Towards Relational Compression on Knowledge Bases

Ruoyu Wang[1,2,3] , **Daniel Sun**[2,3,4] , **Guoqiang Li**[1] , **Raymond Wong**[2] , **Shiping Chen**[4]
[1]Shanghai Jiao Tong University, Shanghai, China
[2]University of New South Wales, Sydney, Australia
[3]Enhitech LLC., Shanghai, China
[4]Data61, CSIRO, Australia
{wang.ruoyu, li.g}@sjtu.edu.cn, {daniel.sun, shiping.chen}@data61.csiro.au, wong@cse.unsw.edu.au

Abstract

Within the realm of knowledge bases, Relational Knowledge Bases (RKBs) emphasises on relational knowledge that from the real world integrates semantic and syntactic information for decision making, reasoning, and higher cognition. In real-world applications, relations are hardly acquired from formal models, and instead human factors are involved in relation extraction, e.g. determining ontology. One cannot guarantee that the essence of relations has been achieved, no mention of merging or expanding RKBs. In this paper, we focus on reduction and refinement of relations such that not only RKBs can be simplified but also be compressed semantically, i.e. relational compression. To achieve our goal, in the following we first define and formalise relational compression, and then we propose a novel solution to crack the hard core. Our solution is evaluated with real life RKBs, and the results show the compression ratios from 1.3 to 2.2 at the current stage.

1 Introduction

Relational Knowledge Bases (RKBs) are focused on relational knowledge data, which complies with certain forms of logic (Müller 2012; Codd 2002), and has been broadly used in research and industrial systems, especially in managing domain-specific knowledge. As technologies are fast evolving, data amount increases dramatically, such that complexity and size of RKBs have created new challenges to applications (Sachan 2020). Although meta-information such as ontologies can be used to depict RKBs, text patterns are usually with human factors such that relation redundancy in semantic sense is inevitable for large RKBs. Therefore, relational compression has valuable merits in practice.

Below shows an example of relational compression in real life. Table 1 shows part of relations in a knowledge base about simplified PC configurations and status. For a PC to boot successfully, hardware shall be consistent and power supply shall be sufficient. From this KB one may conclude that: 1) memory rate of the chipset shall be supported by memory chips; 2) protocol should match between chipset and storage devices; 3) power is turned on. This observation can be written as a Horn rule shown in Rule (1). Therefore, Table 1e can be simplified to Table 2.

The example shows that relations can be simplified and reduced. A straightforward benefit is that KBs can be compressed semantically. Traditional bit compression is able to shrink the storage size of a given KB too, but extra access overhead has to be traded off. Another benefit from the above compression is that compressed relations are queriable by logic inference with learned rules.

Inductive Logic Programming (ILP) is a tool of inducing logic rules from background knowledge and is useful for our goal, but cannot be applied directly for compression. Firstly, its top-down strategy applies changes to candidate rules from the syntactic aspect. The strategy fails to show semantic length of rules, which heavily impacts on compression effect. Secondly, some existing ILP systems (Zeng, Patel, and Page 2014; Yin et al. 2006) require negative examples which cannot be listed specifically, while others focus only on KGs for scalability (Galárraga et al. 2013), which cannot handle multi-arity relations. Thirdly, modern ILP systems (Schoenmackers et al. 2010) enumerate as many rules as possible to aviod missing information and those rules may overlap with each other, while for compression no overlapping rules are expected.

As far as our knowledge, we are the first to compress RKBs by semantic induction. In this paper we define and formalise relational knowledge compression. After that, we exploit important properties of the compression, design a fundamental compression procedure, and deal with above weaknesses of existing ILP systems. Our compression works well on real datasets with the compression ratio between 1.3 and 2.2, and the reduction ratio between 1.3 to 2.4.

Contributions of this paper include:

- An formalization of relational compression on RKBs and proof of its NP-hardness;
- An effective solution and a new search procedure of top-down induction;
- An implementation, on which experiments and evaluation are carried out.

The remaining is organised as follows: Section 2 introduces related work on knowledge compression and other applications our method may be adopted. A formal definition of the problem is given in Section 3 as well as the fundamental solution. Section 4 proposes in detail the search technique to find a best rule in knowledge base and Section 5 shows experimental results of our approach. Finally, Section 6 concludes the entire paper.

Table 1: Examples on Some PC Configurations and Status

(a) memory/2

id	highestRate
pc200	2400
pc300	2666
stone	3200
lense	4000
xvt	2666

(b) chipset/3

id	memRate	stgPtcl
pc200	2666	ide
pc300	2666	ahci
stone	2400	ahci
lense	2666	nvme
xvt	1666	ide

(c) storage/3

id	protocol	volume
pc200	ide	512
pc300	ahci	1024
stone	ahci	2048
lense	nvme	256
xvt	ahci	512

(d) powerOn/1

id
pc200
pc300
stone
lense

(e) status/2

id	status
pc200	error
stone	on
lense	on
xvt	off

$$status(X, on) \leftarrow chipset(X, Y, Z), memory(X, Y_h), Y \leq Y_h, storage(X, Z, W), powerOn(X) \quad (1)$$

Table 2: Simplified *status* Table

id	status
pc200	error
xvt	off

2 Related Work

2.1 Knowledge Simplification

Logic program minimization (Hammer and Kogan 1995a; Boyar, Matthews, and Peralta 2013) is an early attempt for knowledge simplification. The minimization problem was interpreted as Boolean function minimization and has been proved NP-Complete. Some researches (Hammer and Kogan 1995b; Boros et al. 2009) aims at finding subsets of Horn logic under which the minimization of logic programs is solvable in polynomial time. Recently, knowledge graphs have been represented as embedding matrices (Bordes et al. 2013) and is further compressed via a pair of discretization and reverse-discretization functions (Sachan 2020). Graph compression systems, such as GASTON (Nijssen and Kok 2004) and SUBDUE (Holder et al. 1994), can also be applied to compress frequent subgraphs in KGs. Knowledge distillation (Wang et al. 2019) is another methodology that transforms model information into a compressed form. In some experiments (Yim et al. 2017), the compressed model even outer performs the original ones.

However, the approaches based on embeddings and knowledge distillation do not provide enough interpretability. KG compression is still in an infant stage. Logic program minimization is explainable but lacks of real world applications and thus not widely adopted.

2.2 Inductive Logic Programming

Inductive logic programming (ILP) had been studied before 1990s, and was formally proposed in 1991 (Muggleton 1991). Given both positive and negative examples, as well as some probably useful background knowledge, the goal of ILP is to induce rules that explains positive examples and excludes negative ones (Quinlan 1990; Muggleton 1995). Besides this goal, ILP was also used to solve classification problems (Blockeel and De Raedt 1998; Domingos 1996; Cohen 1995a; Cohen 1995b; Mooney 1999; Califf and Mooney 2003). Modern ILP systems (Schoenmackers et al. 2010; Galárraga et al. 2013) use rules for prediction and error detection.

Although general Horn programs are not learnable (Marcinkowski and Pacholski 1992; Schmidt-Schauss 1988; Hanschke and Würtz 1993), ILP algorithms are applicable in several domains with language restrictions (e.g. generativiy) (Niblett 1988; Kietz and Džeroski 1994). ILP algorithms (Quinlan 1990; Muggleton 1995; Zeng, Patel, and Page 2014; Yin et al. 2006) require a target predicate or concept appointed as input, and negative examples are also needed so as to become irrealistic in modern large scale knowledge bases. In our method, neither target nor negative set is necessary. The rules best describing entire knowledge base are enumerated iteratively under *Closed World Assumption (CWA)* until sufficient relations are entailed.

Typical strategies in ILP systems are top-down (Quinlan 1990) and bottom-up (Muggleton 1995), with performing induction from general to specific or in the reverse order respectively. Our solution leverages an improved top-down routine that searches in a well-defined search space. Our procedure is also able to generalise candidates for necessity.

2.3 Probabilistic Rule Learning & Inference

Purely logic or rule based inference systems, such as expert systems, are often unapplicable since data noise is inevitable in real world circumstances. Researchers have been trying to introduce probability or fuzziness into logic systems. Markov Logic Networks (Richardson and Domingos 2006) and Bayesian Logic Programs (Kersting and De Raedt 2001) are two popular methodologies for this purpose. Evans and Grefenstette (Evans and Grefenstette 2018) defined a differentiable ILP procedure, in which a continuous function is used to perform inference. Our solution learns Horn rules with a necessary counter example set, thus it is also able to induce the probability that each rule infers correct answer given certain input.

3 Finding Essential Knowledge

Relational compression implies removal of unnecessary relations in RKBs. Given a set of rules, some relations can be inferred from others, thus can be omitted and regenerated if

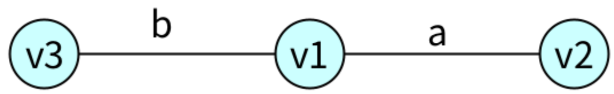

Figure 1: Vertex Cover Example

needed. Therefore, essential knowledge consists of unprovable relations and all rules involved for the inference. For example, Table 1a to 1d, Table 2 and rule (1) are all essential. In this section, we formalise relational compression and determine essentail knowledge by enumerating all induced rules and unprovable predicates.

3.1 Problem Definition

Let L_B be the set of all atomic, grounded predicates and L_H be the set of all atomic first-order Horn rules. Let C be a finite set of constant symbols, $B \subseteq L_B$, a finite set, be the original knowledge base, N and R be two subsets of B and $H \subseteq L_H$ be a finite set of hypothesis. $A \subseteq L_B$ is the set of counter examples. The problem of finding maximal compression ratio of relational knowledge bases is to find H, N, R, A, such that:

- $N \cap R = \varnothing, N \cup R = B$
- $A \cap B = \varnothing$
- $N \wedge H \models R \cup A$
- $|N| + |A| + |H|$ is minimal

where $|N|$ is the number of predicates in N, and so be $|A|$. $|H|$ is defined as the sum of the numbers of minimal equivalence conditions assigned in all rules, and more detail about this definition is in Section 4.3.

Concretely our goal is to use as few rules as possible, which are inducible from B, to entail as many predicates in B as possible, and meanwhile, to generate A as small as possible.

3.2 Problem Complexity

Definition 1 (Minimum Vertex Cover Problem). *Let $\mathcal{G}_{vc} = \langle V_{vc}, E_{vc}\rangle$ be an undirected graph. A minimum vertex cover V_c of $\mathcal{G}_{vc}$ is a minimum subset of V_{vc} such that $(u, v) \in E_{vc} \implies u \in V_c \vee v \in V_c$.*

Complexity of relational compression can be proved by reducing minimum vertex cover problem to relational compression. Let $\mathcal{G}_{vc} = \langle V_{vc}, E_{vc}\rangle$ be the graph in the vertex cover problem. By the following settings we create a relational knowledge base aligning with $\mathcal{G}_{vc}$: Let v be a unary predicate in B for each $v \in V_{vc}$; let $edge$ be a unary predicate in B for edges; add two constants e_{ij} and e'_{ij} to C and six predicates $edge(e_{ij}), edge(e'_{ij}), v_i(e_{ij}), v_i(e'_{ij}), v_j(e_{ij}), v_j(e'_{ij})$ to B for each $(v_i, v_j) \in E_{vc}$; add the following predicates to B: $edge(c_1), edge(c_2), \ldots, edge(c_{2\cdot|E_{vc}|+1})$; and add the following constants to C: $d_1, d_2, \ldots, d_{4\cdot|E_{vc}|+1}$.

For example, Figure 1 shows a graph with three vertices and two edges. The corresponding setting of relational compression is as follows:

- $C = \{a, a', b, b', c_1, \ldots, c_5, d_1, \ldots, d_9\}$
- $B = \{v_1(a),\ v_1(a'),\ v_2(a),\ v_2(a'),\ v_1(b),\ v_1(b'),\ v_3(b),\ v_3(b'),\ edge(a),\ edge(a'),\ edge(b),\ edge(b'),\ edge(c_1),\ \ldots,\ edge(c_5)\}$

By reducibility from minimum vertex cover problem to relational compression we can prove the latter is NP-hard. The details are as follows:

Lemma 2. *$edge(X) \leftarrow true$ is not in H.*

Proof. Let $arg^+(p) = \{c \in C|p(c) \in B\}$, then $|arg^+(edge)| = 2n + 2n + 1 = 4n + 1$, where $n = |E_{vc}|$. Thus the number of predicates this rule entails is $4n+1$. Taking constants $d_1, \ldots, d_{4n+1}$ into consideration, the number of counter examples this rule entails is also $4n + 1$. The size reduced is $4n+1-(4n+1)-1 = -1$, no actual reduction. Therefore, it does not reduce the size of knowledge base. It is not in H. □

Lemma 3. *Predicates of $edge$ can only be entailed by the following rules: $edge(X) \leftarrow v_i(X)$.*

Proof. Let rule r_i be: $edge(X) \leftarrow v_i(X)$, the length of which is 1. Then the number of predicates it entails is $2k$, where k is the number of edges connected to vertex v_i. There are no counter examples entailed by this rule. Thus the size it reduces is $2k - |r_i| = 2k - 1$. If $k \geq 1$, this rule can be used to reduce the size of knowledge base.

According to Lemma 2, $edge$ cannot be entailed by axioms, and since there is no other predicate in B, $edge$ can only be entailed by some v_i. □

Lemma 4. *Let $S = \{edge(e)|edge(e) \in B\} \setminus \{edge(c)|\exists c_i = c\}$. All predicates in S are provable after compression. That is, $S \subseteq R$, where R is the set of all provable predicates.*

Proof. According to Lemma 3, proof of $edge(e) \in S$ relies only on predicates of v_i. No matter predicates of v_i is provable or not, the rules of $edge(X) \leftarrow v_i(X)$ can always be applied to prove $edge(e) \in S$. Suppose $\exists edge(e) \in S$ such that $edge(e) \notin R$. Then there is another predicate $edge(e') \in S$ and $edge(e') \notin R$, where e and e' correspond to some edge in E_{vc} and its duplicate, since these two predicates are both entailed by some rule $edge(X) \leftarrow v_i(X)$ if one of them is entailed by the rule. Then a new rule can be applied to entail these two predicates to further reduce the size of given result. However, according to definition of relational compression, output cannot be further reduced. Contradiction occurs. □

Lemma 5. *Let V_c be the solution of minimum vertex cover problem. Let H_{vc} be a rule set and $H_{vc} = \{edge(X) \leftarrow v(X)|v \in V_c\}$. Let $\bar{H}_{vc}$ be a rule set and $\bar{H}_{vc} = \{edge(X) \leftarrow v(X)|v \notin V_c\}$. Then $H_{vc} \subseteq H$ and $\bar{H}_{vc} \cap H = \varnothing$.*

Proof. According to Lemma 3 and 4, all edges are provable and only provable by vertices, and this is equal to the setting that all edges are covered and only covered by vertices for

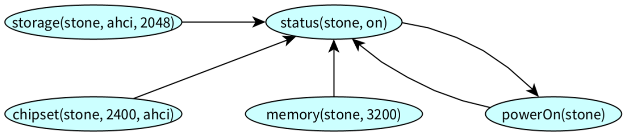

Figure 2: Example of Cycles in $\mathcal{G}$

minimum vertex cover problem. Thus H_{vc} entails S in a minimum cost. $H_{vc} \subseteq H$ and $\bar{H}_{vc} \cap H = \varnothing$. □

Theorem 6. *Relational compression is NP-hard.*

Proof. Let V_c be the set of minimum vertex cover of $\mathcal{G}_{vc}$. According to the lemmas above, $V_c = \{v \in V_{vc} | \exists edge(X) \leftarrow v(X) \in H\}$. All the operations involved with reducibility are with polynomial cost. Thus minimum vertex cover problem can be polynomially reduced to relational compression. Relational compression is NP-hard. □

3.3 Enumerating Unprovable Predicates

To find the unprovable predicates, an intuitive approach is to iteratively find hypothesis that reduces total size, until there is no more hypothesis to further compress. Then according to H, entailment dependencies among predicates can be figured out by grounding each rule (i.e. replacing variables by constants in C).

Each $r \in H$ can be grounded such that every predicate in its body is satisfied by B. Then the corresponding head[1] is said to be entailed by these body predicates with respect to rule r. Let $\mathcal{G} = \langle V, E \rangle$ be a graph, where each vertex in V represents a predicate in B. An edge (b, h) is in E if h is entailed with respect to some rule r where h, b are in the same grounding of r, and b is in its body. Specifically, if the body of r contains no predicate, grounded head can be linked from a new vertex "$\perp$" in $\mathcal{G}$. Therefore, every vertex h in $\mathcal{G}$ can be entailed by all its neighbours that have an edge pointed to h. Then $\mathcal{G}$ depicts entailment dependencies with respect to H. Thus N is set to all vertices with zero indegree. Namely, $N = \{h | \forall b \in V, (b, h) \notin E\} \setminus \{\perp\}$.

However, there may still be predicates not provable by such N if there exists cycles in $\mathcal{G}$. For instance, Rule (2) is also inducible in the PC knowledge base:

$$powerOn(X) \leftarrow status(X, on) \tag{2}$$

These two rules can be grounded by predicates about computer "stone" and generate a dependency graph shown in Figure 2. *status(stone, on)* and *powerOn(stone)* both have non-zero indegree. Thus $N = \{memory(stone, 3200), chipset(stone, 2400, ahci), storage(stone, ahci, 2048)\}$. But neither of those two predicates is provable from N. In such circumstances, at least one vertex in each cycle should be selected into N. However, enumerating all cycles in the dependency graph is expensive. In our solution, *Strongly Connected Components (SCCs)* are detected instead and all vertices of them are added to N. However, this fails to reduce size of B in the worst case. For example, if the only rule in H is about symmetry (i.e. $p(X, Y) \leftarrow p(Y, X)$ for some predicate p), then every predicate is either unprovable or in some SCC. No predicate is removed in this circumstance.

We improve selection of vertices in SCCs in two aspects. Firstly, the number of cycles in $\mathcal{G}$ can be reduced. If a predicate can be entailed by multiple groundings of rules, then only one is remained in $\mathcal{G}$. Secondly, selecting vertices to cover cycles in SCCs can be optimized. This optimization is studied in *Feedback Vertex Set* problem, which is NP-hard (Chen et al. 2008). Our solution adopts a greedy approach to select vertices in each SCC: in each iteration, the algorithm selects a vertex that maximizes $|v_{in}| \times |v_{out}|$ ($|v_{in}|$ and $|v_{out}|$ are indegree and outdegree of vertex v respectively) and then removes cycles that are covered by this vertex.

This procedure is detailed in Algorithm 1, in which $findBestRule(B_t, B)$ is a key step and expanded in next section.

Algorithm 1 Relational Compression

Input: Knowledge Base B
Output: Compressed knowledge base: (N, R, A, H)

$B_t \leftarrow B$
$H \leftarrow \varnothing$
$A \leftarrow \varnothing$
$\mathcal{G} \leftarrow \langle B \cup \{\perp\}, \varnothing \rangle$
while $true$ **do**
    $r \leftarrow findBestRule(B_t, B)$
    if r cannot reduce size of B_t **then**
        break
    end if
    $H \leftarrow H \cup \{r\}$
    $A \leftarrow A \cup \{$negative entailments of $r\}$
    Remove positive entailments of r in B_t
    Update graph $\mathcal{G}$ with respect to r
end while
$Scc \leftarrow$ all strongly connected components in $\mathcal{G}$
$fvs \leftarrow$ FVS of Scc
$N \leftarrow \{h | \forall b \in V, (b, h) \notin E\} \cup fvs \setminus \{\perp\}$
return $(N, B \setminus N, A, H)$

4 Finding the Best Rule

This section introduces an improved ILP procedure for one single rule that entails most predicates in current knowledge base. Off-the-shelf ILP tools, such as QuickFOIL (Zeng, Patel, and Page 2014) and AMIE (Galárraga et al. 2013), do not serve our goal:

- Rule length does not reflect semantic information;
- Negative examples are required explicitly and this does not match CWA;
- Most rules in hypothesis are overlapping.

To solve these problems, we proposed a new search technique as follows.

[1]Head of a Horn rule is the result predicate of it; its body contains condition predicates. For example, head of rule (1) is $status(X, on)$, others are the body.

4.1 Search Space for Rules

Original search space for ILP problems consists of first order Horn clauses. To ensure learnability and fast convergence, state-of-the-art algorithms put several restrictions on the search space, including atomic, generative, closed, maximum rule length and maximum arity of predicates (Quinlan 1990; Pazzani and Kibler 1992). Since predicates in RKBs are atomic and grounded, we restrict the basic search space of our algorithm to be all atomic Horn rules[2]. To guarantee generality of induced rules, we do not impose other restrictions on the search space. Rule (1) in section 1 and Rule (2) in section 3.3 are two examples. The basic search space is noted as Ω. However, some subtypes of rules provide no information for compression, thus should be removed from Ω. Searching for some of them can be avoided, and the others should be removed by ad hoc checks. The remaining search space is written as Ω_m.

Trivial Rules and Corresponding Consequences Trivial rules are with at least two identical predicates. For instance:

$$\begin{aligned} status(X, on) \leftarrow & powerOn(X), status(X, on), \\ & memory(X, Y) \end{aligned}$$

Rules of this type are useless since they have redundant literals and may introduce self-loops in $\mathcal{G}$. Their logical consequences are also useless for the same reason and should be removed.

Rules with Independent Fragments in Body Independent fragment in the body of certain rule r is a subsequence of the body that shares no variable with the remaining part of r (including the head of r). For example:

$$\begin{aligned} powerOn(X) \leftarrow & status(X, on), \\ & memory(Y, Z), storage(Y, W, M) \end{aligned}$$

$\{memory(Y, Z), storage(Y, W, M)\}$ is an independent fragment of the above rule. This type of rules make nonsense since the variables in this subsequence are bounded by existing quantifier, and:

- if the predicates in the fragment are satisfiable, then the rule is logically equivalent to the one without the independent fragment;
- if the predicates in the fragment are unsatisfiable, then the entire body of the rule will never be satisfied.

Neither of above cases is informative.

4.2 Extension of Rules

We define an extension operation that links a rule r with others in Ω_m. The result of this operation is written as $ext(r)$. A rule can be extended by an existing or new bounded variable, or a constant. We order $ext(r)$ excludes trivial rules.

[2]Predicates in atomic Horn rules contains only variables or constants as arguments

Extension by A Bounded Variable The extension turns a free variable to a bounded one in r, or adds a new predicate with free variables to r and turns one of these new free variables to a bounded variable.

For example, let r be:

$$status(X, ?) \leftarrow powerOn(X) \tag{3}$$

(each '?' stands for a unique free variable) The following are some examples of extension in this type:

- $status(X, X) \leftarrow powerOn(X)$
- $status(X, ?) \leftarrow powerOn(X), memory(X, ?)$
- $status(X, ?) \leftarrow powerOn(X), memory(?, X)$
- ...

Extension by A New Bounded Variable This extension turns two different free variables in r or r' as a new bounded variable, where r' is generated by adding a new predicate with free variables to r. To avoid independent fragments, these two variables shall not both be in the newly added predicate. Some extensions of Rule (3) via a new bounded variable are as follows:

- $status(X, Y) \leftarrow powerOn(X), memory(Y, ?)$
- $status(X, Y) \leftarrow powerOn(X), memory(?, Y)$
- ...

Extension by A Constant This extension turns a free variable in r as a constant in C. The following shows some extensions of Rule (3):

- $status(X, on) \leftarrow powerOn(X)$
- $status(X, pc200) \leftarrow powerOn(X)$
- ...

According to the rule extension, $\forall r, r_e \in \Omega_m$, if $r_e \in ext(r)$, then r_e is the extension of r, and r is the origin of r_e (denoted as $r \in ext^{-1}(r_e)$ since one may have multiple origins). Neighbours of a rule in Ω_m consist of all its extensions and origins. The above extension operations can be used to search on Ω_m. Let $S = \{r | r$ has only a head predicate p and all arguments of p are free variables$\}$, every element in Ω_m can be searched from some $r_0 \in S$ (*Search Completeness*), to prove which we define a property *link* between predicates in a certain rule: If two predicates p and q in a rule r share a bounded variable X, then p and q are linked by X in r, written as $p \diamond_X q$, or in short $p \diamond q$. Moreover, if there is a sequence of predicates $p \diamond w_0 \diamond \cdots \diamond w_k \diamond q$, then there is a *linked path* between p and q, written as: $p \leftrightarrow^{\diamond} q$. With this property, we can prove the search completeness as follows:

Lemma 7. *$\forall r \in \Omega_m$, every predicate in r has a linked path with the head of r.*

Proof. Suppose a predicate p in rule r has no linked path with the head. Then p is not itself the head. Let $P = \{q | p \leftrightarrow^{\diamond} q\}$, every predicate in P has no linked path with the head. Then the fragment noted by P does not share any variables with remaining predicates. Namely, P denotes an independent fragment in rule r. According to the definition of Ω_m, we have $r \notin \Omega_m$, which contradicts with $r \in \Omega_m$. $\square$

Theorem 8. *(Search Completeness)There is a set S of starting elements in Ω_m, $\forall r \in \Omega_m, \exists r_0, r_1, \ldots, r_n \in \Omega_m$, such that $r_0 \in S, r_1 \in ext(r_0), \ldots, r \in ext(r_n)$.*

Proof. Suppose $p \diamond_X q$ in r. During the searching process of r, when p is already in a intermediate status r', an extension of r' can be constructed by adding a new predicate q and turning corresponding variables to X. Thus predicate q is introduced into r'. Therefore, if $w \leftrightarrow^{\diamond} q$ and w is already in a intermediate status, then q can be introduced into r'. According to Lemma 7, all predicates in r has linked path with its head. Each predicate can be introduced into the rule iteratively starting from the head predicate where arguments are all different free variables. Other bounded variables and constants can be added to the rule by other extension operations to finally construct r. As a result, $S = \{r|r$ has only a head predicate p and all arguments of p are free variables$\}$ is the start set for searching Ω_m. □

Rules with independent fragments will not be constructed starting from S, as the extension operations do not introduce new predicates without any shared variables with other predicates.

4.3 A New Routine to Find the Best Rule

Quality of rules impact heavily on the effect of compression. The impact can be quantified by several metrics defined on: $\Omega_m \to \mathcal{R}$.

1. Absolute Reduction Size:
$$\delta(r) = |E_r^+| - |E_r^-| - |r|$$
2. Compression Rate:
$$\tau(r) = \frac{|E_r^+|}{|E_r^+| + |E_r^-| + |r|}$$

E_r^+ and E_r^- are the set of positive and negative entailments of r respectively. $|r|$ is the length of rule r. Looking into the definitions, when $\delta(r) \leq 0$ or $\tau(r) \leq 0.5$, r does not reduce the size of B.

The length of a rule in our technique is defined as the minimum number of equivalence restrictions that describe the rule. For example, to describe the rule: $powerOn(X) \leftarrow status(X, on)$, we need two equivalence restrictions: the first argument of $powerOn$ is equal to the first argument of $status$ and the second argument of $status$ is equal to the constant on.

The best rule that maximizes $\delta(r)$ or $\tau(r)$ is reached by Algorithm 2 ($eval(r)$ is $\delta(r)$ or $\tau(r)$), which adopts hill climbing strategy.

Algorithm 2 $findBestRule$

Input: Evaluation knowledge base for head: B_h
Input: Entire knowledge base B
Output: Best rule for compression

$r \leftarrow \arg\max_{p \in S} eval(p \leftarrow true)$
while $true$ **do**
  $L \leftarrow \varnothing$
  for each $r_n \in ext(r) \cup ext^{-1}(r)$ **do**
    if $eval(r_n) > eval(r)$ **then** ▷ B_h and B are used in $eval(\cdot)$ for calculating both positive and negative entailments
      $L \leftarrow L \cup \{r_n\}$
    end if
  end for
  if $L = \varnothing$ **then**
    return r
  else
    $r \leftarrow \arg\max_{r_n \in L} eval(r_n)$
  end if
end while

5 Evaluation

In this section we test our solution under both synthetic and real world relational datasets:

Elti [3] This dataset describes instances of a special kinship: elti, where husbands of two people are brothers. Other related kinship facts are also provided.

StudentLoan StudentLoan contains data about students enrollment and employment status.

DBpedia-factbook This dataset is a subset of latest DBpedia, containing information of languages and countries that speak certain languages.

Family(Simple) [4] This is a synthetic dataset containing simple family relations of multiple families.

Family(Medium) This is a synthetic dataset containing more family relations of multiple larger family instances.

Our solution is implemented in Java 11 with single thread. All tests were executed on Deepin Linux (kernel: 5.4.70-amd64-desktop) with Ryzen 3600.

5.1 Compression Ratio

Table 3 shows compression results on each dataset with detailed parameters. Reduction ratios and compression ratios are given by the following formulas:

$$Reduction\ Ratio = \frac{|B|}{|N| + |A|}$$

$$Compression\ Ratio = \frac{|B|}{|N| + |A| + |H|}$$

The results show that our technique achieves significant reduction and compression ratios on all datasets. Compression ratios are proportional to reduction ratios. And relation reduction dominates the effects of compression.

Larger compression ratios imply more and longer rules, as longer rules define complex concepts with fewer counter

[3]Both Elti and StudentLoan datasets are accessible in: https://relational.fit.cvut.cz/

[4]Synthetic datasets are generated by tools in this project. Source code of this project is accessible in: https://github.com/TramsWang/KBCore/

Table 3: Compression Details

Dataset	$\#F$	$\|B\|$	$\|N\|$	$\|A\|$	$\#H$	$\|H\|$	$\#SCC$	$\|SCC\|$	$\|FVS\|$	**Reduc. Ratio**	**Comp. Ratio**	**Time(s)**
Elti	11	271	155	0	10	25	21	42	21	1.75	1.51	5.12
StudentLoan	9	5283	3599	374	3	4	0	0	0	1.33	1.33	2450.31
DBpedia-factbook	2	545	315	105	1	1	0	0	0	1.30	1.29	0.36
Family(Simple)	4	240	100	0	4	8	0	0	0	2.4	2.22	0.78
Family(Medium)	9	1100	443	120	16	41	70	250	113	1.95	1.82	24.39

Some columns are: number of relation types ($\#F$), number of rules ($\#H$), number of strongly connected components in dependency graph ($\#SCC$), vertices involved in strongly connected components ($|SCC|$), number of vertices in FVS ($|FVS|$).

examples. For example, the following rule defines predicate "father" in "Family(Medium)" dataset:

$$father(X0, X1) \leftarrow parent(X0, X1), brother(X0, ?)$$

$parent(X0, X1)$ restricts $X0$ to be father or mother of $X1$ and $brother(X0, ?)$ suggests $X0$ being male. More rules define concepts in different aspects-that is-more predicates are covered by hypothesis. For example, predicates of relation $parent$ are covered by $father$ together with $mother$. Datasets that generates high compression ratios contain few counter examples. "Family(Medium)" generates many counter examples because an important predicate, which reflects two persons are not identical, is not provided by the dataset. The missing predicate leads to most of its counter examples.

Moreover, larger compression ratios usually appear along with cycles in the dependency graph. Cycles in the dependency graph means that definitions of concepts rely on each other, such that more predicates are provable under the same essential knowledge. For example, $husband$ and $wife$ are mutually defined in "Elti" datatset via the following two rules, which yields 21 cycles of vetex pairs:

$$husband(X1, X0) \leftarrow wife(X0, X1)$$
$$wife(X0, X1) \leftarrow husband(X1, X0)$$

"Family(Simple)" does not generate cycles because it contains only 4 different types of predicates and none of them are mutually defined by each other. From those with cycles we can find that SCCs are usually small but appear frequently in the dependency graph. The simple FVS procedure in our solution is sufficient to cover those cycles in the dependency graph.

Table 4 displays compression results under different metric functions. The results show that the compression ratios are almost the same under these two functions. But the algorithm under $\delta(r)$ usually runs faster than under $\tau(r)$. This is because $\tau(r)$ usually tries more possibilities. However, different metrics may order neighbours of a certain candidate differently, and this results in different body selections in certain cases. This is why $\delta(r)$ and $\tau(r)$ behave differently in "Elti" and "Family(Medium)". For example, under $\tau(r)$, 27 predicates of $sister$ are proved by the following rules in "Elti" dataset:

$$sister(X0, X1) \leftarrow daughter(X0, ?), brother(X1, X0)$$

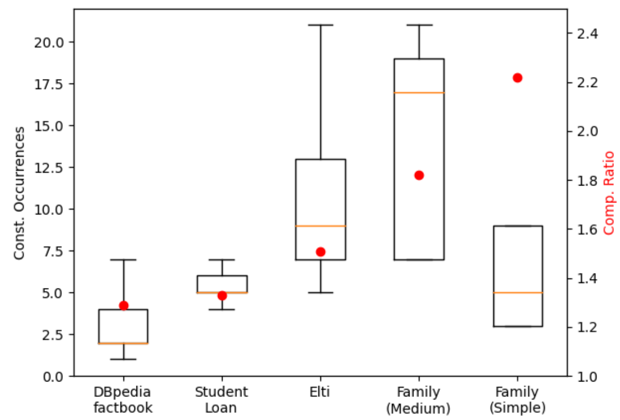

Figure 3: Distribution of the Number of Constant Occurences

$$sister(X1, X0) \leftarrow sister(X0, X1), daughter(X1, ?)$$

However under $\delta(r)$, only 7 predicates are covered by the following rule:

$$sister(X0, X1) \leftarrow mother(X0, ?), sister(X1, X0)$$

5.2 Sign of Promising Compression

To find indicators that imply large compression ratio, we observe three indicators related to constants in RKBs:

Constant Occurrences This is the number of total occurrences of each constant symbol in the given RKB.

Constant Locations This is the number of different locations (i.e. different arguments in predicates) where each constant symbol occurs in the given RKB.

Predicate Argument Similarity All constants occur in the same argument of predicates form some multisets. Predicate argument similarities are Jaccard similarities of pairs of these multisets.

Figure 3, 4 and 5 show distributions of the above indicators in each dataset respectively together with compression ratios on the corresponding datasets. These figures clearly

Table 4: Compression Result under Different Metrics

Metric	$\delta(r)$		$\tau(r)$	
Dataset	**Comp. Ratio**	**Time(s)**	**Comp. Ratio**	**Time(s)**
Elti	1.51	5.12	1.92	3.10
StudentLoan	1.33	2450.31	1.34	6707.10
DBpedia-factbook	1.29	0.36	1.29	0.85
Family(Simple)	2.22	0.78	2.22	1.30
Family(Medium)	1.82	24.39	1.60	29.60

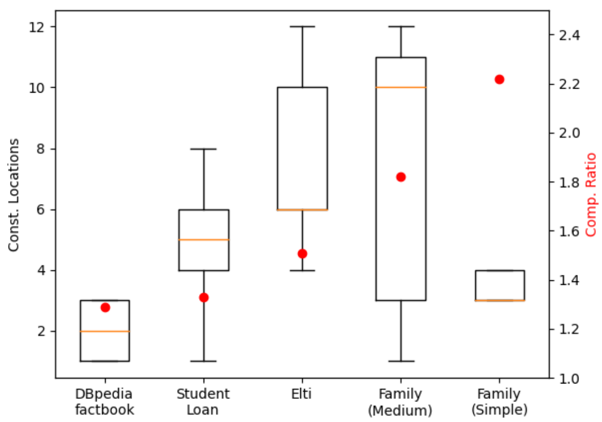

Figure 4: Distribution of the Number of Constant Locations

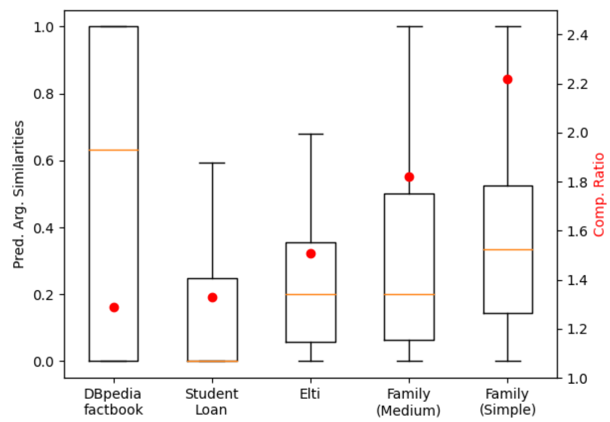

Figure 5: Distribution of Predicate Argument Similarities

show strong relevance between these indicators and compression ratios. More repetitions and locations of most constant symbols as well as larger similarities between most argument pairs imply larger compression ratios.

More repetitions make constant symbols more likely to be under constant equivalent conditions. For exmaple, $male$ and $female$ are two constants that present most repetitions in “Family(Simple)” and “Family(Medium)”. Both of these two constants contribute to equivalent conditions in some rules, such as:

$$gender(X0, female) \leftarrow mother(X0, ?)$$

$$gender(X0, male) \leftarrow brother(X0, ?)$$

More locations of constant symbols provides possibilities for variables to link different predicates or different arguments within the same predicate in a certain rule. Further more, from the aspect of predicates, higher argument similarities suggest that two arguments of predicates are more likely to be bounded by the same variable. And if some argument is grounded by most constants, it is more likely to be a free variable in axiomatic rules. For instance, most constants in “Elti” dataset appears in the only argument of $person$. And the following rule is induced as an axiom with no counter examples:

$$person(X0) \leftarrow True$$

In “StudentLoan”, the definition for $person$ is restricted by another predicate that describes some properties of persons. For example, first argument of $no\_payment\_due$ is the most similar with the first argument of $person$ and $no\_payment\_due$ helps to void most irrelevant constants to be defined as persons:

$$person(X0) \leftarrow no\_payment\_due(X0, ?)$$

“DBpedia-factbook” and “Family(Simple)” does not comply with some of the above phenomena because neither of the number of constants nor the number of predicates is large enough to show statistical significance.

5.3 Impact of Errors in RKBs

In this experiment we perturb “Family(Simple)” dataset by randomly removing or altering predicates in original KB according to some error rates. And the compression procedure is tested on the perturbed KB. The results are shown in Table 5.

The results show that compression is heavily devastated by errors in KBs and searching time increases dramatically under higher error probabilities. The increment of $|N|$ indicates that positive entailments shrink spectacularly while $|A|$ suggests that counter examples remain under control. $\#H$ keeps steady but $|H|$ increases, which shows that induced rules become longer under higher error rates.

Table 5: Compression under Different Error Rate in KB

Error Rate	$\|B\|$	$\|N\|$	$\|A\|$	$\#H$	$\|H\|$	Comp. Ratio	Time(s)
0	240	100	0	4	8	2.22	0.73
0.05	240	108	11	4	8	1.89	1.74
0.1	236	185	12	2	4	1.17	1.64
0.15	237	203	3	2	7	1.11	7.29
0.2	233	181	12	2	6	1.17	16.81
0.25	229	150	32	3	9	1.20	24.18
0.3	221	177	13	4	13	1.09	40.25
0.35	227	195	7	3	12	1.06	424.29

Errors in KBs destroy original regularities mainly by decreasing positive entailments. To be complied with chaotic semantics, rules tend to be much longer for more precise conditions to exclude enough counter examples. This results in longer searching paths before convergence and the number of attempts increases exponentially. When no error occurs, the compression procedure converges before searching for rules longer than 4. But when more than one third predicates are errors in the KB, the procedure does not stop even when candidate rule length is longer than 9.

6 Conclusion

In this paper, we have reported on our challenge to a hard problem of relational compression with semantic approaches on KBs. As far as we know, we are the first to inductively reduce relations for the compression. To accurately capture the substance, the problem should be carefully formalised and so we did in the first step inlcuding formal definition and complexity analysis; finding essential knowledge, especially unprovable predicates, founded our fundamental solution, in which we also proposed an algorithm to find the rules co-working with essential knowledge to achieve the satisfying compression effect. Our technique has been evaluated with real life RKBs, and the results practically testify our expectations on the technique.

Relational compression has been proved in this paper to be NP-hard. Although we have given a competitive algorithm, there is possibly room for further improvement. Moreover, in the future our technique is hopefully a standard tool for KBs such that overall performance improvement is also valuable.

References

Blockeel, H., and De Raedt, L. 1998. Top-down induction of first-order logical decision trees. *Artificial intelligence* 101(1-2):285–297.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems (NIPS)*, 1–9.

Boros, E.; Čepek, O.; Kogan, A.; and Kučera, P. 2009. A subclass of horn cnfs optimally compressible in polynomial time. *Annals of Mathematics and Artificial Intelligence* 57(3):249–291.

Boyar, J.; Matthews, P.; and Peralta, R. 2013. Logic minimization techniques with applications to cryptology. *Journal of Cryptology* 26(2):280–312.

Califf, M. E., and Mooney, R. J. 2003. Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research* 4(Jun):177–210.

Chen, J.; Liu, Y.; Lu, S.; O'sullivan, B.; and Razgon, I. 2008. A fixed-parameter algorithm for the directed feedback vertex set problem. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, 177–186.

Codd, E. F. 2002. A relational model of data for large shared data banks. In *Software pioneers*. Springer. 263–294.

Cohen, W. W. 1995a. Fast effective rule induction. In *Machine learning proceedings 1995*. Elsevier. 115–123.

Cohen, W. W. 1995b. Text categorization and relational learning. In *Machine Learning Proceedings 1995*. Elsevier. 124–132.

Domingos, P. 1996. Unifying instance-based and rule-based induction. *Machine Learning* 24(2):141–168.

Evans, R., and Grefenstette, E. 2018. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research* 61:1–64.

Galárraga, L. A.; Teflioudi, C.; Hose, K.; and Suchanek, F. 2013. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*, 413–422.

Hammer, P. L., and Kogan, A. 1995a. Quasi-acyclic propositional horn knowledge bases: Optimal compression. *IEEE Transactions on knowledge and data engineering* 7(5):751–762.

Hammer, P. L., and Kogan, A. 1995b. Quasi-acyclic propositional horn knowledge bases: Optimal compression. *IEEE Transactions on knowledge and data engineering* 7(5):751–762.

Hanschke, P., and Würtz, J. 1993. Satisfiability of the smallest binary program. *Information Processing Letters* 45(5):237–241.

Holder, L. B.; Cook, D. J.; Djoko, S.; et al. 1994. Substucture discovery in the subdue system. In *KDD workshop*, 169–180. Washington, DC, USA.

Kersting, K., and De Raedt, L. 2001. Bayesian logic programs. *arXiv preprint cs/0111058*.

Kietz, J.-U., and Džeroski, S. 1994. Inductive logic programming and learnability. *ACM SIGART Bulletin* 5(1):22–32.

Marcinkowski, J., and Pacholski, L. 1992. Undecidability of the horn-clause implication problem. In *FOCS*, 354–362.

Mooney, R. 1999. Relational learning of pattern-match rules for information extraction. In *Proceedings of the sixteenth national conference on artificial intelligence*, volume 334.

Muggleton, S. 1991. Inductive logic programming. *New generation computing* 8(4):295–318.

Muggleton, S. 1995. Inverse entailment and progol. *New generation computing* 13(3-4):245–286.

Müller, M. E. 2012. *Relational Knowledge Discovery*. Cambridge University Press.

Niblett, T. 1988. A study of generalisation in logic programs. In *Proceedings of the 3rd European Conference on European Working Session on Learning*, 131–138.

Nijssen, S., and Kok, J. N. 2004. A quickstart in frequent structure mining can make a difference. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 647–652.

Pazzani, M., and Kibler, D. 1992. The utility of knowledge in inductive learning. *Machine learning* 9(1):57–94.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine learning* 5(3):239–266.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine learning* 62(1-2):107–136.

Sachan, M. 2020. Knowledge graph embedding compression. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2681–2691.

Schmidt-Schauss, M. 1988. Implication of clauses is undecidable. *Theoretical Computer Science* 59(3):287–296.

Schoenmackers, S.; Davis, J.; Etzioni, O.; and Weld, D. 2010. Learning first-order horn clauses from web text. In *Proceedings of the 2010 Conference on Empirical Methods on Natural Language Processing*, 1088–1098.

Wang, J.; Bao, W.; Sun, L.; Zhu, X.; Cao, B.; and Philip, S. Y. 2019. Private model compression via knowledge distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1190–1197.

Yim, J.; Joo, D.; Bae, J.; and Kim, J. 2017. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4133–4141.

Yin, X.; Han, J.; Yang, J.; and Philip, S. Y. 2006. Crossmine: Efficient classification across multiple database relations. In *Constraint-Based mining and inductive databases*. Springer. 172–195.

Zeng, Q.; Patel, J. M.; and Page, D. 2014. Quickfoil: Scalable inductive logic programming. *Proceedings of the VLDB Endowment* 8(3):197–208.