Theory Revision on ILP systems

Lokhin Chan
lhc18@ic.ac.uk

October 2020

1 Abstract

Inductive Logic Programming (ILP) aims to learn a program by producing rules, called hypothesis given some pre-existing background knowledge[4]. This article aims to suggest a new tool, Revision on Inductive Logic Programming (RoILP), which could refine hypothesis that are pre-defined using state-of-the-art ILP systems such as FastLAS[2] and ILASP[3].

2 Introduction

Revision on Inductive Logic Programming (RoILP) is a system that could revise an existing clauses by breaking the possibly incorrect clauses into meta-rules. Then these rules are solved under existing ILP solving systems such as FastLAS or ILASP, in which generalised result are produced. These results are then interpreted and transform the revisable clauses such that these clauses covers the positive examples but not the negative ones.

There are tools that also perform theory revision on ILP such as RASPAL[1]. However, this tool is considered as more expressive since it supports a more flexible revision such as revision on head literals or learning a new constraint.

2.1 Types of rules that can be revised

The revisable rules can be expressed in one of the following forms:

- As a normal rule form $\texttt{h:-b}_1\texttt{,b}_2\texttt{,...,b}_\texttt{n}$, where `h` is atom and `b1, b2,..,bn` are atoms or negation of atoms.
- As a constraint form $\texttt{:-b}_1\texttt{,b}_2\texttt{,...,b}_\texttt{n}$.
- As a disjunctive head form $\texttt{h}_1\texttt{,h}_2\texttt{,...,h}_\texttt{m}\texttt{:-b}_1\texttt{,b}_2\texttt{,...,b}_\texttt{n}$.

There are also two modes of revision in the tool:

- Body-only revision: In this mode of revision, only body literals are added or deleted.
- Full revision: In this mode of revision, head literals and body literals can both be added or deleted, and one of the three forms mentioned above might be generated, depending on the examples in the ILP program.

3 How to run the tool

There are multiple of examples that are included in the source file, as well as in this article. The revision task can be invoked with the following command:

```
python revision.py [--solver SOLVER] [--revise-type REVISE_TYPE]
[--max-head MAX_HEAD] [--nc] task_path
```

The meaning and the function of each flags are as explained as follows:

- –solver refers to the ILP task solver that is used. SOLVER would be either "ILASP" or "FastLAS". By default solver would be "ILASP" if such flag is not given.
- –revise-type refers to the types of revision discussed from the previous section. REVISE_TYPE would be either "body" or "full". By default revise-type would be body. Note that body revision mode can work with both FastLAS and ILASP, but full revision can only work with ILASP.
- –max-head refers to the maximum number of head atom that could appear on any single revised rule. Note that works only with full revision mode as max-head refers to head modification. By default maximum number of head atoms are not restricted.
- –nc refers to no-constraint. It only works with full revision mode, and when the flag is enabled, no constraint rule can be learnt. By default full revision is possible to remove all head atoms and generate revised constraint rule.

3.1 Reserved Keywords

Here are a list of keywords that are reserved that should not be defined in the input task file, else there might be unexpected behaviour:

3.1.1 Constants

```
revid_type   id_type   hid_type   head_extension_id
```

3.1.2 predicates

`delete/2`	`extension/2`	`head_delete/2`	`head_extension/3`
`v`	`prove/1`	`possible_head/3`	`try/3`
`empty_head/1`	`is_head/2`		

These constants and predicates mentioned above, with the reserved word in the specified solver (ILASP/FastLAS), constitutes the whole set of reserved words.

4 Meta-representation

In this section, the mapping from actual revisable theory to meta-representation will be discussed. For the implementation of the meta-rules, please read section 5 Algorithm.

The input revisable theory, as well as possible head of these theories will be broken down into meta-data and the task will be solved by ILP solvers. The transformation of input data to rules and meta-data can be separated into two parts: body and head modification.

4.1 Body meta-rules

Each revisable theory and literals on the body would be translated to a form of $\texttt{try(revid}_\texttt{i}\texttt{,id}_\texttt{ij}\texttt{,b}_\texttt{ij}\texttt{)}$, where i is the ith revisable theory, and j is the jth literal on the body. Furthermore, each `try/3` predicates can be derived by either $\texttt{try(revid}_\texttt{i}\texttt{,id}_\texttt{ij}\texttt{,b}_\texttt{ij}\texttt{) :- b}_\texttt{ij}\texttt{,delete(revid}_\texttt{i}\texttt{,id}_\texttt{ij}\texttt{),grounding\_atoms}$ or $\texttt{try(revid}_\texttt{i}\texttt{,id}_\texttt{ij}\texttt{,b}_\texttt{ij}\texttt{) :- not delete(revid}_\texttt{i}\texttt{,id}_\texttt{ij}\texttt{),grounding\_atoms}$. If the result from ILP solver contains `delete/2`, then the corresponding body literal will be deleted as revised theory, else the body literal will be retained.

Apart from `try` clauses, a `extension/2` will also be formed for each revisable theory. The `extension` clause would then interact with head declaration and will be solved in ILP solver as part of hypothesis space. The solver result will have a form of $\texttt{extension(revid}_\texttt{i}\texttt{, variables\_in\_theory) :- predicate}_\texttt{1}\texttt{, predicate}_\texttt{2}\texttt{,...,predicate}_\texttt{n}$. Then each of the `predicate` would be appended to the body of that revisable theory. The construction of these predicates are an adaption from RASPAL approach.[1]

Predicates `tried/1` and `extended/1` are included to ensure that there are one try clause and one extend clause that derived.

4.2 Head meta-rules

For each head and possible head defined in the input task file, they are translated into `head_delete/2` and `head_extension/3` respectively. The purpose of these two predicates are to decide which head literals should be included in the head of the revised theory. Similar to `delete/2` and `extension/3` discussed above, if `head_delete(revid`$_i$`,id`$_{ij}$`)` is derived, where i is the ith revisable theory and j is jth head literal, then head literals with id^{ij} would be deleted, whereas if `head_extension(revid`$_i$`, id`$_i$`, literal)` is derived, then literal will be included in the head.

For each head, `possible_head(revid`$_i$`, variables_in_theory, literal)` is derived when `head_delete/2` does not hold. For each possible head literals, on the other hand, `possible_head/3` is derived if `head_revision/3` holds.

A special predicate `possible_head(revid`$_i$`, variables_in_theory, null)` indicates a empty head for a particular instance of variables. There are two ways that the above predicate could be derived: Either all `head_delete/2` and none of the `head_extension/3` are derived, then the predicate would be derived for all instances; or there exist at least one head literal after revision, but for some instances heads could not be derived from body, in which the corresponding `possible_head/3` would not hold.

`prove(literal)` is a predicate that indicates literal is being proved or derived in the ASP program, and `prove/1` interacts with `possible_head/3` as a conditional rule.

5 Algorithm

The algorithm of the revision tool can be roughly seperated into three parts:

- Pre-processing: Break down the revisable theory into meta-data and generate rules with a form of ILP program such that it could be solved by ILP solver.
- Solving: The program is solved by ILP solver.
- Post processing: Revise on the theory according to the result given by ILP solver.

Since the solving process is delegated to ILP solver, this section would mainly focus on pseudocode on pre-processing and post-processing.

Algorithm 1: PRE-PROCESSING

```
Input   : Revisable Theory R, Possible Heads PH, workfile W
Output: Workfile to be solved by ILP solver $\tilde{W}$
$\tilde{W} = W \cup \{\#modeh(delete(revid\_type, id\_type).\}$
      $\cup\{\#bias(\text{"} :- head(delete(\_, \_)), body(\_).\text{"}).\}$
if $revise\_mode==\text{"}full\text{"}$ then
    $\tilde{W} = \tilde{W} \cup \{\#modeh(head\_delete(revid\_type, hid\_type)).\}$
    $\tilde{W} = \tilde{W} \cup \{\#bias(:- head(head\_delete(\_, \_)), body(\_).\text{"})$
          $\cup\{\#bias(:- head(head\_extension(\_, \_, \_)), body(\_).\text{"})$
    $\tilde{W} = \tilde{W} \cup \{prove(X) : possible\_head(R, V, X) :- extension(R, V).\}$
end
foreach $h_{i1}; h_{i2}, ..., h_{im} :- b_{i1}, b_{i2}, ..., b_{in} \in R$ do
    $v_i$ = set of variables that appears in the theory
    $v'_i$ = wrapped predicate for $v_i$
    $\tilde{W} = \tilde{W} \cup \{extended(i) :- extension(i, v'_i), v_i.\} \cup \{:- not\ extended(i).\}$
    $\tilde{W} = \tilde{W} \cup \{\#constant(revid\_type, i)\} \cup \{\#modeh(extension(i, v'_i)).\}$
    if $revise\_mode==\text{"}body\text{"}$ then
        $\tilde{W} = \tilde{W} \cup \{h_{i1}; h_{i2}, ..., h_{im} :- try(i, id(b_{i1}), b_{i1}), ...,$
              $try(i, id(b_{in}), b_{in}), extension(i, v'_i).\}$
    else
        $\tilde{W} = \tilde{W} \cup \{possible\_head(i, v'_i, null) :- empty\_head(i), v_i.\}$
        $\tilde{W} = \tilde{W} \cup \{possible\_head(i, v'_i, null) :- not\ empty\_head(i)$
              $not\ possible\_head(i, v_i, ph_{i1}), .., not\ possible\_head(i, v_i, ph_{ik}),$
              $v_i.\}$
        $\tilde{W} = \tilde{W} \cup \{empty\_head(i) :- head\_delete(i, id(h_{i1})), ..,$
              $head\_delete(i, id(h_{ik})), not\ head\_extension(i, id(ph_{i1}), ph_{i1}),$
              $.., not\ head\_extension(i, id(ph_{il}), ph_{il}), v_i.\}$
        $\tilde{W} = \tilde{W} \cup \{:- empty\_head(i), try(i, id(b_{i1}), b_{i1}), ...,$
              $try(i, id(b_{in}), b_{in}), extension(i, v'_i).\}$
        $\tilde{W} = \tilde{W} \cup \{:- not\ empty\_head(i), possible\_head(i, v'_i, null) : v_i.\}$
        foreach $h_{ij}, j \in \{1..m\}$ do
            $\tilde{W} = \tilde{W} \cup \{\#constant(hid\_type, id(h_{ij}).\}$
            $\tilde{W} =$
             $\tilde{W} \cup \{possible\_head(i, v'_i, h_{ij}) :- not\ head\_delete(i, id(h_{ij})),$
                  $try(i, id(b_{i1}), b_{i1}), .., try(i, id(b_{in}), b_{in}), extension(i, v'_i).\}$
        end
        foreach $ph_{ij} \in PH, j \in \{1..k\}$ do
            $ph'_{ij}$ = modeh abstraction of $ph_{ij}$
            $\tilde{W} = \tilde{W} \cup \{\#modeh(head\_extension(i, h\_extension\_id, ph'_{ij}).\}$
            $\tilde{W} = \tilde{W} \cup \{\#constant(h\_extension\_id, id(ph_{ij}).\}$
            $\tilde{W} = \tilde{W} \cup \{ph_{ij} :- prove(ph_{ij}).\}$
            $\tilde{W} = \tilde{W} \cup$
             $\{possible\_head(i, v'_i, ph_{ij}) :- head\_extension(i, id(ph_{ij}), ph_{ij}),$
                  $try(i, id(b_{i1}), b_{i1}), .., try(i, id(b_{in}), b_{in}), extension(i, v'_i).\}$
        end
    end
    foreach $b_{ij}, j \in \{1..n\}$ do
        $\tilde{W} = \tilde{W} \cup \{\#constant(id\_type, id(b_{ij})).\}$
        $\tilde{W} = \tilde{W} \cup \{try(i, id(b_{ij}), b_{ij}) :- delete(i, id(b_{ij})), v_i.\}$
              $\cup\{try(i, id(b_{ij}), b_{ij}) :- b_{ij}, not\ delete(i, id(b_{ij})), v_i.\}$
        $\tilde{W} = \tilde{W} \cup \{tried(i, b_{ij}) :- try(i, id(b_{ij}), b_{ij}), v_i.\}$
              $\cup\{not\ tried(i, id(b_{ij}).\}$
    end
end
return $\tilde{W}$
```

Algorithm 2: POST-PROCESSING

```
Input  : Reviable theory R, Extension result E, Deletion result D,
         Head Extension result HE, Head Deletion Result HD
Output: Revised theory $\tilde{R}$
$r_i$ = revisable theory with revision id $i$
foreach $extension(i, v(X_1, ..X_n)) :- b'_1, .., b'_m \in E$ do
    foreach $b'_j(Y_{j1}, .., Y_{jk}\}, j \in \{1..m\}$ do
        appends $b'_j$ at the body of $r_i$
    end
end
foreach $delete(i, id(b) \in D$ do
    deletes body literal with $id(b)$ in $r_i$
end
foreach $head\_delete(i, id(h)) \in HD$ do
    deletes head literal with $id(h)$ in $r_i$
end
foreach $head\_extension(i, id(ph_i), ph'_i(X_1, .., X_n)) \in HE$ do
    $ph_i(Y_1, .., Y_n)$ = head literal with $id(ph_i)$
    appends $ph_i(Y_1, .., Y_n)$ in $r_i$
    if $X_i == X_j \wedge Y_i\,! = Y_j$ then
        append $Y_i = Y_j$ at the body of $r_i$
    end
end
return $\tilde{R}$
```

6 Examples

Below are two examples where one revise on body mode, where the another one revises on a full mode.

Example 1 Input file provided:

```
%Mode Declaration
#modeb(c2(var(atom))).
#modeb(c3(var(atom))).

%Background theory
atom(a).
atom(b).

c2(X) :- c3(X).
c3(b).
c2(a).
```

```
%Examples
#pos(eg1, {p(a)}, {p(b), q(b)}, {}).
#revisable(rev1, (p(X); q(X):- c2(X), c3(X).), (X: atom)).
```

Solving it with ILASP yields the following intermediate result:

```
delete(rev1,c3_X).
extension(rev1,v(V1)) :- not c3(V1); atom(V1).
```

Therefore the revised theory would be:

```
p(X); q(X) :- c2(X), not c3(X).
```

Example 2 Input file provided:

```
%Mode Declaration
#modeb(c2(var(atom))).
#modeb(c3(var(atom))).

%Background theory
atom(a).
atom(b).

%c2(X) :- c3(X).
c3(b).
c2(a).

%Examples
#pos(eg1, {q(a)}, {}, {}).
#pos(eg2, {r(a)}, {}, {}).
#neg(eg4, {q(b)}, {}, {}).
#neg(eg3, {p(a)}, {}, {}).

#revisable(rev1, (p(X) :- c2(X).), (X: atom, Y:atom)).
#modeh(q(var(atom))).
#modeh(r(var(atom))).

% Other ways to define possible head.
%#possible_head(rev1, {p(X), q(X)}).
%#possible_head_modeh(rev1, {q(var(atom)), r(var(atom))}).
```

Solving it with ILASP yields the following intermediate result:

```
head_delete(rev1,p_X).
head_extension(rev1,r_Y,r(V1)) :- atom(V1).
head_extension(rev1,q_Y,q(V1)) :- atom(V1).
extension(rev1,v(V1,V1)) :- atom(V1).
```

Therefore the revised theory would be:

```
r(Y); q(Y) :- c2(X), atom(Y), X = Y.
```

7 Future Work

There are two aspects that the current tool can be improved:

- Enable more ILP solver: Currently the tool only supports ILASP with body or full revision, whereas supports FastLAS with only body revision. By extending full revision with FastLAS, or supporting revision with other similar ILP solver would be a possible extension in the future.
- Supports more output features for revision: The currently revision only revise on body with atoms and negation of atoms, also revise on head with constraints or disjunction. It is possible to extend the revision solver such that it supports more feature, for example modifying choice rules on the head, revise on in-build comparison functions or arithmetic functions on the body.

8 Conclusion

This article shows the methods and algorithms on how a theory can be revised with using ILP solver tools such as ILASP and FastLAS. This article extends and adapts the method that is shown in [1]. It is believed that revision could be applied onto a variety of real world problems by learning a correct theory under the context of ILP system.

References

[1] D. Athakravi, D. Corapi, K. Broda, and A. Russo, "Learning through hypothesis refinement using answer set programming," *International Conference on Inductive Logic Programming.*

[2] M. Law, A. Russo, E. Bertino, K. Broda, and J. Lobo, "Fastlas: Scalable inductive logic programming incorporating domain-specific optimisation criteria," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 2877–2885, 2020.

[3] M. Law, A. Russo, and K. Broda, "The ilasp system for inductive learning of answer set programs," 2020.

[4] S. Muggleton, "Inductive logic programming," *New Generation Computing*, vol. 8, pp. 295–318, 1991.