasActor)}{|rng(directed)| \times fun(directed)}$$

Here, $|rng(directed)|$ is the number of distinct movies in the range of *directed* and $ov_{rd}(directed, hasActor)$ denotes the distinct movies in the overlap between the objects of *directed* and the subjects of *hasActor*. The term $\frac{1}{fun(directed)}$ corresponds to step 1. Our join estimation assumes that the movies in the overlap of *directed* and *hasActor* are uniformly distributed among the different directors in *directed*.

For steps 3 and 4, we can approximate the number of actors in the variable y for each movie in the variable z as follows:

$$\#y_{per\ z} := \frac{ifun(hasActor)}{fun(hasActor)}$$

The term $\frac{1}{fun(hasActor)}$ corresponds to Step 3. At the end of this step, we already have, for a single director x, a bag of actors y associated to him. However, these are not necessarily distinct actors, since x and y are connected through the variable z (movies). Therefore, a duplicate elimination step is needed. To see why, assume that each director has directed on average 3 movies and that each movie has 5 actors. Then, the rule will produce on average 15 actors y for each director x. However, there is no guarantee that these actors are distinct. If the director trusts specific actors and collaborates repeatedly with them in some or all of his movies, there will be less than 15 distinct actors. The term $ifun(hasActor)$ achieves this duplicate elimination: since each actor participated in $\frac{1}{ifun(hasActor)}$ different movies, the actor contributes to the final count with a weight that is inversely proportional to this number.

In this way of performing duplicate elimination, a single actor y belongs to $\frac{1}{ifun(hasActor)}$ different movies z, which are chosen from *all* the movies in the relation *hasActor*. In reality, we want the number of different movies to be chosen from those that remain after Step 2, i.e., the average number of movies by the same director that an actor acts in. This number is obviously smaller, which implies that the factor $ifun(hasActor)$ is a pessimistic estimator. This makes our approximation an underestimation of the real confidence denominator, and the overall confidence approximation an overestimation of the actual confidence.

With all that said, we can estimate the number of actors y that are supposed to be married with each director x as:

$$\#y_{per\ x} := \#z_{per\ x} \times \#y_{per\ z}$$

To calculate $\widehat{d}_{pca}$ of Eq. 5, we are now only missing the expression $|dom(\overrightarrow{B}) \cap dom(isMarried)|$. Here we make the simplifying assumption that $dom(\overrightarrow{B}) = dom(directed)$, so that the expression becomes the size of the join between the relations *directed* and *married*, on the subject argument, i.e., $ov_{dd}(directed, married)$.

To summarize, the factor $\widehat{d}_{pca}(R)$ for a rule $r_1(x,z) \wedge r_2(z,y) \Rightarrow r_h(x,y)$ can be approximated by:

$$\widehat{d}_{pca}(R) := \frac{ov_{dd}(r_1, r_h) \cdot ov_{rd}(r_1, r_2) \cdot ifun(r_2)}{fun(r_1) \cdot |rng(r_1)| \cdot fun(r_2)}$$

For the more general case of a rule that contains $n-1$ existential variables forming a single path from x to y

$$r_1(x, z_1) \wedge r_2(z_1, z_2) \wedge ... \wedge r_n(z_{n-1}, y) \Rightarrow r_h(x, y)$$

the formula becomes:

$$\widehat{d}_{pca}(R) := \frac{ov_{dd}(r_1, r_h)}{fun(r_1)} \times \prod_{i=2}^{n} \frac{ov_{rd}(r_{i-1}, r_i)}{|rng(r_{i-1})|} \frac{ifun(r_i)}{fun(r_i)}$$

6.2.3 Discussion

Application. We precompute the functionalities, the inverse functionalities, and the overlaps between the domains and ranges of each pair of relations when the KB is loaded into the in-memory database. This results in longer loading times, but pays off easily during rule mining. The sizes of the ranges of the relations are given by our indexes in constant time. After this preprocessing, the approximation of the confidence can be computed as a simple product of precomputed values without actually firing a single query. We apply the approximation only if the query is expensive (see Section 6.2.1). If the approximated value is smaller than the threshold, we abandon the rule. Otherwise, we compute the exact PCA confidence and proceed as usual.

Assumptions. Our approximation makes a series of assumptions. First, we make use of functionalities as average values. In other words, we assume that for any relation all objects are uniformly distributed among the subjects (which corresponds to a zero variance). In reality, this is not always the case. Additionally, the estimation of the expression $\#z_{per\ x}$ uses the term $\frac{ov_{rd}(r_1,r_2)}{|rng(r_1)|}$. This term assumes that the entities in the overlap are uniformly distributed among the entities in the range of r_1. This also introduces some error that depends on the variance of the real distribution. Nevertheless, the duplicate elimination largely underestimates the count of $\#y_{per\ x}$, and therefore we expect our approximation to usually result in an overestimation of the actual confidence. This is indeed the case, as our experiments in Section 7.2 show.

6.2.4 Confidence Upper Bounds

In some particular cases, we can derive lower bounds for the confidence denominator (d_{pca}, d_{std}) instead of using the approximation described in Section 6.2.2. Consider a rule of the form:

$$r(x, z) \wedge r(y, z) \Rightarrow r_h(x, y)$$

Here, the confidence denominator is given by

$$d_{std} := \#(x, y) : \exists z : r(x, z) \wedge r(y, z)$$

Since both atoms contain the same relation, we know that all the entities of z in the first atom will join with the second atom. Furthermore, we know that the join will result in *at least* one y-value for each binding of x, i.e., the case where $y = x$. This allows us to deduce

$$d_{std} \geq \#(x, x) : \exists z : r(x, z) \wedge r(x, z)$$

$$d_{std} \geq \#x : \exists z : r(x, z) \quad (6)$$

This expression can be calculated in constant time with the indexes of our in-memory database (Section 5.3). Similar analyses can be used for rules of the form $r(z, x) \wedge r(z, y) \Rightarrow r_h(x, y)$.

The same reasoning applies to the denominator of the PCA confidence, yielding

$$d_{pca} \geq \#x : \exists\, z, y' : r(x, z) \wedge r_h(x, y') \quad (7)$$

Although this expression requires to fire a query, it contains fewer atoms than the original expression and counts instances of a single variable instead of pairs. It is therefore much cheaper than the original query.

Both Inequalities 6 and 7 define lower bounds for the number of pairs in the denominator expressions of the standard and the PCA confidence, respectively. Thus, AMIE+ uses them to upper-bound the respective confidence scores. If the upper bound is below the threshold, the rules can be pruned even before computing the approximate confidence denominator.

7 Experiments

We conducted four groups of experiments. In the first round (Section 7.2) we compared AMIE with AMIE+. We show the performance gain carried by each of the new techniques presented in Section 6. In the second group of experiments (Section 7.3), we compared AMIE+ to competitor systems. The comparison was based on runtime and prediction quality of the rules. In the third round of experiments (Section 7.4), we investigated the Partial Completeness Assumption (PCA). We evaluated how often the PCA actually holds in a real-world KB (YAGO). Finally, in Section 7.5, we conducted a proof of concept to show how the rules mined by AMIE can be used to make predictions. We compared the performance of the PCA confidence with the performance of the standard confidence for this purpose. We also showed how post-processing of the results can increase the precision of our predictions.

Our experimental results show that:

1. The optimizations implemented in AMIE+ allow us to run on KBs with more than 1K relations and 10M facts in a matter of minutes – while AMIE took more than one day for them.
2. AMIE, and in particular AMIE+, outperforms competing systems by a large margin in terms of the quality and the quantity of the mined rules.
3. The PCA is often a valid assumption, even for relations that are not strictly functional.
4. Type constraints can improve the precision of the predictions made by rules to about 70%.

KB	Facts	Subjects	Relations
YAGO2 core	948K	470K	32
YAGO2s	4.12M	1.65M	37
DBpedia 2.0	6.70M	1.38M	1595[4]
DBpedia 3.8	11.02M	2.20M	650
Wikidata	8.4M	4.00M	431

Table 2: Knowledge bases used to test AMIE and AMIE+.

7.1 Experimental Setup

Hardware. All experiments were run on a server with 48GB of RAM, 8 physical CPUs (Intel Xeon at 2.4GHz, 32 threads) and using Fedora 21. All rules and all experimental results are available at http://www.mpi-inf.mpg.de/departments/ontologies/projects/amie/.

Datasets. We ran our experiments on different KBs. Table 2 shows a summary of the KBs used for our experiments. In all cases, we removed all facts with literals (numbers and strings). This is because literal values (such as geographical coordinates) are shared by only very few entities, which makes them less interesting for rule mining. For both DBpedia 2.0 and 3.8, we used the person data and mapping-based properties datasets. For Wikidata, we used a dump from December 2014, available for download at http://tools.wmflabs.org/wikidata-exports/rdf/exports/20140420/.

[4] Relations with more than 100 facts only.

Settings. In their default settings, AMIE and AMIE+ use a 1% head coverage threshold (i.e., $minHC = 0.01$), and a maximum of 3 atoms for rules (i.e., $maxLen = 3$, see Section 5.1). By default, AMIE does not impose a confidence threshold, i.e., $minConf = 0$. In contrast, AMIE+ defines a PCA confidence threshold of 0.1, i.e., $minConf = 0.1$ (Section 6.2). Unless explicitly mentioned, the instantiation operator $\mathcal{O}_I$ was disabled. ("without constants"). Both systems use as many threads as available logical cores in the system (32 in our hardware platform). Any deviation from these settings will be explicitly stated. Whenever AMIE and AMIE+ behave equivalently, we will refer to these systems jointly as AMIE(+).

Metrics. We compared AMIE and AMIE+ in terms of quality and runtime to two popular state-of-the-art systems: WARMR [13,14] and ALEPH[3]. To have an equal basis for the comparison with these systems, we made AMIE(+) simulate their metrics. AMIE(+) can threshold on support, head coverage, standard confidence, and PCA confidence, and can rank by any of these. She can also deviate from the default setting and count support on one of the head variables (like WARMR). In that case, AMIE(+) counts on the most functional variable of the relation (see Section 3.2 about functions). Again, any such deviation from the default behavior will be mentioned explicitly.

7.2 AMIE vs. AMIE+

In this section, we discuss the runtime improvements of AMIE+ over the previous version AMIE. Let us first discuss only AMIE. Recall from Section 5.1 that the AMIE algorithm consists of three main phases:

- Refinement (i.e., rule expansion).
- Output, which includes confidence calculation.
- Duplicate elimination.

Table 3 shows the proportion of time spent by AMIE in each phase when running on YAGO2 – first without constants and then with constants. We observe that the refinement and output phases dominate the system's runtime. When constants are not enabled, most of the time is spent in the refinement phase. In contrast, the addition of the instantiation operator increases the number of rules and therefore the time spent in the output and duplicate elimination phases. In both cases, the duplicate elimination is the least time-consuming phase. The enhancements introduced by AMIE+ aim at reducing the time spent in the refinement and output phases.

Dataset	Rules	Refinement	Output	Dup. elim.
YAGO2	135	87.48%	8.79%	3.74%
YAGO2 (c)	19132	53.54%	35.64%	10.82%

Table 3: Time spent in the different phases of the AMIE algorithm on YAGO2, first without the instantiation operator and then with this operator.

Runtime Comparison. Table 4 shows the runtimes of AMIE and AMIE+. We set a threshold of 0.1 PCA Confidence for AMIE to make it comparable with AMIE+. For the latter, we show the results in several categories:

1. Only output: only the improvements affecting the output process are active, i.e., the confidence approximation and the confidence upper bounds, both with confidence threshold 0.1 (Section 6.2).
2. Only refinement: only the improvements affecting the refinement process (Section 6.1) are active, namely the maximum rule length (MRL), the query rewriting (QRW) and the perfect rules (PR).
3. Output + MRL/QRW/PR: the output improvements and one of the refinement improvements are active.
4. Full: All improvements are active.

We first note that AMIE is not able to finish within a day for YAGO2s, DBPedia 2.0, DBpedia 3.8, and Wikidata. In contrast, AMIE+ can mine rules on all these datasets in a matter of hours, and even minutes. For YAGO2 (const), we can see that the full version of AMIE+ is 3.8x faster than AMIE. For YAGO2, this speed-up nearly doubles to 6.7x. This boost is mainly due to the improvements in the refinement process: AMIE+ with only these improvements is already 3.2x faster on YAGO2 (const) and 6.5x faster on YAGO2 than AMIE. This is not surprising since for YAGO2 most of the time is spent on refining rules (Table 3). Therefore, the improvements in this phase result in a significant gain.

Notice also that AMIE+ (only output) is only marginally faster than AMIE for the YAGO2 family of datasets. This is because the confidence approximation heuristic requires computing the join cardinalities for every pair of relations in the KB. This means that there is a trade-off between an initial additional cost for pre-computing these values and the potential savings. For the case of YAGO2, the output phase takes only around 9% of the overall mining time, i.e., the confidence evaluation is not really a problem.

For YAGO2s, DBpedia 2.0, DBpedia 3.8, and Wikidata, we see that using only the refinement improvements or only the output refinements is not enough. If we activate all improvements, however, AMIE+ is able

KB	AMIE	AMIE+					
		Only Refinement	Only Output	Output +			Full
				MRL	QRW	PR	
YAGO2	3.17min	29.37s	2.82min	29.03s	38.16s	2.80min	28.19s
YAGO2 (const)	37.57min	11.72min	37.05min	8.90min	12.04min	36.48min	9.93min
YAGO2 (4)	27.14min	9.49min	26.48min	8.65min	15.69min	24.20min	8.35min
YAGO2s	> 1 day	> 1 day	> 1 day	1h 7min	1h 12min	> 1 day	59.38min
DBpedia 2.0	> 1 day	> 1 day	> 1 day	45.11min	46.37min	> 1 day	46.88min
DBpedia 3.8	> 1 day	> 1 day	11h 46min	8h 35min	7h 33min	10h 11min	7h 6min
Wikidata	> 1 day	> 1 day	> 1 day	1h 14min	7h 56min	> 1 day	25.50min

Table 4: Runtime and output comparison between AMIE and AMIE+ on different KBs. On YAGO2 (4), $maxLen = 4$. On YAGO2 (const), the instantiation operator was switched on.

to terminate in the majority of cases within an hour or in the worst case over-night.

Table 4 also shows the benefit of individual refinement improvements over the baseline of AMIE+ (only output). The improvement that offers the highest speedup (up to 6.7x) is the maximum rule length (MRL), closely followed by query rewriting (QRW, up to 5.1x speedup), whereas perfect rules (PR) rank last. This occurs because MRL is much more often applicable than QRW and PR. Besides, perfect rules are relatively rare in KBs. AMIE found, for instance, 1 perfect rule on YAGO2s and 248 (out of 5K) in DBpedia 3.8.

All in all, we find that AMIE+ can run on several datasets on which AMIE was not able to run. Furthermore, on datasets on which both can run, AMIE+ achieves a speed-up of up to 6.7x.

Output Comparison. Table 6 shows a comparison of AMIE and AMIE+ in terms of output (number of rules). For AMIE+ (full), we report the number of rules that were pruned by the confidence approximation. To assess the quality of the confidence approximation, we report in addition the *pruning precision*. The pruning precision is the ratio of rules for which the confidence approximation introduced in Section 6.2.2 overestimates the actual confidence. We calculate this ratio by counting the number of times that the heuristics produce a higher value than the real confidence (among the rules on which the approximation is applicable). For example, a pruning precision of 96% means that in 4% of the cases the system erroneously pruned rules with a confidence higher than 0.1. As in the previous section, we set a threshold of 0.1 PCA Confidence for AMIE. We also interrupted the system if it ran more than one day. In those cases, we report the output until the point of interruption (denoted by a "*" in Table 6).[5]

As we can see, the pruning by approximation does not entail a serious decrease in the quality of the output:

[5] In these cases, the pruning precision in Table 6 was computed by comparing the output of AMIE+ to the output of AMIE on the mined subset.

$y{:}isCitizenOf(x, y) \Rightarrow y{:}livesIn(x, y)$
$y{:}wasBornIn(x, y) \wedge y{:}isLocatedIn(y, z) \Rightarrow y{:}citizenOf(x, z)$
$y{:}hasWonPrize(x, G.\ W.\ Leibniz) \Rightarrow y{:}livesIn(x, Germany)$
$y{:}hasWonPrize(x, Grammy) \Rightarrow y{:}musicalRole(x, Guitar)$
$d{:}countySeat(x, y) \Rightarrow d{:}largestCity(x, y)$
$d{:}jurisdiction(z, y) \wedge d{:}successor(x, z) \Rightarrow d{:}jurisdiction(x, y)$
$w{:}ownedBy(x, y) \Rightarrow w{:}subsidiary(y, x)$
$w{:}relative(y, z) \wedge w{:}sister(z, x) \Rightarrow w{:}relative(x, y)$

Table 5: Some Rules mined by AMIE on different datasets (y: YAGO, w: Wikidata, d: DBpedia).

KB	AMIE	AMIE+(full)		
	Rules	Rules	Pruned	Prun. prec.
YAGO2	68	68	24	100.00%
YAGO2 (c)	15634	15634	24	100.00%
YAGO2 (4)	645	645	203	100.00%
YAGO2s	94*	94	78	100.00%
DBpedia 2.0	24308*	112865	5380	98.26%
DBpedia 3.8	2470*	5087	2621	98.41%
Wikidata	889*	1512	773	95.35%

Table 6: Output comparison of AMIE (PCA conf ≥ 0.1) and AMIE+ full. Starred: output after processing for 1 day. On YAGO2 (4), $maxLen = 4$. On YAGO2 (const), the instantiation operator was switched on.

AMIE+ does not miss more than 5% of the rules with confidence above 10%. At the same time, the pruning yields a speed-up by a factor of up to 3, as Table 4 shows. Table 5 shows some examples of rules with high confidence that we mined.

Longer Rules. To investigate the performance of AMIE and AMIE+ with longer rules, we ran both systems also with $maxLen = 4$. As Table 4 shows, this affects the runtime: AMIE on YAGO2 with $maxLen = 4$ is 9x slower than with $maxLen = 3$. This is because the number of rules is much larger now: As Table 6 shows, the number of output rules increases by one order of magnitude from 68 to 645. Irrespective of the rule length, the confidence approximation of AMIE+ works correctly, with a 100% pruning precision. At the

$exports(y,z) \wedge imports(y,z) \wedge livesIn(x,y) \Rightarrow citizenOf(x,y)$
$diedIn(x,z) \wedge locatedIn(z,y) \wedge livesIn(x,z) \Rightarrow politician(x,y)$
$advisor(z,w) \wedge citizenOf(w,y) \wedge livesIn(z,x) \Rightarrow deals(x,y)$

Table 7: Examples of rules mined by AMIE on YAGO2 with $n = 4$ atoms

Category	Relation	% of hits
Functions	*wasBornIn*	96.67
	diedIn	96.42
	hasCapital	93.33
Quasi-Functions	*hasCurrency*	75
	hasOfficialLanguage	73.33
	graduatedFrom	64.29
	isCitizenOf	96.42
	$directed^{-1}$	90
	hasAcademicAdvisor	88.89
	$created^{-1}$	86.67
	isLeaderOf	89.47
	isPoliticianOf	100
	isAffiliatedTo	89.47
Granularity Differences	*isLocatedIn*	50
	livesIn	20.83
Implicit Assumptions	*livesIn*	20.83
Source Incompleteness	$influences^{-1}$	34.78
	imports	0
	exports	0
	$actedIn^{-1}$	0
	worksAt	89.66
	hasMusicalRole	22.22
	dealsWith	10
Extraction Incompleteness	$participatedIn^{-1}$	48.14
	isMarriedTo	79.31
	$produced^{-1}$	56.67
	$actedIn^{-1}$	0
	playsFor	20
	holdsPoliticalPosition	26.67
	$hasChild^{-1}$	26.67
	hasWonPrize	31.03
	dealsWith	10
	$influences^{-1}$	34.78
	hasMusicalRole	22.22

Table 8: Categories of relations w.r.t. the validity of the PCA.

same time, the approximation reduces the runtime drastically, so that AMIE+ runs 3 times faster than AMIE on rules with 4 atoms. Table 7 shows some rules with 4 atoms mined on YAGO2. Such rules motivate us to keep the default rule length at 3 atoms.

7.3 AMIE(+) vs. State-of-the-Art Systems

In this section we compare AMIE and AMIE+ to two state-of-the-art rule mining systems that are publicly available: WARMR [17] and ALEPH [31]. We compare the systems in terms of runtime and quality of produced rules. A more detailed description of these experiments (for AMIE), as well as a comparison of usability, can be found in [16]. For these experiments, we did not use any confidence threshold ($minConf = 0$), and hence AMIE+ only used refinement improvements.

Constants	WARMR	AMIE	AMIE+
no	18h	6.02s	2.59s
yes	(48h)	1.43min	1.45min

Table 9: Runtimes on YAGO2 sample

7.3.1 AMIE(+) vs. WARMR

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. It generates Datalog queries of the form "$? - A_1, A_2, \ldots, A_n$", where A_i are logical atoms. WARMR applies a closed world assumption for assessing the quality of the produced rules.

Runtime. We first compare WARMR with AMIE and AMIE+ in terms of runtime only. For a fair comparison, we have to make sure that both systems run in the same settings. Hence, we tweaked AMIE(+) to simulate WARMR's notion of support. We run all systems with an absolute support threshold of 5 entities. We also use the standard confidence as quality metric for rules, instead of the PCA confidence.

In our initial experiment, WARMR was not able to terminate on YAGO2 in a time period of 1 day. Therefore, we created a sample of YAGO2 containing 47K triples (see [16] for details about the sampling method). Table 9 summarizes the runtime results for WARMR, AMIE, and AMIE+ on this dataset. We see that AMIE mines her rules in 6.02 seconds, and AMIE+ even in 3 seconds. WARMR, in contrast, took 18 hours.

We also ran both systems in a mode that allows them to mine rules with constants. For AMIE(+), this means enabling the instantiation operator $\mathcal{O}_I$ (see Section 5.1). AMIE and AMIE+ completed the task in less than 2 minutes. WARMR, in contrast, did not terminate in 3 days. Therefore, we ran it only for the relations *diedIn*, *livesIn*, *wasBornIn*, for which it took 48h. To understand this drastic difference, one has to take into account that WARMR is an ILP algorithm written in a logic programming environment, which makes the evaluation of all candidate queries inefficient.

Results. After filtering out non-connected rules, WARMR mined 41 closed rules. AMIE and AMIE+, in contrast, mined 75 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 $B_1, \ldots, B_n$, 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(+) does not make a closed world assumption as WARMR does. In Section 7.4 we show that the PCA confidence defined by AMIE(+) is more suitable than the standard confidence to identify predictive rules in a web-extracted KB designed under an open world assumption.

7.3.2 AMIE(+) vs. ALEPH

ALEPH is an ILP system that implements a variety of scoring functions for measuring a rule's quality. For our experiments we used the *Positives-only* evaluation function [28, 31], which is the most interesting for our setting, since it does not require the existence of explicit negative examples. It takes random facts as negative evidence, instead:

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

Here, P is the number of known true facts covered (KB-true, or A resp., in Figure 1), R is the number of random examples covered, $Rsize$ is the total number of randoms, and L is the number of atoms in the rule. The intuition behind the formula 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.

Runtime. We ran AMIE, AMIE+ and ALEPH on YAGO2. 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 and AMIE+ to run with a support threshold of 2 facts.

Table 10 shows the results. AMIE terminated in 4.41 minutes and found rules for all relations. AMIE+ was slightly faster. ALEPH runs 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 (Table 11). 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 12). Some relations ran in a few seconds, others did not terminate in a day. AMIE, in contrast, found her rules in 6 seconds, and AMIE+ in half that time.

KB	ALEPH	AMIE	AMIE+
YAGO2 full	4.96s to > 1 day	4.41min	3.76min
YAGO2 Sample	0.05s to > 1 day	5.65s	2.90s

Table 10: Runtimes ALEPH vs. AMIE 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 11: 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 12: Runtimes of ALEPH on YAGO2 sample

Results. We compared the output of ALEPH with the positives-only evaluation function to the output of AMIE(+) using the PCA confidence on the sample of YAGO2 used for the runtime experiments. Since ALEPH required more than one day for some relations, we used only rules for which the head relation runs in less than one day. ALEPH mined 56 rules, while AMIE(+) mined 302 rules. We ordered the rules by decreasing score (ALEPH) and decreasing PCA confidence (AMIE(+)). We computed the precision of the rules by evaluating whether a prediction made by the rule is correct or not (more on that metric in Section 7.5). Table 13 shows the number of predictions and their total precision. 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).

	Top n	Predictions	Precision
Positives-only	7	2997	27%
PCA Confidence	12	2629	62%
Positives-only	9	5031	26%
PCA Confidence	22	4512	46%
Positives-only	17	8457	30%
PCA Confidence	23	13927	43%

Table 13: PCA confidence vs. positives-only score: aggregated precision of rules mined on YAGO2 sample.

We suspect that ALEPH's positives-only evaluation function manages to filter out overly general rules only to some extent. The reason is that this measure "guesses" negative examples at random, whereas rules usually create false predictions in a non-random way. Even if a rule produces many false predictions, the intersection of these false predictions and the random counterexamples may be very small. Consider for example the rule $bornIn(x, y) \Rightarrow diedIn(x, y)$, which produces false predictions for example for persons who have moved to a different place during their life. By creating negative examples just by considering random person-location pairs, we might not produce any case for which the rule will give a false prediction, simply because such a negative example will have a relatively small probability to be generated.

Summary. Our experimental results show that AMIE (and in particular AMIE+) can be up to 3 orders of magnitude faster than other state-of-the-art systems, namely WARMR [17] and ALEPH [31]. The PCA confidence was shown to rank productive and correct rules higher than other confidence metrics.

7.4 Evaluation of the PCA

The Partial Completeness Assumption (PCA) says that if, for a given subject s and a given relation r, the KB knows one object o with $r(s, o)$, then the KB knows *all* objects o' with $r(s, o')$ (Sec. 4). The original AMIE paper used the PCA but it did not evaluate whether this assumption is true or not [16]. Since the PCA is one of the basic ingredients of AMIE(+)'s mining model, we wanted to know to what extent this assumption holds in a real-world KB.

Setup. We looked into each of the 31 relations between entities in YAGO2. For each relation r, we randomly sampled 30 subjects. For each subject x, we checked whether the KB knows all y with $r(x, y)$. If the relation is more inverse functional than functional ($ifun(r) > fun(r)$, see Section 3.2), we considered r^{-1} instead.

As a ground truth, we took the Wikipedia page of x and what we could find on the Web by a search engine. It is obvious that such an evaluation cannot be done strictly quantitatively. For example, a person might have worked for a company, but this fact might appear nowhere on Wikipedia – or even on the Web. Or a musician might play 10 instruments at different levels of proficiency, but Wikipedia mentions only the 4 main instruments. Even a search on the Web might not tell us that there are more than 4 instruments. Therefore, we resorted to a qualitative analysis. We analyzed each of the relations manually, and grouped the relations into categories. Some relations fall into multiple categories. Table 8 shows, for each relation, the percentage of subjects in our sample for which the PCA holds.

Functions and Quasi-Functions. By definition, the PCA holds for functions. Our manual analysis, however, did not result in 100% precision for functional relations in Table 8. This is because our analysis also counts the cases where the KB contains bugs. If, for instance, YAGO knows the wrong place of death of a person, then there exists another value outside YAGO that is the right value. However the PCA would reject it. Hence, we count this case as a miss.

The PCA extends well to relations that are strictly speaking not functions, but that have a high functionality. These are relations that usually have one object per subject, even though there could be several objects. For example, a person can graduate from several universities, but most people graduate from a single university. We call these relations *quasi-functions*. The PCA worked very well also on these, and predicted completeness correctly for 73% – 100% of the subjects under investigation. Since the PCA takes into account the direction of the functionality, the PCA also holds for quasi inverse-functional relations such as *directed*.

Granularity Differences. Some relations, such as *locatedIn* and *livesIn*, hold between an entity and a geographical region. In that case, the region can be given at the granularity of a city, a region, a country, or a continent. Naturally, if YAGO contains one of these, the others are possible options. Hence, PCA fails and we found rather low precision values. However, these cases could be addressed if one restricts the range of the relation (say, to cities). With such a restriction, the relations become functions or quasi-functions, which lifts them into the category where the PCA works well. As we will see in Section 7.5, the use of types can significantly improve the performance of AMIE.

Implicit Assumptions. Some statements can be inferred from the Wikipedia page even if the page does not mention them. People usually do not state information that can easily be inferred by what they have stated before (following Grice's Maxim of quantity and manner [18]). For example, if someone graduated from a

university, people usually do not feel obliged to mention that this person used to live in the country in which the university is located, because this can easily be inferred by the reader. Only less obvious residences will be explicitly mentioned. Therefore, the PCA does not always hold. Note that rules such as $graduatedFrom(x, y) \Rightarrow livesIn(x, y)$ can only be mined if Grice's maxims are occasionally violated by the authors of the articles. If the authors always follow the maxims, then such rules cannot be mined, because there are not even positive examples for which the rule holds (lack of support). In the case of YAGO, the only relation that we found in this category is *livesIn*.

Source Incompleteness. For many relations, the source itself (Wikipedia) is incomplete. Usually, these relations have, for each subject, some objects that are undisputed. For example, it is undisputed that Albert Einstein is interested in physics. However, these relations also have objects that are less important, disputed, or unknown. For example, Albert Einstein might also be interested in music (he played the violin), but maybe also in pancakes. These less prominent objects are a lot less likely to appear in Wikipedia, or indeed on any Web page. Even if they do, we can never be sure whether there is not still something else that Einstein was interested in. For these relations, the knowledge sources are often incomplete by nature. For example, not every single product that a country imports and exports is explicitly mentioned. Whether or not this poses a problem depends on the application. If ground truth is defined as what is universally true, then source incompleteness is a problem. If ground truth is the source of the KB (i.e., Wikipedia in this case), then source incompleteness is not an issue.

Extraction Incompleteness. For a large number of relations, the Wikipedia page contains more objects for a given subject than the KB. These are cases where the extraction process was incomplete. In the case of YAGO, this is due to a strong focus on accuracy, which causes the extraction to discard any extracted fact that cannot be type checked or linked to an entity. This class of relations is the most sensitive category for the PCA. The success of the PCA will depend on how many relations and to what extent they are affected by incomplete extractions.

Discussion. In summary, our analysis shows that it depends on the nature of the relation and on its type signature whether the PCA holds or not. There is a large number of relations for which the PCA is reasonable. These are not just functions and inverse functions, but also relations that exhibit a similar behavior.

For many other cases, the PCA does not hold. In these cases, AMIE(+) will falsely assume that a rule is making incorrect predictions – although, in reality, the predictions might be correct. Thus, when the PCA does not hold, AMIE(+) will err on the side of caution.

At the same time, the PCA is not as restrictive as the closed world assumption (CWA): the PCA admits that there can be facts that are true, but not known to the KB. For example, if a person has a birth date, then both the CWA and PCA would not admit another birth date. However, if a person does not have a birth date, then the PCA will admit that there can be a birth date, while the CWA will assume that there cannot be a birth date. Thus, the PCA is more permissive than the CWA. This encourages us to use the PCA for the definition of our confidence. In the following, we will show that this definition of confidence produces more predictive and more accurate rules than the standard confidence, which is based on the CWA.

7.5 Predicting Facts

Prediction. One of the applications of the mined rules could be to predict new facts. Based on what the KB knows, one aims to predict what else might be the case in the real world. This is a difficult endeavor: It amounts to guessing the places of residence for people, their birth place, or even their death place. Naturally, we may not assume a high precision in the prediction of the future. We may only expect educated guesses.

To evaluate the precision of these guesses, we proceeded as follows: We ran our system with the default setting on the YAGO2 dataset. For each rule, we evaluated whether the predictions that go beyond YAGO2 were true. We did this by either checking whether the prediction appears in a newer version of the KB (YAGO2s), or by manually checking them in Wikipedia. If we could find the predicted fact in neither, we evaluated it as false.

Standard vs. PCA Confidence. Our first goal is to see whether the PCA confidence or the standard confidence perform better in this task. Since both AMIE and AMIE+ can work with both confidence metrics, and since their output is the same, we report here our results from [16] with AMIE. We ran AMIE, and sorted the resulting rules first by descending PCA confidence, and then by descending standard confidence. We looked at the top ranked rules in each case, and evaluated the precision of the predictions. The bottom curves of Figure 2 plot the aggregated predictions versus the aggregated precision for the standard and the PCA confidence. The n-th dot from the left represents the total number of unique predictions and the total precision of these predictions, aggregated over the first n rules. As we see, ranking the rules by standard confidence is

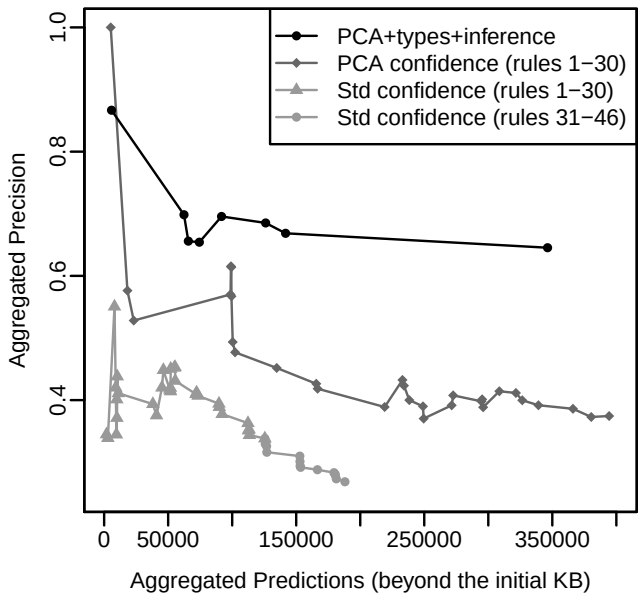


Fig. 2: Std. confidence vs. PCA confidence

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 34%. In contrast, if we rank the rules by PCA confidence, we quickly get large numbers of predictions. The top 10 rules already produce 135K predictions – at a precision of 45%. 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. We thus conclude that the PCA confidence is better suited for making predictions than the standard confidence. We show in [16] that the PCA confidence also correlates better with the actual precision of a rule.

Using Type Information. The previous experiment showed us the precision of individual rules for prediction. To make more accurate predictions, we have to combine these rules with more signals. We proceed as follows. In Section 7.4 we discussed the granularity differences in relations. For instance, the relation *livesIn* is used to express a person's city or country of residence. This implies that, for example, the rule $livesIn(x, y) \Rightarrow isCitizenOf(x, y)$ can predict that some people are citizens of cities. Such spurious predictions decrease the precision of the inference process. Therefore, we configured AMIE+ to mine *typed rules*. These have the form:

$$\overrightarrow{B} \wedge \textit{rdf:type}(x, D) \wedge \textit{rdf:type}(y, R) \Rightarrow r(x, y)$$

where D and R correspond to the domain and range of the head relation r in YAGO3.[6] To allow AMIE+ to find such rules, we augmented the YAGO2 dataset by adding the *rdf:type* statements about the subjects and objects of the triples.

Joint Prediction. Our second observation is that the same prediction can be fired from multiple rules. If we consider rules as signals of evidence, then facts predicted by more rules should get a higher confidence score. In YAGO2, 9% of the predictions are fired by more than one rule (with a PCA confidence threshold of 0.1). To take this into account, we changed the way predictions are ranked. In the original experimental setup, if multiple rules $R_1, \ldots R_k$ made a prediction p, the prediction was only counted the first time it was fired. Since the rules were ranked by decreasing PCA confidence, this was equivalent to ranking the predictions according to their highest PCA confidence:

$$score(p) := max\{conf_{pca}(R_1), \ldots, conf_{pca}(R_k)\}$$

We propose an alternative score instead:

$$score^*(p) := 1 - \prod_{i=1}^{k} (1 - conf_{pca}(R_i)) \quad (8)$$

Equation 8 aggregates the PCA confidence of the rules so that the predictions concluded by multiple rules are ranked higher. It also confers a probabilistic interpretation to the PCA confidence. The score of a prediction is the probability that at least one of the rules in $R_1, \ldots R_k$ concludes p. This is computed as 1 minus the probability that none of the rules concludes p. The probability of a rule not concluding p is defined as 1 minus the PCA confidence of the rule. The probability that none of the rules concludes p is the product of the individual probabilities. Although this scoring-scheme is very simplistic (it assumes independence of the rules, and confers a probabilistic interpretation to the confidence), it can still serve as a proof of concept. In real applications, more involved methods [32,36] can be used for joint prediction.

Results. The upper curve in Figure 2 shows the precision of the predictions made with both heuristics. We proceeded as in the previous experiment, that is, we first used the rules to fire predictions, and then we ranked these predictions by descending score and computed their cumulative precision. Unlike in the original experimental setup, the n-th point from the left in the new curve corresponds to the cumulative precision of

[6] We used the YAGO3 [27] types because the type signatures in older versions of YAGO were too general. E.g., the relation *livesIn* is defined from person to location in YAGO2s, whereas in YAGO3 it is defined from person to city.

the predictions up to the n-th bucket. We bucketized the predictions by score using a bucket size of 0.1, i.e., the first point corresponds to the predictions with score between 1 and 0.9, the next one accounts for the predictions with score between 0.9 and 0.8 and so on.

As we can observe, our heuristics have a significant effect on the precision of the predictions. The precision is much higher at each level of recall, compared to the original experiment. We can make 100,000 predictions at a precision of 70%. At 400K predictions, we still achieve a precision of 60%. While these predictions should not be added directly to a KB, they could be sent to human evaluators to check their correctness. It is much easier for a person to check fact candidates for their correctness than to invent them from scratch. In addition, this experimental setup can serve as a baseline for more sophisticated inference approaches.

8 Conclusion

In this paper, we have presented AMIE, an approach to mine Horn rules on large RDF knowledge bases. AMIE is based on a formal model for rule mining under the Open World Assumption, a method to simulate counter-examples, and a scalable mining algorithm. In contrast to state-of-the-art approaches, AMIE requires no input other than the KB and does not need configurations or parameter tuning.

We have extended AMIE to AMIE+ by a series of pruning and query rewriting techniques, both lossless and approximate. 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. If we combine these rules with simple heuristics for type checking and joint prediction, we can use them to predict facts with a precision of about 70%.

For future work, we aim to develop better joint inference approaches based on the rules mined by AMIE. We also aim to extend the set of rules beyond the language of closed Horn rules, so that even more facts can be predicted.

References

1. Abedjan, Z., Lorey, J., Naumann, F.: Reconciling ontologies and the web of data. In: CIKM (2012)
2. Adé, H., Raedt, L., Bruynooghe, M.: Declarative bias for specific-to-general ilp systems. Machine Learning **20** (1995)
3. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: SIGMOD (1993)
4. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In: Advances in knowledge discovery and data mining (1996)
5. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: DBpedia: A nucleus for a Web of open data. In: ISWC (2007)
6. Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Jr., E.R.H., Mitchell, T.M.: Toward an architecture for never-ending language learning. In: AAAI (2010)
7. Chasseur, C., Patel, J.M.: Design and evaluation of storage organizations for read-optimized main memory databases. Proc. VLDB Endow. **6**(13) (2013)
8. Chi, Y., Muntz, R.R., Nijssen, S., Kok, J.N.: Frequent Subtree Mining - An Overview. Fundam. Inf. **66**(1-2) (2004)
9. Cimiano, P., Hotho, A., Staab, S.: Comparing Conceptual, Divisive and Agglomerative Clustering for Learning Taxonomies from Text. In: ECAI (2004)
10. d'Amato, C., Bryl, V., Serafini, L.: Data-driven logical reasoning. In: URSW (2012)
11. d'Amato, C., Fanizzi, N., Esposito, F.: Inductive learning for the Semantic Web: What does it buy? Semant. web **1**(1,2) (2010)
12. David, J., Guillet, F., Briand, H.: Association Rule Ontology Matching Approach. Int. J. Semantic Web Inf. Syst. **3**(2) (2007)
13. Dehaspe, L., Toironen, H.: Discovery of relational association rules. In: Relational Data Mining. Springer-Verlag New York, Inc. (2000)
14. Dehaspe, L., Toivonen, H.: Discovery of frequent DATALOG patterns. Data Min. Knowl. Discov. **3**(1) (1999)
15. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In: KDD (2014)
16. Galárraga, L.A., Teflioudi, C., Hose, K., Suchanek, F.M.: AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases. In: WWW (2013)
17. Goethals, B., Van den Bussche, J.: Relational Association Rules: Getting WARMER. In: Pattern Detection and Discovery, vol. 2447. Springer Berlin / Heidelberg (2002)
18. Grice, P.: Logic and conversation. J. Syntax and semantics **3** (1975)
19. Grimnes, G.A., Edwards, P., Preece, A.D.: Learning Meta-descriptions of the FOAF Network. In: ISWC (2004)
20. Hellmann, S., Lehmann, J., Auer, S.: Learning of OWL Class Descriptions on Very Large Knowledge Bases. Int. J. Semantic Web Inf. Syst. **5**(2) (2009)
21. Huang, Y., Tresp, V., Bundschus, M., Rettinger, A., Kriegel, H.P.: Multivariate prediction for learning on the semantic web. In: ILP (2011)
22. Jozefowska, J., Lawrynowicz, A., Lukaszewski, T.: The role of semantics in mining frequent patterns from knowledge bases in description logics with rules. Theory Pract. Log. Program. **10**(3) (2010)
23. Kuramochi, M., Karypis, G.: Frequent Subgraph Discovery. In: ICDM. IEEE Computer Society (2001)
24. Lehmann, J.: DL-Learner: Learning Concepts in Description Logics. Journal of Machine Learning Research (JMLR) **10** (2009)
25. Lisi, F.A.: Building rules on top of ontologies for the semantic web with inductive logic programming. TPLP **8**(3) (2008)
26. Maedche, A., Zacharias, V.: Clustering Ontology-Based Metadata in the Semantic Web. In: PKDD (2002)

27. Mahdisoltani, F., Biega, J., Suchanek, F.M.: Yago3: A knowledge base from multilingual wikipedias. In: CIDR (2015)
28. Mamer, T., Bryant, C., McCall, J.: L-modified ilp evaluation functions for positive-only biological grammar learning. In: F. Zelezny, N. Lavrac (eds.) Inductive logic programming, no. 5194 in LNAI. Springer-Verlag (2008)
29. McGuinness, D.L., Fikes, R., Rice, J., Wilder, S.: An Environment for Merging and Testing Large Ontologies. In: KR (2000)
30. Muggleton, S.: Inverse entailment and progol. New Generation Comput. **13**(3&4) (1995)
31. Muggleton, S.: Learning from positive data. In: ILP (1997)
32. Nakashole, N., Sozio, M., Suchanek, F., Theobald, M.: Query-time reasoning in uncertain rdf knowledge bases with soft and hard rules. In: Workshop on Very Large Data Search (VLDS) at VLDB (2012)
33. Nebot, V., Berlanga, R.: Finding association rules in semantic web data. Knowl.-Based Syst. **25**(1) (2012)
34. Nickel, M., Tresp, V., Kriegel, H.P.: Factorizing yago: scalable machine learning for linked data. In: WWW (2012)
35. Noy, N.F., Musen, M.A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: AAAI/IAAI. AAAI Press (2000)
36. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62**(1-2) (2006)
37. Schoenmackers, S., Etzioni, O., Weld, D.S., Davis, J.: Learning first-order Horn clauses from web text. In: EMNLP (2010)
38. Suchanek, F.M., Abiteboul, S., Senellart, P.: PARIS: Probabilistic Alignment of Relations, Instances, and Schema. PVLDB **5**(3) (2011)
39. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: WWW (2007)
40. Tan, P.N., Kumar, V., Srivastava, J.: Selecting the right interestingness measure for association patterns. In: KDD (2002)
41. Technologies, M.: The freebase project. http://freebase.com
42. Völker, J., Niepert, M.: Statistical schema induction. In: ESWC (2011)
43. Word Wide Web Consortium: RDF Primer (W3C Recommendation 2004-02-10). http://www.w3.org/TR/rdf-primer/ (2004)
44. Zeng, Q., Patel, J., Page, D.: QuickFOIL: Scalable Inductive Logic Programming. In: VLDB (2014)
45. Ziawasch Abedjan, F.N.: Synonym analysis for predicate expansion. In: ESWC (2013)