Active learning of relational action models

Christophe Rodrigues, Pierre Gérard, Céline Rouveirol, Henry Soldano

L.I.P.N, UMR-CNRS 7030, Université Paris-Nord,
93430 Villetaneuse, France

Abstract. This paper addresses a relational reinforcement learning-like problem in which an agent learns an action model in order to be able to predict the effects of his actions. We propose here an integrated system for both action model learning and action selection. Here the agent uses the current action model to perform active learning and uses its planning abilities to have a realistic evaluation of the accuracy of the model.

1 Introduction

Adaptive behavior studies how an autonomous agent can revise its knowledge so as to adapt to an unknown environment. Such an agent needs to simultaneously learn from experience, and act so as to fulfill goals. Therefore, it needs to *integrate* learning and action selection mechanisms. When the agent's knowledge is constantly revised to take new examples into account, we call this *online* and *incremental* learning.

Adaptation within relational representations is primarily addressed by Relational Reinforcement Learning (RRL) [2] that extends Reinforcement Learning (RL) to 1st order representations. *Indirect* RL [6], in which an action model is explicitly learned, is proved to be very efficient with relational representations [1]. Given a state space $\mathcal{S}$ and an action space $\mathcal{A}$, learning an action model T consists in learning a transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. Knowing such a model T allows the agent to predict as $\hat{s'}$ the next state s' given the current state s and an action a. In the following, correct prediction such that $\hat{s'} \neq s'$ is referred to as a prediction error or a *mistake*. Learning an action model, in the *realizable* case investigated here, comes down to searching for an element in some class of action models $\mathcal{T}$ that make no prediction error.

In this paper, the agent is equipped with an implementation of IRALe [5] that starts from an empty action model and performs online learning of a conditional STRIPS model. Then, we both add an active learning mechanism to the online revision algorithm and provide the agent with planning capabilities.

2 On-line Learning of a Relational Action Model

IRALe [5] is a theory revision algorithm dedicated to action rule learning: starting from scratch, it learns the different possible effects observed after executing

an action in a given state. Only counter-examples are stored, namely examples that raised a prediction mistake at some point during the model construction.

States are described as conjunctions of ground literals. We assume that when an agent emits an action, state literals that are not affected by the action are not described in the effect part. The examples are denoted by $x.s/x.a/x.e.add, x.e.del$, with $x.s$ a conjunction of literals describing the state, $x.a$ a literal of action and an effect part with $x.e.add$ a conjunction of positive literals and $x.e.del$ a conjunction of negated literals, as in a STRIPS-like notation.

IRALe builds an action model T represented as a set of rules $T.R$ and a set $T.X$ of counter-examples that have been memorized during the agent history. Each rule r is composed by a precondition $r.p$, an action $r.a$ and an effect $r.e$, and is denoted as $r.p$ / $r.a$ / $r.e$. As opposed to [5], we restrict in this work to rules where all variables of the precondition are connected to variables in the action literal. According to a rule r, an action $r.a$ has no other effects but those described by $r.e$. For instance, a well-formed rule is the following: $on(X,Z), on(Y,W)/move(X,Y)/on(X,Y), \neg on(X,Z)$. Note that a given action may be represented by several rules.

Matching operations between rules and examples rely on OI-subsumption [3]. Note that two formulas may have several least general generalizations (*lgg*'s) under OI-subsumption, each corresponding to a largest common substructure between the input formulas.

Definition 1. *For any rule r, state s, action a and effect e,*

- r pre-matches (s,a) *($r \overset{sa}{\sim} (s,a)$) iff there exists injective substitutions σ and θ such that i) $(r.a)\sigma = a$, and ii) $(r.p)\sigma\theta \subseteq s$.*
- r post-matches (a,e) *($r \overset{sa}{\sim} (s,a)$) iff there exists an inverse substitution ρ^{-1}, and two injective substitutions σ and θ such that i) $(r.a)\rho^{-1}\sigma = a$ ii) $(r.e)\rho^{-1}\sigma\theta = e$.*
- r covers x *($r \approx x$) iff $r \overset{sa}{\sim} (x.s, x.a)$ and $r \overset{ae}{\sim} (x.a, x.e)$ for the same injective substitutions σ and θ.*
- x contradicts r *($x \nsim r$) if r pre-matches $(x.s, x.a)$ for σ and θ substitutions, and r doesn't post-match $(x.a, x.e)$ with the same substitutions.*

The model T is revised whenever the current action model fails to correctly predict the effect part of some incoming example. This happens, either because no rule pre-matches the current example (*completeness* issue) or because there are rules that pre-match this example, but do not post-match it (*coherence* issue). In both cases, the model T needs to be updated to T' in order to preserve coherence and completeness *w.r.t.* x_u and other past counter-examples previously memorized in an example memory $T.X$.

When a counter-example x_u is encountered (in which case x_u is stored in $T.X$), two kinds of modifications may be performed. cf [5] for more details. We focus here on generalization; if no rule $T.R$ pre-matches x_u, the revision algorithm (in order to preserve completeness) then searches for a generalization of a rule r of $T.R$ such that r post-matches x_u and does not contradict any example in $T.X$ (preserving coherence). If no such generalization exists, x_u becomes a

rule and is added as such to $T.R$. Note that rules are generalized by computing least general generalizations (lgg) between examples, and between rules and examples.

3 Learning and action selection integration

At any moment, the agent is in a given state s and then performs an action a that will have some effects resulting in a new state s'. In this work, we consider that the agent is in an *exploration* mode, which goal is to acquire a correct and complete action model. We study here ϵ-active exploration: with probability ϵ_a, the action to perform is chosen randomly, otherwise active exploration is performed. In a random mode, in a given state and as in MARLIE [1], any syntactically correct action can be selected and performed by the agent, and not only the *legal* ones. In the following, a *syntactically correct* action is an action instantiated with any object of the world satisfying type constraints (when available). A *legal* action has observable effects (for example, moving a clear block on another clear block), while an *illegal* action has no observable effects. Note that so-called illegal actions for a given state are numerous (such as stacking a block on a non clear block, or stacking the floor on a block).

In the active mode, the agent chooses an action that it expects to lead to a revision/generalization of the model. Hopefully, this should help increasing the ratio of the informative $state_i/action_i/state_{i+1}$ examples (*i.e.* counter-examples as previously defined) within the sequence of action/state/$state_1/action_1/state_2/$ .../$state_n$ representing the trajectory of the agent in the state space.

Intuitively, our *active exploration* strategy uses the current action model to select an action a which is not applicable, according to the current model, to the current state s but the effects of which are compatible with s. If this action is successfully applied, it will generate an example (s, a, s') that is expected to yield a generalization of one action rule for a.

In the current state s, the idea is to select all rules r such that $r.p$ does not OI-subsume s (the corresponding action is therefore not applicable to state s) and that post-match s, i.e. such that $r.e.del$, generalized with inverse substitution ρ_j^{-1}, is included in the current state s up to an injective substitution σ_j. ACTIVE-SELECT then computes, for all corresponding $\rho_j^{-1}\sigma_j$, a random least general generalization OI of preconditions of r with s (therefore $lgg_j\sigma_j\theta_j \subseteq s$).

The candidate action to apply to state s is therefore $(r.a)\rho_j^{-1}\sigma_j\theta_j$, provided that $r.a\rho^{-1}$ is grounded by $\sigma_j\theta_j$. Among all candidate actions (computed for all rules r and for all $\rho_j^{-1}\sigma_j$, an action is then randomly selected.

Example 1 *We consider here the* Logistics *domain. Let us suppose we have a world composed of three trucks, three boxes and three cities and a current action model T. Let $r \in T.R$ be the following rule:*
$boxOnTruck(b_2, c_a), boxInCity(b_1, c_a), truckInCity(T_b, c_a)/$
$load(b_1, T_b)/boxOnTruck(b_1, T_b), \neg boxInCity(b_1, c_a)$

Algorithm 1 ACTIVE-SELECT(T,s)

Require: An action model T, and a state s
Ensure: An action a likely to yield a generalization of some rule in T

```
1: LA ← ∅
2: for all r ∈ T.R s.t r.p does not OI subsume s do
3:    for all (injective) post-matching substitutions ρ_j^{-1} and σ_j such that
      (r.e.del)ρ_j^{-1}σ_j ⊆ s do
4:       Compute lgg_j = lgg_OI((r.p)ρ_j^{-1}, s) a random lgg given σ_j (lgg_j σ_j θ_j ⊆ s)
5:       if r.a.ρ_j^{-1}σ_j θ_j is ground then
6:          LA ← LA ∪ {((r.a)σ_j θ_j, size(lgg_j))}
7:       end if
8:    end for
9: end for
10: if LA = ∅ then
11:    Randomly select an action a to apply to s
12: else
13:    Select a_i such that (a_i, size_i) ∈ LA and size_i is max in LA
14: end if
```

and suppose that the agent is in the following state s:
$boxInCity(b_1, c_b), truckInCity(t_b, c_b), boxInCity(b_2, c_a)$.

The rule r does not apply because there is no literal $boxOnTruck(b_2, c_a)$ in the current state s (condition line 2 of Alg.1 is true). The del *list of the rule, $\{boxInCity(b_1, c_a)\}$, generalized with inverse substitution $\rho_1^{-1} = \{c_a/X\}$ is included in the current state with substitution $\sigma_1 = \{X/c_b\}$. For these substitutions, Algorithm 1 computes a random least general generalization under Object Identity, namely $lgg_1 = boxInCity(b_1, X), TruckInCity(T_b, X)$ with $\theta_1 = \{T_b/t_b\}$. Substituting r.a with $\rho_1^{-1}\sigma_1\theta_1$ yields the ground action $load(b_1, t_b)$ added to LA.*

There is another couple of post-matching substitutions, $\rho_2^{-1} = \{b_1/Y\}$ and $\sigma_2 = \{Y/b_2\}$. For these substitutions, the following random lgg is computed: $lgg_2 = boxInCity(Y, c_a)$ with $\sigma_2 = \emptyset$. Substituting r.a with $\rho_2^{-1}\sigma_2\theta_2$ yields a non ground action $load(b_2, T_b)$, which is not added to LA. The agent applies the action $load(b_1, t_b)$ and the resulting state $s' = boxOnTruck(b_1, t_b), truckInCity(t_b, c_b), boxInCity(b_2, c_a)$ leads to an example which, as expected, is not covered by the current action model. The revision then consists in generalizing the rule r: the literal $boxOnTruck(b_2, c_a)$ is dropped from the preconditions of r.

The closest related work concerning active learning in the context of a RRL system is [4]. Our work mainly differs from this one because: i) it is fully online and incremental while restricted to a deterministic context; ii) it does not rely on any estimation of how much a relational state is known (fully or partially) or novel, which can be quite complex to evaluate in a relational context. We do not either use planning capabilities for our active learning strategy, which is quite simple : a state s is known by a rule r if the rule preconditions OI-subsume s and it is useful to apply action a in a state s if we expect that applying a to s

will generate a state s' such (s, a, s') may yield generalizing of a rule r of action a in the model. This strategy proves to be quite efficient in the following section.

4 Experimental results

We provide experimental results for both *blocks world* and *Logistics* domains, as in [1] and [5]. We consider a variant of the blocks world domain in which color predicates as $b(X)$ (black) and $w(X)$ (white) are introduced. In the *colored-blocks* world, when $move(X, Y)$ is chosen, X is actually moved on top of Y only if X and Y have the same color. Otherwise, X is not moved and its color shifts to the same color as Y. For instance, the 2-colors 7-blocks world is more challenging to learn than the 7-blocks world as the action model needs 7 rules to model the action *move*. We also consider the *logistics* domain as described in [1].

The IRALe approach has already been shown more effective than MARLIE [1] when measuring the prediction errors of the current model *w.r.t.* the number of actions performed by the agent, *i.e.* the total number of examples encountered [5]. This number is considered here as a time scale. By integrating learning with planning we can evaluate the model with respect to the actual purpose of the system, *i.e.* acting so as to fulfill assigned goals.

In what follows, each experiment is averaged over 100 runs. A run consists in performing an exploration of the environment starting from a random state and an empty model. During the run, the action model is periodically tested by executing 20 trials. Therefore each test corresponds to a certain number of actions performed. For each trial, start and goal states are drawn at random, ensuring that a path with less than 20 actions exists between them. The FF planner is then allowed a short time (10s) to find a plan. The trial is stated as a success if applying the plan results in reaching the goal state. Each test returns the *variational similarity* v_s computed as the average ratio of the number of successful plans obtained using the current model, to the number of successful plans using the perfect (hand coded) model.

For each experiment, we display the average *variational distance* $(1 - v_s)$ *vs* the number of actions performed for various exploration modes. The random exploration mode is compared to the ϵ_a-active exploration, with $\epsilon_a = 0.25$ and $\epsilon_a = 0.5$.

In Figures 1 and 2, we experiment IRALe extended with the active exploration strategy.

In all domains, adding active learning results in faster convergence to a null error model. Even a low proportion of active learning ($\epsilon_a = 0.25$) shows a clear improvement over pure random exploration. However a larger proportion of active learning ($\epsilon_a = 0.5$) does not improve the convergence speed.

5 Conclusion

In this paper, we propose an integrated system implemented in an autonomous agent situated in an environment. The environment is here supposed to be de-

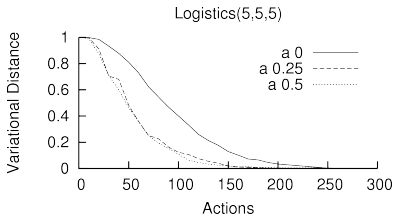

Fig. 1. Experiments in the Logistics(5,5,5) problem with increasing values of ϵ_a

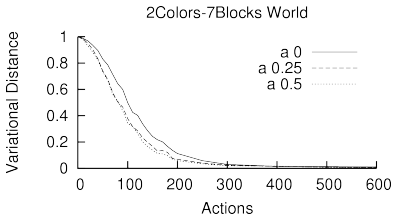

Fig. 2. Experiments in the 2-Colors 7-blocks problem with increasing values of ϵ_a

terministic: in a given state, the effects of a given action are unknown but determined. The agent uses the revision mechanism IRALe to perform online action model learning as he explores the environment by repetitively selecting and applying actions. The main contribution of this paper is the action selection strategy. Random selection is replaced, with probability ϵ_a, with an active selection mechanism that selects actions expected to enforce a modification of the current model. As a second contribution of the paper, the agent is equipped with planning capabilities, so we can evaluate the quality of the current action model in a realistic way: after the agent has performed a given number of actions, we can build plans to reach random state goals and estimate the proportion of plans that succeed using the current model.

Experimental results show that active learning, as implemented here, actually improves learning speed as follows: an accurate action model is obtained after performing much less actions than when using only random exploration. Regarding future works, active learning is limited here by the states accessible from the current state. Better active learning can be achieved by enabling the agent to plan *experiments*, *i.e.* to plan to reach some desirable, informative state. Finally an important perspective is to extend the system to handle noisy or indeterministic environments, using noise-tolerant revision algorithms.

References

1. T. Croonenborghs, J. Ramon, H. Blockeel, and M. Bruynooghe. Online learning and exploiting relational models in reinforcement learning. In *IJCAI*, pages 726–731, 2007.
2. S. Dzeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001.
3. F. Esposito, A. Laterza, D. Malerba, and G. Semeraro. Refinement of Datalog programs. In *MLnet Familiarization Workshop on ILP (KDD)*, pages 73–94, 1996.
4. T. Lang, M. Toussaint, and K. Kersting. Exploration in relational worlds. In *ECML/PKDD (2)*, pages 178–194, 2010.
5. C. Rodrigues, P. Gérard, C. Rouveirol, and H. Soldano. Incremental learning of relational action rules. In *ICMLA*, 2010.
6. R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2:160–163, July 1991.