Relatedness and TBox-Driven Rule Learning in Large Knowledge Bases

Giuseppe Pirrò[1]
[1]Department of Computer Science, Sapienza University of Rome
Via Salaria 113, 00198, Rome, Italy
pirro@di.uniroma1.it

Abstract

We present RARL, an approach to discover rules of the form *body* $\Rightarrow$ *head* in large knowledge bases (KBs) that typically include a set of terminological facts (TBox) and a set of TBox-compliant assertional facts (ABox). RARL's main intuition is to learn rules by leveraging TBox-information and the semantic relatedness between the predicate(s) in the atoms of the *body* and the predicate in the *head*. RARL uses an efficient relatedness-driven TBox traversal algorithm, which given an input rule *head*, generates the set of most semantically related candidate rule bodies. Then, rule confidence is computed in the ABox based on a set of positive and negative examples. Decoupling candidate generation and rule quality assessment offers greater flexibility than previous work.

1 Introduction

Knowledge Bases (KBs) like Yago, Wikidata, and DBpedia contain millions of facts (ABox) structured with the help of an underlying schema (TBox), which in its essential form allows to define entity types and their hierarchy along with the domain and range of properties. Fig. 1 (a) shows part of the DBpedia T-Box, where the class Artist is a subClassOf Person and the predicate director has as a domain Film and as a range Person. Fig. 1 (c) shows an excerpt of A-Box conforming to the TBox. KBs are automatically populated thus being inherently incomplete (Darari et al. 2018) and prone to errors. As an example, the fact (Virginia Kelley, spouse, Roger Clinton) is missing in DBpedia. One way to overcome these issues is to define declarative logical rules like:

$$\mathsf{child}(\mathsf{x},\mathsf{y}) \wedge \mathsf{parent}(\mathsf{y},\mathsf{z}) \Rightarrow \mathsf{spouse}(\mathsf{x},\mathsf{z}) \qquad (1)$$

where spouse(X, Z) is the rule head (a single atom) and child(X, Y)$\wedge$parent(Y, Z) is the rule body (a conjunction of atoms). The rule states that some X is the spouse of some Z if some child Y of X has as a parent Z and allows to infer, among the others, the above mentioned missing fact. Rules capture common correlations in the data and are useful in a variety of tasks including KB completion (Chen et al. 2016), ontology enrichment (d'Amato et al. 2016), fact -checking (Shi and Weninger 2016; Fionda and Pirrò 2018), and storage optimization (Gayathri and Kumar 2015). Rules can be used to construct relatedness explanations (Pirrò 2019) that allow to (visually) understand data correlations; as an example, one may infer that a profession typically involves a specialization in a particular field.

We present RARL (Relatedness-Aware Rule Learning), a rule learning approach, which combines a relatedness-driven TBox traversal algorithm for candidate rule generation (§ 3.1) with an algorithm for candidate verification and rule confidence assessment in the ABox (§ 3.2, § 3.3). Given a rule head pred(D, R), candidate rule bodies (§ 2.1) are generated as TBox paths that start from one of the domains D of pred and end into one of its ranges R. An example of candidate rule body for spouse(Person, Person) is the TBox path $\mathsf{Person} \xrightarrow{\text{child}} \mathsf{Person} \xrightarrow{\text{parent}} \mathsf{Person}$ (Fig. 1 (b)). To prune the candidate search space, RARL considers bounded-length bodies only including the top-k semantically related predicates to pred (§ 2.2). However, not all candidate bodies are verified, that is, have bindings in the ABox and not all of them offer the same confidence. RARL leverages an algorithm based on positive/negative examples for candidate verification and confidence assessment (§ 2.3). We show (§ 4) that RARL is scalable and finds good rules, especially on schema-rich KBs like DBpedia, Yago, and Freebase.

RARL has been inspired by Ontological Path Finding (OPF) (Chen et al. 2016) for candidate generation. However, OPF generates an enormous amount of candidates, especially in schema-rich KBs like DBpedia. As an example, on the TBox in Fig. 1 (b) and for the input predicate spouse, OPF would not be able to distinguish between the candidates $\mathsf{child}(\mathsf{x},\mathsf{y}) \wedge \mathsf{parent}(\mathsf{y},\mathsf{z})$ and $\mathsf{lyrics}(\mathsf{x},\mathsf{y}) \wedge \mathsf{deathPlace}(\mathsf{y},\mathsf{z})$, while our approach, which is driven by a relatedness criterion, will discard the second candidate since lyrics and deathPlace have a much lower degree of relatedness to the predicate spouse than child or parent. We were inspired by RuDiK (Ortona, Meduri, and Papotti 2018) for the usage of examples. However, while RuDiK focuses on learning the smallest number of rules satisfying the ex-

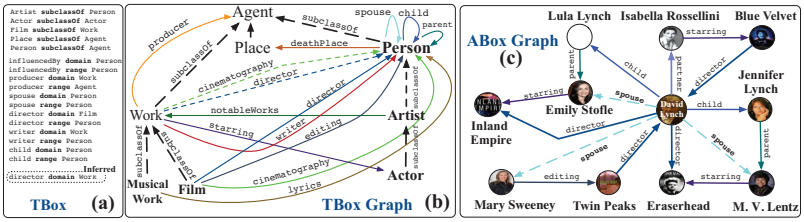

Figure 1: An example of T-Box (a), its T-Box graph (b) and an A-Box graph (c) conforming to the T-Box.

amples, RARL learns every possible rule prioritizing those having high head-body relatedness. We refer the reader to Section 5 for a more comprehensive discussion about related work.

2 Definitions and Background

A Knowledge Base (KB) contains facts that can be divided into an ABox and a TBox. We see the ABox as a node and edge-labeled directed multi-graph $G=(V,E,T)$ where V is the set of uniquely identified vertices representing entities (e.g., D. Lynch), E the set of predicates (e.g., director) and T a set of facts (s, p, o), where s, o $\in V$ and p $\in E$. We denote by ˆp the *inverse* of a predicate (from the object o to the subject s). We represent a fact (s, p, o) as p(s, o) and (s, ˆp, o) as p(o, s). The TBox is a set of facts T_f expressed via a reserved vocabulary and is used to structure knowledge in the ABox. We focus on RDFS TBoxes and in particular on the subset of the RDFS vocabulary E_ρ={rdf:type, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, rdfs:range}. An excerpt of facts about the DBpedia TBox is shown in Fig. 1 (a). We considered RDFS instead of more expressive languages like OWL for two main reasons. First, information about class hierarchies along with information about the domain/range of predicates is widely available in many KBs including DBpedia, Yago, Freebase, and Wikidata. Second, new facts about the TBox can be efficiently derived by applying a subset of the RDFS inference rules (Munoz, Pérez, and Gutierrez 2009; Franconi et al. 2013). We apply RDFS inference rules to the TBox to deduct novel subclass/subproperty and domain and range information. For instance, by applying the RDFS inference rules on the facts of the TBox in Fig. 1 (b), a new domain for the predicate director is Work since Film is a subClassOf Work and Film has as domain director.

Given a set of TBox facts T_f, its TBox graph is defined as $G_S=(V_s,E_s,T_s)$, where each $v_i \in V_s$ denotes a class (i.e., entity type) and E_s includes all predicates defined in T_f. Moreover, $(v_s,p_i,v_t) \in T_s$ is a triple, where v_i (resp., v_t) is the domain (resp., range) of the predicate $p_i \in E_s \cap E$. An excerpt of TBox graph is shown in Fig. 1 (b). A rule consists of a *head* (a single atom) and a *body* (conjunction of atoms). A rule having pred(X, Y) as a *head* and $B_1 \wedge B_2 \wedge ...B_d$ as a *body* is represented as $B_1 \wedge B_2 \wedge ...B_d \Rightarrow$ pred(X, Y) where X and Y are variables. RARL, like many other approaches (e.g., (Galárraga et al. 2015; Omran, Wang, and Wang 2018)) does not learn arbitrary Horn rules but introduces a language bias towards what rules can be learned. In particular, it focuses on closed path rules of bounded length, that is, rules having at most d atoms in the body and where the sequence of predicates in the body forms a path from the domain to the range of the predicate in the input head.

2.1 Candidate Rule Bodies and Traversal Queries

Given a head pred(D, R) and an integer d, a candidate rule body is a set of d atoms $B_1 \wedge B_2 \wedge ...B_d$ that form a closed path. We use D and R for the domain (resp., range) of pred. A candidate rule body is a TBox path that starts from some D∈domain(pred) and ends into some R∈range(pred). Consider the predicate spouse and d=2, we have that domain(spouse)=range(spouse)={Person}. Hence, child(Person$_X$, Person$_Y$)∧parent(Person$_Y$, Person$_Z$) is a candidate body, where Person$_X$, Person$_Y$, and Person$_Z$ are variables meaning that the bindings of these variables have to be entities of type Person. RARL can relax typing information constraints by substituting types with fresh variables (e.g., child(X, Y) ∧ parent(Y, Z)). To verify candidates in the ABox, we translate them into a subset of the language of traversal queries (Fionda, Pirrò, and Gutierrez 2015; Fionda, Pirrò, and Consens 2015) including concatenation (/), inverse predicate (ˆ), and node tests ([·]).

Definition 1 *(Traversal Query) Given* $\Pi=B_1 \wedge B_2 \wedge ...B_d$, *let* v_s *be the shared variable between* B_i *and* B_{i+1} *(we consider closed-path rules),* v_1^1 *and* v_2^1 *(resp.,* v_1^2 *and* v_2^2*) the first (resp., second) other variable, and* p_i *the predicate in* B_i.

- *If* $v_s=v_1^1=v_1^2$, *then* $B_i \wedge B_{i+1}$ *becomes* $\hat{}p_i/p_{i+1}$.
- *If* $v_s=v_1^1=v_2^2$, *then* $B_i \wedge B_{i+1}$ *becomes* $\hat{}p_i/\hat{}p_{i+1}$.
- *If* $v_s=v_2^1=v_1^2$, *then* $B_i \wedge B_{i+1}$ *becomes* p_i/p_{i+1}.
- *If* $v_s=v_2^1=v_2^2$, *then* $B_i \wedge B_{i+1}$ *becomes* $p_i/\hat{}p_{i+1}$.

By concatenating the chunks from all B_i *and* B_{i+1}, *we get the traversal query associated with the candidate body.*

When considering typed variables (e.g., Person$_X$), each chunk also includes a type check (via the $[\cdot]$ operator) of the endpoints. Given child(Person$_X$, Person$_Y$)∧parent(Person$_Y$, Person$_Z$) the corresponding traversal queries is [Person]child/parent[Person]. When no typing information is considered the query is child/parent.

A candidate is verified if there exists a copy of it in the ABox where all (typed) variables are substituted by constants. For instance, the previous candidate is verified by child(D. Lynch, L. Lynch)∧parent(L. Lynch,

E. Stofle) and child(D. Lynch, J. Lynch)∧parent(J. Lynch, M. V. Lentz) while the candidate parent(Person_Y, Person_X)∧parent(Person_X, Person_Z) cannot be verified (Fig. 1 (c)). Note that traversal queries get rid of variables from candidate bodies. The usage of traversal queries in our algorithm for body verification and confidence assessment guarantees efficiency. Indeed, the rule learning algorithm implemented by RARL algorithm runs in polynomial time. On the contrary, if we were to consider bodies with variables, that is, conjunctive queries, performance will significantly worsen. We will treat this aspect in Section 4.1.

2.2 Predicate and Rule Relatedness

To assess the relatedness between a pair of predicates p_i and p_j, we start by computing the Triple Frequency defined as: $TF(p_i, p_j)$=$\log(1 + C_{i,j})$, where $C_{i,j}$ counts how many times p_i and p_j connect the same subjects and objects in the ABox. The Inverse Triple Frequency is defined as: $ITF(p_j, E)$=$\log \frac{|E|}{|\{p_i : C_{i,j} > 0\}|}$. This approach resembles the TF-IDF scheme used in information retrieval that has been further extended to KGs (Shiralkar et al. 2017; Pirrò 2012). Hence, we can build a (symmetric) matrix C_M where each element is $C_M(i, j)$=$TF(p_i, p_j) \times ITF(p_j, E)$. At this point, a relatedness matrix M_R can be constructed, where $M_R(p_i, p_j)$=$Rel(p_i, p_j)$=$Cosine(W_i, W_j)$, where W_i (resp., W_j) is the row of p_i (resp., p_j) in C_M. The intuition of RARL is that meaningful rules contain (sequences of) predicates in the *body* that are semantically close to the predicate in *head*. Let r: $B_1 \wedge B_2, ..., B_d \Rightarrow \mathsf{p}(\mathsf{X}, \mathsf{Y})$ be a rule where p and $p_i \in B_i$ are predicates. Hence, we define three rule relatedness measures:

1. $rel_H(r) = \frac{Rel(p,p_1)+...+Rel(p,p_d)}{d}$, which is the average relatedness between the predicate in the head and the predicates in the body.
2. $rel_B(r)$=$\sum_{i=2}^{d} \sum_{j=1}^{i-1} Rel(p_i, p_j)$, which considers

relatedness values between predicates in the body, where:

$$M = \begin{bmatrix} Rel(p_1,p_1) & Rel(p_1,p_2) & \cdots & Rel(p_1,p_d) \\ Rel(p_2,p_1) & Rel(p_2,p_2) & \cdots & Rel(p_2,p_d) \\ \cdots & \cdots & \cdots & \cdots \\ Rel(p_d,p_1) & Rel(p_d,p_2) & \cdots & Rel(p_d,p_d) \end{bmatrix}$$

3. $rel_{HB}(r) = \gamma \cdot rel_H(r) + \theta \cdot rel_B(r)$, with $\gamma + \theta$=1

2.3 Rule Confidence

The confidence of a rule is usually defined as the number of body groundings, divided by the number of those body groundings that make the head true. The Partial Completeness Assumption (PCA) (Galárraga et al. 2015) allow to guess counter-examples for rules assuming that if a KB contains one or more object values for a given subject and predicate, then it contains all possible values. Zupanc et al. (Zupanc and Davis 2018) further refine the PCA as for information about triples not in the KBs. Other approaches like (Tanon et al. 2017) consider completeness-aware scores. These approaches can be computationally demanding in large KBs due to the potential many joins to be evaluated that are related to the number of body atoms. In this work, to better cater for data errors and incompleteness that characterize KBs (Ortona, Meduri, and Papotti 2018) RARL computes rule confidence based on a set of (automatically generated) positive (V^+) and negative (V^-) examples (Ortona, Meduri, and Papotti 2018). Positive examples $(p_s, p_e) \in V^+$ can be directly found by sampling from the ABox triples of the form (p_s, p, p_e), where p is the predicate in the rule head. As for the set V^-, a negative pair (r_s, r_e) is generated if *(i)*: either r_s (resp., $\overline{r_s}$) is the subject of one or more facts $(r_s, p, \overline{r_e})$ (resp.,$(\overline{r_s}, p, r_e)$ with $r_e \neq \overline{r_e}$ and $r_s \neq \overline{r_s}$; *(ii)*: (r_s, r_e) belongs to $(r_s, \overline{p}, r_e)$ with $\overline{p} \neq p$; *(iii)* the constrain type(r_s)=domain(p) and type(r_e)= range(p) is respected.

The confidence measure for a candidate rule body Π and a set of positive and negative pairs is :

$$c_{ex}(\Pi, V^+, V^-) = \alpha \cdot \frac{|S^+|}{|V^+|} - \beta \cdot \frac{|S^-|}{|V^-|} \qquad (2)$$

where $|S^+|$ (resp., $|S^-|$) is the number of positive (resp., negative) pairs that verify the candidate body and α+β=1. RARL learns *uncertain rules* covering at least some positive and usually also some negative example. Moreover, by switching the role of positive and negative examples RARL can mine negative rules useful to spot inconsistencies (e.g., if A is married to B, then A cannot be the child of B).

3 Rule Learning Algorithm

We now describe the RARL rule learning algorithm (Algorithm 1). Given an input predicate p, the algorithm finds candidate rule bodies having at most d atoms, including the top-k predicates related to p, and ranked by the measure rel_R (§ 2.2) (line 2). The algorithm builds a reduced (typically smaller) ABox graph by considering the examples V^+ and V^- upon which rule confidence is assessed (line 6) based on the weight of positive (α) and negative (β) examples.

Algorithm 1: RARL (p, k, d, G, G_S, M_R, V^+, V^-, rel_R, α, β)

```
1 R = ∅
2 B= generateCandidateBodies(p, k, d, G_S, M_R, rel_R)
3 while B ≠ ∅ do
4     Π_i = REMOVEFIRST(B) /* priority queue ranked by
        rule relatedness*/
5     G_e(V+, V−) = getReducedABoxGraph(G, Π_i,
        V+, V−)
6     r=getRuleAndConfidence
        (Π_i, α, β, V+, V−, G_e)
7     R = R ∪ r
```

3.1 Candidate Generation

Candidate generation (Algorithm 2) is based on a traversal of the TBox graph G_S, which is treated as an undirected graph. To prune the search space during the traversal, the algorithm adopts two strategies. First, it only considers the

top-k most related predicates to p (found via the relatedness matrix M_R) and the domains and ranges of these predicates as starting and ending nodes, respectively (lines 3-4). To avoid the generation of too abstract rules, for each predicate it considers the domain/range C_a asserted in the TBox and, among those inferred, only that immediately up to C_a in the class hierarchy, if any. For instance, in the TBox in Fig. 1 (b) for director, it considers Film and Work, but do not consider Agent. On the other hand, for producer it considers Agent. Second, the traversal is bound by the value d, that is, the distance (in terms of edges) from the starting node (one of the domains of the target or related predicates) to the end node (one of the ranges of the target or related predicates). The algorithm in parallel (lines 5-19):

1. Generates length-1 candidate bodies (lines 6-11) starting from the domains or ranges of predicates in R. For instance, if writer is the input predicate and director is a related predicate, length-1 candidate bodies must start either from Work or Film (i.e., the domain of writer and director, respectively) and end into Person (the range of both writer and director). In this case, writer(Work, Person) and director(Work, Person) (the edge director from Work was inferred via RDFS reasoning) are both length-1 valid candidates (see Fig.1 (b)) that are added to the results along with their rule relatedness values (line 11).
2. Expands length-1 candidates up to length d and add to the results if a valid range is reached (lines 12-19). For instance, from Person, that is, the last node of the previously found length-1 candidates, the traversal continues (via *expandBody* line 15) and leads to Film and Work. This gives, among the others, the length-2 candidates writer(Work, Person)$\wedge$writer(Person, Work) and writer(Work, Person)$\wedge$director(Person, Work). However, for both candidates, the check at line 18 fails since they do not reach the only valid range Person. When further expanding these candidates the algorithm gets, among the others, the length-3 candidate writer(Work, Person)$\wedge$writer(Person, Work)$\wedge$writer(Work, Person), which passes the check and becomes a valid result (line 19).

We mention a couple of important aspects. First, we exclude the RDFS vocabulary predicates during the traversal of the TBox graph as we are interested in learning rules that are reflected in the ABox. Second, candidate rule bodies include predicates and entity types (i.e., class names). To release the constraint on entity types, RARL can replace the latter with fresh variables. Finally, when domains/ranges are missing RARL considers them to be the abstract concept Thing. Note that in this case, our approach is still useful as it can prune the search space by only considering the top-k related predicates to the input predicate during the traversal of the TBox.

3.2 Verification and Confidence Computation

To build the reduced ABox graph from a candidate rule body Π=$B_1 \wedge B_2 \wedge ...B_d$, Algorithm 3 first translates it into its corresponding traversal query (Definition 1) e=$\bar{p}_1/\bar{p}_2/\bar{p}_d$ (where $\bar{p}_i$ denote a predicate p_i or its inverse $\hat{}p_i$).

From e, it is possible to construct the associated Nondeterministic Finite-state Automaton with ϵ-transitions (Hopcroft, Motwani, and Ullman 2006) $\mathcal{A}_e$, which recognizes strings belonging to the language defined by e. Let Σ_e be the set of predicates that appear in e and $|e|$ its size (number of symbols). The automaton corresponding to e is $\mathcal{A}_e$=$(Q, \Sigma_e, \delta, q_0, F)$, where Q is the set of states, δ:$Q \times (\Sigma_e \cup \{\varepsilon, p_s\}) \rightarrow 2^Q$ the transition function, $q_0 \in Q$ the initial state and $F \subseteq Q$ the set of accepting (final) states. $\mathcal{A}_e$ can be built with costs $O(|e|)$ following the Thomson's construction rules (Hopcroft, Motwani, and Ullman 2006). For instance, Fig. 2 (b) shows a candidate body along with the associated traversal query and automaton. Algorithm 3 builds $\mathcal{G}_e(V^+, V^-)$ by evaluating the expression e on G. We observe that $\mathcal{G}_e(V^+, V^-)$ is a graph where each node is a pair $(id, state)$, where id is the label of a node in the original ABox G and $state$ is the state of $\mathcal{A}_e$ at which the node has been reached during the evaluation of e (see Fig. 2 (c)). The algorithm consists of two main phases:

1. Initialization: A node n_s^* is added to the set of nodes of the ABox graph G; moreover, an edge labeled with the special label p_s, not present in the KB, connects n_s^* to the first element of both positive and negative examples (line 4). This serves the purpose of starting the algorithm from the node n_s^* no matter the sets V^+ and V^-. Fig. 2 (a) shows the ABox after the initialization with the examples

Algorithm 2: *generateCandidateBodies*(p, k, d, G_S, M_R)

```
1  B = ∅; /* priority queue based on rule relatedness */
2  R= getTopKPredicates(p,k)
3  Doms= getDomains(R) /* set of domains of predicates
     in R */
4  Rangs= getRanges(R) /* set of ranges of predicates in
     R */
   // Parallel Execution:
5  for each predicate p_i ∈ R do
6      Let Δ_1 = ∅
7      for (t_r, p_i, t_s) ∈ G_S do
8          if t_r ∈ Doms then
9              Δ_1=Δ_1 ∪ {p_i(t_r, t_s)}
10             if t_s ∈ Rangs then
11                 B = B ∪
                   {p_i(t_r, t_s),getRuleRelatedness(p, p_i)}
           // Same for t_s ∈ Rangs
12     for j = 2, ..., d do
13         Let Δ_j = ∅;
14         for each candidate body π ∈ Δ_{j-1} do
15             for (t_i, p, t_j)∈expandBody(π, G_S, R) do
16                 Δ_j = Δ_j ∪ {π ∧ p(t_i, t_j)} //add atom
17         for each candiate body π_j ∈ Δ_j do
18             if checkRange(π_j,Rangs) then
19                 B=B ∪ {π_j, getRuleRelatedness(p, π_j)}
20 return B
```

Algorithm 3: *getReducedABoxGraph*(G, Π_i, V^+, V^-)

1 e=*getTraversalQuery*(Π) /* Definition 1 */
2 $\mathcal{A}_e = (Q, \Sigma_e, \delta, q_0, F)$ /* NFA associated with e */
3 S=$V^+ \cup V^-$ /* S contains pairs (v_s, v_e) */
4 V=$V \cup \{n_s^*\}$, $E = E \cup \{p_s\}$,
$T = T \cup \{(n_s^*, p_s, v_s)\}, \forall v_s \in S$.
5 Q'= $\{(n_s^*, q_0)\}$
6 δ'= $\emptyset$
7 $\mathcal{G}_e(V^+, V^-)$=$(\{(n_s^*, q_0)\}, \Sigma_e, \emptyset, (n_s^*, q_0), \emptyset)$
8 **foreach** $(n, q) \in Q'$ **do**
9 **foreach** $\delta(q, p) \in \mathcal{A}_e$ **do**
10 **if** p *in* Σ_e **then**
11 **foreach** q' *in* $\delta(q, p) \wedge (n, p, n')$ *in* *traverseEdge*(n, p) **do**
12 Q'= $Q' \cup \{(n', q')\}$
13 add (n', q') to $\delta'((n, q), p)$
14 **if** q'*in* F **then**
15 add (n', q') to F'
16 **return** $\mathcal{G}_e(V^+, V^-)$

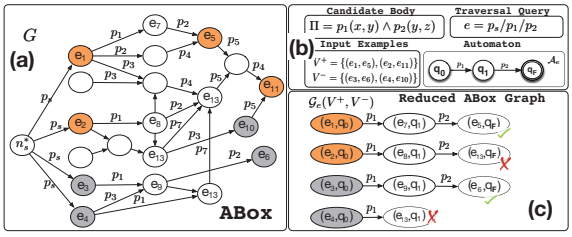

Figure 2: An example of reduced ABox graph computation.

in Fig. 2 (b).

2. Loops: For each pair of (node, state) (n, q) in Q' (line 8), the algorithm considers all outgoing transitions and the transition function. The state (n', q') and the transition $((n, q), p, (n', q'))$ are added to $\mathcal{G}_e(V^+, V^-)$ if and only if $q' \in \delta(q, p)$, $p \in \Sigma_e$ and the triple $(n, p, n') \in$ *traverseEdge*(n, p), which find nodes n' reached in the ABox when traversing edges labeled as p starting from n. If a node n' is reached at a final state q' in F, the pair (n', q') is added to the final states F' of $\mathcal{G}_e(V^+, V^-)$. In Fig. 2, we note that the reduced ABox graph contains in the leftmost part all starting nodes reached at the initial state of $\mathcal{A}_e$. These nodes are linked to other nodes reached at the state q_1 when traversing edges labeled as p_1 (note that p_1 links the state q_0 to the state q_1 in $\mathcal{A}_e$). From e_4 it is not possible to reach a final state; this is because there is no path in the ABox, which starting from e_4 allows reaching any node via the sequence of edges p_1, p_2. Note also that the node (e_{13}, q_F) in $\mathcal{G}_e(V^+, V^-)$ is not useful; this is because from e_2 we are interested in finding a path that leads to e_{11} while the path found allows reaching e_{13}.

Our implementation of Algorithm 3 parallelizes lines 3-17; moreover, it constructs $\mathcal{G}_e(V^+, V^-)$ by only loading into main memory the portion of the ABox G traversed (via repeated calls to the procedure *traverseEdge(n,p)*) when evaluating e (line 11).

Theorem 2 *Let G_e=(V_e, E_e, T_e) the subgraph of the ABox graph G traversed when evaluating the traversal query e starting from n_s^*. The reduced ABox graph $\mathcal{G}_e(V^+, V^-)$ can be built in time $O(|G_e| \times |e|)$ using Algorithm 3.*

The result holds since the maximum number of transitions in $\mathcal{G}_e(V^+, V^-)$ is bound by $|T_e| \times |Q|$ (lines 8-10); moreover additional $|V_e| \times |\mathcal{A}_e|$ can be ϵ transitions. Hence, the maximum number of transitions in $\mathcal{G}_e(V^+, V^-)$ is $|\delta'| = O(|G_e| \times |\mathcal{A}_e|) = O(|G_e| \times |e|)$.

3.3 Rule Confidence Computation

The last step of the RARL algorithm is computing rule confidence. Given a (positive or negative) pair $(\mathsf{e}_i, \mathsf{e}_j)$, the rationale is to check in $\mathcal{G}_e(V^+, V^-)$ whether from (e_j, q_F), with q_F in F', it is possible to reach (e_i, q_0), where q_0 is the initial state, by navigating both $\mathcal{G}_e(V^+, V^-)$ and $\mathcal{A}_e$ backward. For instance, in Fig. 2, for the pair $(\mathsf{e}_1, \mathsf{e}_5)$, the check starts from $(\mathsf{e}_5, \mathsf{q}_F)$; then the algorithm checks whether by traversing the sequence of edges p_2, p_1 it is possible to reach the node $(\mathsf{e}_1, \mathsf{q}_0)$. As this is the case, the pair verifies the candidate body and is added to the set S^+. With the same reasoning, the negative pair $(\mathsf{e}_3, \mathsf{e}_6)$ is added to the set S^-. In this example, no other pair is verified. When α=β=0.5, the confidence of the rule having as body $\mathrm{p}_1(x, y) \wedge \mathrm{p}_2(y, z)$ is 0.5. The verification of pairs to build the sets S^+ and S^- is done in parallel. The algorithm runs in polynomial time as, for each pair to be verified, each state and each transition in $\mathcal{G}_e(V^+, V^-)$ is visited at most once with cost $O(|Q'| + |\delta'|)$.

4 Experiments

We considered four-real world datasets, that is, WN-18RR and FB15-237 used in (Meilicke et al. 2019), and excerpts of Yago3-10 (Yago) (Galárraga et al. 2015) and DBpedia (Shiralkar et al. 2017). For all these datasets we used the portion of the TBox schema including subclass information and domain and range of properties. Details about the datasets are available in Table 1.

Table 1: Datasets characteristics.

	WN-18RR	FB15-237	YAGO	DBpedia
Entities	∼40K	∼14K	∼123K	∼5.5M
Predicates	11	237	37	663
Triples	∼86K	∼272K	∼1M	∼12M
Testset	3K	20K	5K	

Following (Meilicke et al. 2019), we computed the filtered hits@1, filtered hits@10, and the mean reciprocal rank (MRR); we did not compute the filtered MRR as we are only interested in computing top-k ranks only. In this sense, we assume a candidate entity to be an *incorrect* prediction if it is ranked at a position >k. We considered the following default parameter values: d=3 (max. body length), $topPs$=10 (top-10 related predicates), $topC$=80% (percentage of candidate bodies for which we want to compute confidence), α=β=0.5 (weights for the confidence score in equation (2)),

$nExs$=80% (number examples used as a percentage of all available positive facts for a predicate). We also found that the rel_{HB} rule relatedness measure gave the best performance and then consider it in the experiments that follow; the weights of the two components were set to $\gamma = \theta = 0.5$. We implemented RARL in Java and ran experiments on a laptop with 4 cores (each with 2,7 GHz) and 16GB RAM. We pre-computed the (symmetric) relatedness matrices: on the largest dataset DBpedia, the time required was ∼22m.

4.1 Performance Analysis

We tested several aspects of our approach on the largest DBpedia dataset. *First*, to show the benefits of our automata-based algorithm for body verification and confidence computation, we considered a variant of it, which generates candidate bodies via SPARQL queries on the TBox (loaded in memory) and treats the candidates as a conjunctive query. Then, examples are verified in the ABox (loaded on a local endpoint) via Boolean (ASK) SPARQL queries. We refer to this variant of our approach as RARL-SPARQL. Table 2 reports the running times for different values of the d parameter. Running times refer to the discovering of all rules associated with each of the 663 predicates.

Table 2: Comparison between RARL-SPARQL and RARL with $topPs$=10, $nExs$=100%, and $topC$=80%.

	d≤1		*d≤2*		*d≤3*	
	TGen	*TConf*	*TGen*	*TConf*	*TGen*	*TConf*
RARL-SPARQL	11.42	10.46	21.95	31.55	35.47	418.2
RARL	4.12	6.45	11.45	18.32	16.73	121.4

We observe that RARL-SPARQL is always slower than RARL both in terms of candidate generation ($TGen$) and verification and confidence assessment ($TConf$). This difference becomes more evident as the length of the rule (parameter d) grows. The number of candidate bodies verified by at least one positive example was of ∼13.5K. *Second*, to dig deeper into the impact of parameters on the running times of RARL, we conducted additional experiments varying $nExs$ and fixing $topPs$=10 and d=3. We noted that the running time increases almost linearly with the number of examples. For instance, when $nExs$=20% the running time for the computation of confidence was of ∼30 minutes and the number of verified candidates decreased to ∼3K. Indeed, with a lower number of examples, it is less likely that one of them verifies a candidate. *Third*, we conducted additional experiments varying $topPs$. Even in this case, we observed an increase of both the running time and number of bodies verified as $topPs$ increases. For instance, when $topPs$=30 the running time for both the generation and the verification almost doubled and the number of candidates verified reached the value of ∼34K. However, the more predicates are considered during the generation of candidates, the lower will be their rule relatedness, which leads to the generation of rules with lower confidence.

4.2 Performance and Rule Quality Comparison

We compared RARL (Table 3) with two recent systems, that is, anyBURL (Meilicke et al. 2019) and RLvLR (Omran, Wang, and Wang 2018), in terms of num. of rules found (**#R**), num. of high quality rules (**#QR**), where a rule is of high quality if the confidence is ≥0.7 (Omran, Wang, and Wang 2018), and running time in minutes (**Time**). As reported in (Omran, Wang, and Wang 2018), RLvLR performs better than other competitors like AMIE and Ontological Path Finding (Chen et al. 2016); hence, we did not consider them in the evaluation.

Table 3: Rule Learning Comparison on DBpedia.

Approach	**#R**	**#QR**	**Time**
RLvLR 20 predicates	855	183	351
anyBURL(10Ks, sample=1000, d=3)	3609	425	48
anyBURL(10Ks, sample=100, d=3)	20564	3246	91
RARL($topPs$=10, $nExs$=100%, $topC$=80%)	13561	3564	138
RARL($topPs$=5, $nExs$=80%, $topC$=50%)	5465	2132	58

Running time for RLvLR refers to the 20 predicates selected in (Omran, Wang, and Wang 2018). We stopped RLvLR after 6h when using all predicates. For RLvLR we considered the best configuration reported in (Omran, Wang, and Wang 2018). The table shows that RLvLR is the slowest system; indeed, it took 351m to learn rules for 20 predicates only. The reason could be the fact that it uses embeddings (expensive to compute) for rule learning. RARL took less time than RLvLR and considered all DBpedia predicates. As for anyBURL, we considered a time of 10K seconds and set the max. rule length (parameter d) equal to 3, the same as the other systems. When changing the sample size (used for restricting the number of samples drawn for computing confidence) of anyBURL the system kept running for a longer time. By digging into the kind of rules learned, we observed that ∼80% of the high-quality rules learned by anyBURL are of length 1 and include constants (e.g., standardTime(Y, Myanmar Standard Time)⇒birthPlace(Y, Yin Yin Nwe), genre(Dsign Music, Y)⇒genre(Y, Rebelde) having both confidence 1). We observed that longer rules (including 2 or 3 atoms) are much less in number, have much lower confidence values, and do not include constants. The number of (constant-free) rules learned by RLvLR and RARL including 2 or 3 atoms in the body is much larger. When adopting a more restricted configuration for RARL, focusing more on candidates having higher relatedness, we observed a ∼7x speedup in terms of time. Although the number of high-quality rules decreases, it is still higher than that of the other systems. We point out that the decoupling between candidate generation and confidence assessment and the possibility to pick a specific predicate for rule learning is a useful feature. Indeed, one could quickly generate candidate rules (*TGen* is lower than *TConf*), inspect them and proceed to confidence assessment.

4.3 Evaluation on Link Prediction

The goal of this experiment was use rules for link prediction. Given a head $\mathsf{p}(e, \mathsf{X})$, the goal is to predict entities $e' \in G$ that can be bound to the variable X; here, p is a predicate, $e \in G$ and $\mathsf{p}(e, e') \notin G$. As these entities can be suggested by multiple rules, following the methodology in (Meilicke et al. 2019) we order the candidate entities via the maxi-

Table 4: Experiments on Link Prediction.

	WN-18RR			FB15K-237			YAGO		
Approach	**h@1**	**h@10**	**MRR**	**h@1**	**h@10**	**MRR**	**h@1**	**h@10**	**MRR**
ConvE (Dettmers et al. 2018)	39	48	46	23.9	49.1	31.6	45	66	52
ComplEx-N3 (Lacroix, Usunier, and Obozinski 2018)		57	48		56	37		71	58
R-GCN+ (Schlichtkrull et al. 2018)				15.1	41.7	24.9			
CrossE (Zhang et al. 2019)				21.1	47.4	29.9			
AMIE+ (Galárraga et al. 2015)	35.8	38.8		1.4	40.9				
RuleN (Meilicke et al. 2018)	42.7	53.6		18.2	42.0				
RLvLR (Omran, Wang, and Wang 2018)					39.3	24.0		39.3	24
AnyBURL (Meilicke et al. 2019)	44.1	55.2	$\geq$47	23.3	48.6	$\geq$31	47.7	67.3	$\geq$54
RARL	35.1	40.9	$\geq$36	25.12	49.12	$\geq$32	48.2	69.3	$\geq$56
Δ $topPs$=20	-5.2	-2.1	-2	-7.5	-6.4	-5.3	-4.8	-4.5	-3.2
Δ $nExs$=100%	+0.01	+0.11	+0.13	+0.3	+0.2	+0.01	+0.02	+0.04	+0.023

mum of the confidence of all rules that have generated them. If the maximum score of multiple candidate entities is the same we proceed by considering the second-best rule that generated them and so forth until we find a rule that makes a difference. For the evaluation we did not consider the FB-15K and WN18 datasets as they have been criticized for the presence of redundancies; we used their modified versions WN-18RR and FB15K-237 as done by (Meilicke et al. 2019) instead. Besides, we considered Yago (Dettmers et al. 2018). As competitors, we considered approaches using embedding techniques, symbolic techniques and a combination of them recently published. For RARL we started with the configuration with $topPs$=10, $nExs$=80%, $topC$=80% but we also experimented other parameter values and reported the performance variation in the last block of Table 4. The first block shows results for embedding-based techniques. The performance of RARL is already competitive on both FB15K-237 and Yago with the default configuration. We observe, for instance, that on FB15K-237 RARL outperforms competitors (apart from ConvE on hit@1). At the same time, we observe that on WN-18RR, RARL performs worse than all other systems. The reason is that the WN-18RR schema is very simple; it includes a few relations whose domain/range is almost always a Synset. Hence, the rule generated may not be adequate to finally predict missing links. Indeed, we noted that the confidence of rules found by RARL on WN-18RR is usually low. On the other hand, on FB15K and Yago that feature a richer schema RARL behaves quite well. We want to mention that the running time of embedding-based approaches is usually much larger than that of RARL; we were not able to run any of these approaches on the DBpedia dataset within a timeout of 5h (times for RARL are reported in Table 3). Moreover, most of these approaches require the tuning of hyper-parameters via a time-consuming grid search. We ran RARL on a laptop while most of the embedding-based approaches require complex processing infrastructures (e.g., based on expensive GPUs). The other approaches in Table 4 includes AMIE and RuleN, both top-down approach that explore the whole ABox to find rules of bound-length. AMIE is based on (expensive) join operations for both rule learning and confidence computation based on the Partial Completeness Assumption. Indeed, the authors defined an approximation of PCA to reduce the running time. RuleN uses a sampling mechanism to speed-up the learning process. In this group, we also considered anyBURL, which differently from the previous systems adopts a bottom-up approach to learn rules. It allows to learn rules within a certain amount of time given as input. Apart from WN-18RR, RARL outperformed all competitors. Finally, we observe that increasing the number of related predicates usually degrades the performance while increasing the number of examples brings a negligible improvement.

5 Related Work

Most of existing approaches rely on different kinds of ABox-centered algorithms to discover rules. AMIE (Galárraga et al. 2015) explores the ABox search space and computes (approximate) rule quality measures. This hinders its applicability to large KBs. RuDIK (Ortona, Meduri, and Papotti 2018) learns approximate (negative) rules relying on examples. It only considers the ABox and aims at finding a small set of rules that cover the majority of positive and as few negative examples as possible. AnyBURL (Meilicke et al. 2019) learns rules that cover at least some positive and some negative examples. Our approach differs in several ways. First, it starts from the TBox available in many KBs (e.g., DBpedia, Freebase) but overlooked by these approaches. Second, it learns rules driven by a relatedness criterion, prioritizing rules having higher rule relatedness (§ 2.2). To verify candidate bodies in the ABox, RARL relies on a examples like RuDIK. However, while RuDIK focuses on learning the smallest number of rules satisfying the examples, RARL learns every possible rule prioritizing those with high relatedness. RARL is inspired by Ontological Path Finding (OPF) (Chen et al. 2016), which generates candidate rules from the TBox. However, OPF disregards the fact that not all candidates are the same, which leads to generating too many candidates. For instance, for the head $\mathsf{spouse}(\mathsf{X}, \mathsf{Y})$, the candidate rule body $\mathsf{child}(\mathsf{x}, \mathsf{y}) \wedge \mathsf{parent}(\mathsf{y}, \mathsf{z})$ with the predicates child and parent will be more plausible than a body with lyrics or deathPlace since child and parent are more related to the predicate spouse than lyrics or deathPlace. Moreover, our algorithm for confidence assessment has a lower memory footprint as it only loads the portion of the ABox of interest for the verification and thus can scale to large KBs.

Our work differs from embedding-based approaches like

EMBEDRULE (Yang et al. 2015) and RLvLR (Omran, Wang, and Wang 2018) in several respects. First, differently from EMBEDRULE it does not impose the constraint on the body atoms to include different predicates. Second, both approaches focus on simple confidence measures that do not to capture all the subtleties of KB's information (Galárraga et al. 2015); RARL leverages positive/negative examples. Finally, RARL can scale to large KBs.

6 Conclusions and Future Work

We introduced the RARL rule learning approach based on the intuition that rule bodies for an input head can be found by looking at the relatedness between the predicate in the head and those in the bodies. RARL leverages an algorithm, which traverses the TBox for the generation of candidate rule bodies most related to the input head. RARL also features an efficient automaton-driven algorithm that traverses the ABox to verify candidate rule bodies and compute rule confidence. We found that when the KB provides a rich schema RARL offers good performance both in terms of running time and rule quality. We also observed that generating a relatedness-controlled number of rule bodies from the TBox offers an immediate benefit in itself. Indeed, we were able to discover rules by "reading" candidate bodies that our approach ranks by rule relatedness. Projecting RARL in the embedding space is in our research agenda.

Acknowledgment

This work was partially supported by the Ministry of Education, University and Research (MIUR) under the grant "Dipartimenti di Eccellenza 2018-2022" awarded to the Department of Computer Science, Sapienza University of Rome.

References

Chen, Y.; Goldberg, S.; Wang, D. Z.; and Johri, S. 2016. Ontological Pathfinding. In *Proc. of Int. Conf. on Management of Data*, 835–846.

d'Amato, C.; Staab, S.; Tettamanzi, A.; Minh, T.; and Gandon, F. 2016. Ontology Enrichment by Discovering Multi-Relational Association Rules from ontological Knowledge Bases. In *Proc. of Symp. on Appl. Computing*, 333–338.

Darari, F.; Nutt, W.; Pirrò, G.; and Razniewski, S. 2018. Completeness Management for RDF Data Sources. *ACM Trans. on the Web (TWEB)* 12(3):18.

Dettmers, T.; Minervini, P.; Stenetorp, P.; and Riedel, S. 2018. Convolutional 2D Knowledge Graph Embeddings. In *Proc. of AAAI Conference*, 1811–1818.

Fionda, V., and Pirrò, G. 2018. Fact Checking via Evidence Patterns. In *Proc. of International Joint Conference on Artificial Intelligence*, 3755–3761.

Fionda, V.; Pirrò, G.; and Consens, M. P. 2015. Extended property paths: Writing more SPARQL Queries in a Succinct Way. In *Proc. of AAAI Conference*, 102–108.

Fionda, V.; Pirrò, G.; and Gutierrez, C. 2015. NautiLOD: A Formal Language for the Web of Data Graph. *ACM Trans. on the Web (TWEB)* 9(1):5:1–5:43.

Franconi, E.; Gutierrez, C.; Mosca, A.; Pirrò, G.; and Rosati, R. 2013. The Logic of Extensional RDFS. In *Proc. of International Semantic Web Conference*, 101–116.

Galárraga, L.; Teflioudi, C.; Hose, K.; and Suchanek, F. M. 2015. Fast Rule Mining in Ontological Knowledge Bases with AMIE+. *The VLDB Journal* 24(6):707–730.

Gayathri, V., and Kumar, P. S. 2015. Horn-rule based Compression Technique for RDF Data. In *Proc. of Symposium on Applied Computing*, 396–401.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2006. *Introduction to automata theory, Languages, and Computation*, volume 32. Addison-Wesley.

Lacroix, T.; Usunier, N.; and Obozinski, G. 2018. Canonical Tensor Decomposition for Knowledge base Completion. In *Proc. of Int. Conf. on Machine Learning*, 2869–2878.

Meilicke, C.; Fink, M.; Wang, Y.; Ruffinelli, D.; Gemulla, R.; and Stuckenschmidt, H. 2018. Fine-grained Evaluation of Rule-and Embedding-based systems for Knowledge Graph Completion. In *Proc. of Int. Sem. Web Conf.*, 3–20.

Meilicke, C.; Chekol, M. W.; Ruffinelli, D.; and Stuckenschmidt, H. 2019. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In *Proc. of International Joint Conference on Artificial Intelligence*, 3137–3143.

Munoz, S.; Pérez, J.; and Gutierrez, C. 2009. Simple and Efficient Minimal RDFS. *J. of Web Semantics* 7(3):220–234.

Omran, P. G.; Wang, K.; and Wang, Z. 2018. Scalable Rule Learning via Learning Representation. In *Proc. of Int. Joint Conference on Artificial Intelligence*, 2149–2155.

Ortona, S.; Meduri, V.; and Papotti, P. 2018. Robust Discovery of Positive and Negative Rules in Knowledge-Bases. In *Proc. of Int. Conf. on Data Engineeering*, 1168–1179.

Pirrò, G. 2012. REWOrD: Semantic Relatedness in the Web of Data. In *Proc. of AAAI Conference*, 129–135.

Pirrò, G. 2019. Building Relatedness Explanations From Knowledge Graphs. *Semantic Web* 1–28.

Schlichtkrull, M.; Kipf, T.; Bloem, P.; Van Den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling Relational Data with Graph Convolutional Networks. In *Proc. of European Semantic Web Conference*, 593–607.

Shi, B., and Weninger, T. 2016. Discriminative Predicate Path Mining for Fact Checking in Knowledge Graphs. *Knowledge-Based Systems* 104:123–133.

Shiralkar, P.; Flammini, A.; Menczer, F.; and Ciampaglia, G. L. 2017. Finding Streams in Knowledge Graphs to Support Fact Checking. In *Proc. of International Conference on Data Mining*, 859–864.

Tanon, T. P.; Stepanova, D.; Razniewski, S.; Mirza, P.; and Weikum, G. 2017. Completeness-Aware Rule Learning from Knowledge Graphs. In *Proc. of International Semantic Web Conference*, 507–525.

Yang, B.; Yih, W.-T.; He, X.; Gao, J.; and Deng, L. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proc. of International Conference on Learning Represenation*.

Zhang, W.; Paudel, B.; Zhang, W.; Bernstein, a.; and Chen, H. 2019. Interaction Embeddings for Prediction and Explanation in Knowledge Graphs. In *Proc. of International Conference on Web Search and Data Mining*, 96–104.

Zupanc, K., and Davis, J. 2018. Estimating Rule Quality for Knowledge Base Completion with the Relationship Between Coverage Assumption. In *Proc. of The Web Conference*, 1–9.