Inductive Logic Programming in Answer Set Programming

Domenico Corapi, Alessandra Russo, and Emil Lupu

Department of Computing,
Imperial College London,
Exhibition Road London, SW7 2AZ, United Kingdom,
{d.corapi, a.russo, e.c.lupu}@imperial.ac.uk

Abstract. In this paper we discuss the design of an Inductive Logic Programming system in Answer Set Programming and more in general the problem of integrating the two. We show how to formalise the learning problem as an ASP program and provide details on how the optimisation features of modern solvers can be adapted to derive preferred hypotheses.

1 Introduction

Over the last years we have witnessed increasing interest in *nonmonotonic logic programming*, as an alternative to systems based on classical logic. Nonmonotonicity is considered essential for systems based on common-sense reasoning. Answer Set Programming (ASP) is a recent approach to nonmonotonic logic programs that builds on the stable model semantics. The success of ASP is due in part to the recent availability of efficient and powerful inference engines and solvers. Despite the recent growth of ASP, little has been done to investigate the potential contribution to the field of Inductive Logic Programming (ILP).

Besides the challenges posed by nonmonotonicity, ILP systems are not declarative enough. Users are sometimes required to experiment with the ordering of the clauses, this being relevant not only to efficiency but even to termination or correctness. Also, whenever the task is particularly knowledge-intensive, ILP systems usually tend to perform redundant computations and a high share of the total computation time is taken by particularly heavy inferences. This is also the case of those ILP applications in which a large portion or all of the admissible solutions are required. ASP provides a natural solution to computational problems that are more adequately solved by SAT-based techniques rather than resolution. We also believe the rate of improvements of tools and solver over the last years makes ASP a very promising area for the field of ILP.

In this paper we present an ILP system, called ASPAL, that relies on the mapping of the mode declarations into partially instantiated rules. An ASP solver is used to find an optimal or all the possible hypotheses.

2 The hypothesis space

We assume the reader is familiar with logic programming [3], and ILP [4]. In particular in this paper we approach the problem of nonmonotonic ILP [6], [1].

We recall the definition of mode declaration from [4] as it plays a key role in this paper.

Definition 1. *Let m be a mode declaration and l a literal. l is* compatible *with m iff l corresponds to the schema of m where all the input and output placemarkers are replaced with variables and constant placemarkers are replaced with constants. Let r be a rule $h \leftarrow b_1, ..., b_n$ with a specific total order on the literals (*ordered rule*) in the body and M be a set of mode declarations. r is* compatible *with M on $(m_h, m_{b_1}, ..., m_{b_n})$ iff the following conditions are met: (i) $m_h = modeh(s) \in M$ and h is compatible with m_h (ii) for each $b_i, i = 1, ..., n$, $m_{b_i} = modeb(s) \in M$ and b_i is compatible with m_{b_i}; (iii) every input variable in any of the literals is either an input variable in h or an output variable in some literal $b_j, j < i$. The set of all compatible rules is denoted ${\mathcal{R}^o}_M$*

2.1 Hypothesis complexity

In order to derive an effective strategy we want to study how the given inputs affect the number of candidate solutions defined by a set of mode declarations. We use $|out(m)|$, $|inp(m)|$, $|con(m)|$ to denote respectively the number of output, input and constant placemarkers, in a mode declaration m. The space of the admissible hypotheses depends, amongst other factors, on the number of constants that can be instantiated. We want to abstract from it and reason on an intermediate case.

Definition 2. *A* skeleton rule *for a given set of mode declarations M is a compatible rule where all the constants placemarkers are replaced with different variables instead of constants. A* skeleton hypothesis *is a set of skeleton rules.*

We want to analyse the space of possible skeleton hypotheses for a given language bias M. The number of possible heads N_h is the number of head mode declarations, since each placemarker is substituted by a different variable with no degree of freedom. The number of possible conditions depends on the number of variables that can be bound to input variables (namely the set of variables that appear as output variables in preceding conditions or as input variables in the head), denoted as v in the following:

$$N_c(v) = \sum_{m \in M_b} v^{|inp(m)|} \leq |M_b| v^{max\{|inp(m)|:m \in M_b\}}$$

We define a limit on the number of conditions MNC. To simplify the notation we denote $max\{|inp(m)| : m \in M_b\} = max_i$ and $max\{|out(m)| : m \in M_b\} = max_o$. An upper bound for the number of variables given as input to a condition is the maximum number of outputs in a conditions multiplied by the maximum number of conditions: $max_o * MNC$. An upper bound for the number of rules can be derived considering for each place in the body all possible conditions:

$$N_r \leq |M_h| * (|M_b| * (max_o * MNC)^{max_i})^{MNC}$$

This upper bound confirms the intuitive idea that the number of input placemarkers and the maximum number of conditions are the factors that affect the most the hypothesis complexity. ASPAL iterates on the maximum number of conditions allowed in a rule in order to avoid the generation of superfluous skeleton rules, control the computation time and avoid costly groundings.

3 Abduction in ASP

The approach presented here is based on the mapping of the ILP problem into an abductive problem where language bias is "flattened" into logic atoms (similarly to [1]). In most of ILP problems we are not interested in all solutions but we want to find optimal solutions, according to some notion of optimality. In order to achieve this, we use *abduction with penalisation* [5].

Definition 3. *An* abduction with penalisation task *is a tuple* $\langle T, E, A, \gamma \rangle$*, where* T *is a normal logic program called* background knowledge*,* E *is a conjunction of literals called* examples*,* A *is a set of atoms called* abducibles *and* γ *is a function from* A *to the non-negative reals called* penalty function*. A set of hypotheses* $\Delta \subseteq A$ *is called* (abductive) hypothesis *if there exists a stable model* M *of* $T \cup \Delta$ *such that* E *is true in* M*.* $\Gamma(\Delta) = \sum_{a \in \Delta} \gamma(a)$ *is called the* penalisation *of* Δ*.*

Modern ASP solver support aggregates [2]. In particular, we can explicitly state the set of abducibles $A = \{a_1, ..., a_n\}$ as follows in *clingo* [2], the solver used in ASPAL:

$$0\ \{a_1, ..., a_n\}\ max\_abducibles.$$

where *max_abducibles* defines the maximum number of elements in A that can be true in an answer set. Note that the abducibles can be stated intensionally by using variables and defining the type of variables. The penalisation function is supported by an *optimisation statement*:

$$minimize[a_1 = \gamma(a_1), ..., a_n = \gamma(a_n)]$$

The solver will find an answer set S and an abductive solution $\Delta \subseteq S$ such that $\Gamma(\Delta)$ is minimum.

4 Inductive learning in ASP

The learning process starts by generating all the possible skeleton rules. Given the skeleton rules and a set of abducibles associated to them, the burden of the search for the final hypotheses is shifted onto the ASP solver.

The overall procedure is described in Algorithm 1. A first phase derives from the given mode declarations the skeleton rules with an additional abducible in the body that identifies the rule and contains the constants that appear in it.

The outer loop iterates on the number of maximum conditions allowed in the rule. The loop terminates when a condition is met, e.g. when a satisfactory

Algorithm 1 FIND-HYPOTHESIS

Inputs: E examples; B background theory; M mode declarations; γ penalisation function

Outputs: H hypotheses

$MNC = 0$
$H = \emptyset$
while $\langle termination\_condition \rangle$ **do**
  $Q, A =$ DERIVE-SKELETON-RULES(M, MNC)
  $\{\Delta_1, ..., \Delta_n\} =$ ASP-ABDUCE($Q \cup B$, E, A, γ)
  $H = H \cup$ TRANSLATE-SOLUTIONS($\{\Delta_1, ..., \Delta_n\}, M$)
  $\langle increase\ MNC \rangle$
end while

number of solutions is generated or when an optimal solution is found. Optimisation statements are used to find an optimal solution within each iteration. In the most common case where the optimisation is on the complexity of the rule we can derive lower bounds on the penalisation in order to terminate the computation of an optimal answer[1].

The generation of skeleton rules is the core part of the algorithm and it is driven by mode declarations. A direct generation from mode declarations is not possible since the definition introduces redundancies in the hypotheses space, due to the use of ordered rules. For example, consider the ordered (skeleton) rule $p(X) \leftarrow q(X), r(X)$. In the construction of skeleton rules we need to be careful not to construct also the logically equivalent rule $p(X) \leftarrow r(X), q(X)$. Redundancies like these have the effect of producing multiple answers for the same inductive solution with no benefit on the search. We could be tempted to define a subset of semantically equivalent compatible ordered rule just by commuting conditions. Unfortunately only some of the resulting rules would still be compatible because of condition (iii) in Definition 1, that imposes an ordering over conditions that share variables. To overcome this, we define a subset of $\mathcal{R}^o$ (we omit the implicit M in the following), whose elements satisfy a given total order.

Definition 4. *Let f_1 and f_2 be two conditions within an ordered rule. We define a full order as follows: $f_1 \leq f_2$ iff (i) There is an input variable in f_2 that is bound to an output variable in f_1 or (ii) Condition (i) is false and $f_1 \leq_l f_2$. The ordering $\leq_l$ is an arbitrary total order on $\mathcal{R}^o$.*

We call the set of rules that respect the above order $\mathcal{R}^r \subseteq \mathcal{R}^o$.

Theorem 1. *Each element r in $\mathcal{R}^r$ characterises an equivalence class $[r] = \{r' \in \mathcal{R}^o : r' \sim r\}$.*

[1] For example, if a solution with complexity 2 has been found in an iteration where $MNC = 1$, we know that the iteration with $MNC = 2$ will not find a better solution since the new solutions explored will have at least complexity 3 (the head plus the two conditions).

The mapping $\mathcal{R}^o \mapsto \mathcal{R}^r$ is a canonical projection from an ordered rule to an equivalent (non-ordered) rule. We can use the new set to remove the redundancy caused by ordered rules without loss of completeness and soundness.

In order to derive a concrete implementation we need to define the arbitrary total order $\leq_l$ in Definition 4. In ASPAL we further refine $\leq_l$ to be a lexicographic order on an internal mode-based representation. The skeleton rules are constructed using a dynamic programming algorithm based on the extension of the set of skeleton rules. This is achieved by adding the head first, then adding all possible sequences of producers and then adding the consumers. When deciding whether to add or not a condition, the order defined in Definition 4 is used. The outcome of this process is a set of rules of the following type:

```
head :-
    ..., conditions, ...
    rule(<internal representation>)
```

where *rule* is an abducible containing a list of constants used in the rule. Whenever a certain *rule* atom is abduced an instantiation of the constants is chosen and the effect on the semantics of the final theory augmented with the abducible is the same as adding the skeleton rule with the specified instantiations of the constants. This is made clear in the next section with an example.

5 Example

Consider the following ILP problem:

$$B = \begin{cases} bird(a). \ \ bird(b). \\ can(a, fly). \ \ can(b, swim). \\ ability(fly). \\ ability(swim). \end{cases} \qquad M = \begin{cases} modeh(penguin(+bird)). \\ modeb(notcan(+bird, \#ability)). \end{cases}$$

$$E = \begin{cases} penguin(b). \\ not\ penguin(a). \end{cases}$$

We consider as penalisation function the number of literals in a hypothesis. The following skeleton rules Q are derived for $MNC = 2$[2]:

```
penguin(A):-
    rule(r1).
penguin(A):-
    A=B, not can(B,C),
    rule(r2,c(C)).
```

```
penguin(A):-
    A=B, not can(B,C),
    A=D, not can(D,E),
    rule(r3,c(C,E)).
```

The abducibles $A = \{rule(r1), rule(r2, c(C)), rule(r3, c(C, E))\}$ contain an identifier for the rule, and a list of constants that must be instantiated in a final solution. The following aggregates and optimisation statements are also derived:

[2] Type conditions are not shown for brevity. Integrity constraints, not shown here are used to enforce the order over conditions with same structure but different constants. The encoding for the abducibles (here using rule ids $r1, r2, r3$ for simplicity) uses the flattening mechanism introduced in [1]

```
0 {rule(r1), rule(r2,c(C)):ability(C),
rule(r3,c(C,E):ability(C):ability(E)} 2.

minimize [rule(r1)=1, rule(r2,c(C))=2:ability(C),
rule(r3,c(C,E)=3:ability(C):ability(E)].
```

The first statement limits the hypotheses to maximum two rules. The second statement assigns the penalisation to abducibles. The background theory, together with the skeleton rules, the aggregates, optimisation statements and the integrity constraint $\leftarrow penguin(b), not\ penguin(a)$ derived from the examples are provided as input to the ASP solver. The final solution can be translated back into a rule. For example, since the final theory has an answer set containing $rule(r2, c(fly))$ then, as supported by completeness and soundness results not shown here, $penguin(A) \leftarrow not\ can(A, fly)$ is a hypothesis for the original ILP task.

6 Conclusion

We presented an integrated approach to solve ILP problems in Answer Set Programming. The approach is based in a preliminary construction of so called skeleton rules that serve as base for final hypotheses. The number of skeleton rules grows exponentially with the number of allowed conditions, but for many significant applications, compressive rules (thus with shorter conditions) are preferred. We tackle this complexity issue by using a iterative deepening approach on the number of conditions.

Acknowledgments

This work is funded by the UK EPSRC (EP/F023294/1) and supported by IBM Research as part of their OCR initiative and Research Councils UK.

References

1. D. Corapi, A. Russo, and E. Lupu. Inductive logic programming as abductive search. In *Tec. Comm. of the 26th ICLP*, volume 7 of *LIPIcs*, pages 54–63, Dagstuhl, Germany, 2010.
2. M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. T. Schneider. Potassco: The potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, 2011.
3. J. W. Lloyd. *Foundations of logic programming.* Springer-Verlag New York, Inc., New York, NY, USA, 1984.
4. S. Muggleton. Inverse entailment and progol. *New Gen. Comp.*, 13(3&4):245–286, 1995.
5. S. Perri, F. Scarcello, and N. Leone. Abductive logic programs with penalization: Semantics, complexity and implementation. *TPLP*, 5(1-2):123–159, 2005.
6. C. Sakama. Nonmonotonic inductive logic programming. In *LPNMR*, page 62, 2001.