Incremental learning of relational action models in noisy environments

Christophe Rodrigues, Pierre Gérard, and Céline Rouveirol

LIPN/A[3], University of Paris 13, email: firstname.lastname@lipn.univ-paris13.fr

Abstract. In the Relational Reinforcement Learning framework, we propose an algorithm that learns an action model (or an approximation of the transition function) in order to predict the resulting state of an action in a given situation. This algorithm learns incrementally a set of first order rules in a noisy environment following a data-driven loop. Each time a new example is presented that contradicts the current action model, the model is revised (by generalization and/or specialization). As opposed to a previous version of our algorithm that operates in a noise-free context, we introduce here a number of indicators attached to each rule that allows to evaluate if the revision should take place immediately or should be delayed. We provide an empirical evaluation on usual RRL benchmarks.

INTRODUCTION

In this paper[1], we propose an algorithm that simultaneously tackles the problems of incrementality and indeterminism in action model learning when using relational representations for states and actions. A relational representation is expected to have better generalization capabilities, improved scaling-up and transfer of solutions since it does not rely on a number of attributes describing states nor on their order.

When a system is involved in a sensori-motor loop with its environment, it perceives its state, chooses and performs an action before perceiving its new situation, acting again and so on. Action models permit to anticipate the outcomes of any action in a given state. Such models are necessary for planning and may be used in Reinforcement Learning (RL) to speed up the overall learning of the optimal action [13].

This work takes place at the intersection of Relational Reinforcement Learning ([5], see [14] for a review) and Planning in first order logics [12]. In both fields, automatically acquiring action models from experience is a major concern.

The main originality of our work is that it tackles both incremental learning and learning in non deterministic environments. In [11], we already proposed incremental algorithms in the deterministic case, with a convergence proof. By incremental learning, we mean revising the model each time a new example contradicting the current action model is presented to the system, without storing every example and periodically re-running batch learning. Incremental learning is suitable for designing adaptive systems. In the deterministic case, the order of the provided examples might be misleading and yield inadequate generalization (or specialization) choices, in particular when

[1] This work was partially supported by the ANR project HARRY.

an action has disjunctive preconditions. The system therefore has to be provided with mechanisms for reconsidering early inadequate decisions.

The deterministic assumption is quite a strong one as, in real world applications, the outcomes of actions are often stochastic (noise, inadequate representations, partial observability treated as stochasticity, etc.). Furthermore, indeterminism and incrementality should be tackled together, considering that in a non deterministic context, the order in which examples are presented to the system is even more critical.

On the one hand, several works in the Relational RL framework [2] or in the planning field [7, 1, 12, 17, 15] propose to learn action models incrementally, but they all only consider deterministic environments. On the other hand, other works [10, 9, 18] address stochasticity, but are limited to batch learning. Based on KWIK [8], [16] addresses stochastic problems but can hardly be considered as incremental : all examples are stored and a new batch-learning is performed each time. Other related work do neither tackle incrementality nor indeterminism such as INTHELEX [6]. Figure 1 gives an overview of related work.

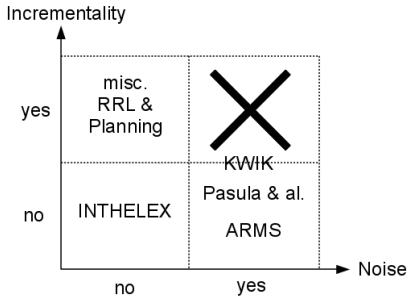

Fig. 1. Incrementality together with indeterminism

In Section 1, we detail our learning framework by describing the relational first order representation of the state and action spaces and introducing the general mechanisms of the algorithm presented in Section 2. Here, we provide an overview of the proposed incremental generalization and specialization mechanisms. Before concluding, the algorithms are empirically tested on the regular RRL benchmark environment.

1 LEARNING PROBLEM

1.1 Action model

An action model is a theory of the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$ of a Markov Decision Process (MDP) with a set $\mathcal{S}$ of possible states and a set $\mathcal{A}$ of possible actions. Here, $\Pi(\mathcal{S})$ denotes a probability distribution over the state space. Unlike in the deterministic case, the outcomes of the same action in the same state might be different from time to time, due to noise or other concerns.

An example x for the learning process is composed of a given state, a chosen action and the resulting new state. We respectively note them as $x.s$, $x.a$ and $x.s'$. The *effect* $x.e$ of an action describes what has changed in the state after applying the action, *i.e.* $x.e = \delta(x.s, x.s')$. Rather than strictly learning a model of the transition function, we model what changes when the action applies, by taking advantage of $x.s$, $x.a$ and $x.e$. Let us denote this model T.

Due to the model complexity, we have chosen not to learn the whole distribution probability over possible effects for a given action in a given state. We rather restrict the model T to most likely outcomes. We assume in this paper this will be sufficient to distinguish between noise and relevant information in most cases.

1.2 Relational representation

Examples. Examples are described by a Datalog-like language (no function symbol but constants). Objects are denoted by constants (denoted in the following as a, b, $f \ldots$). Variables are denoted by upper-case letters (X, $Y \ldots$). In a noise-free context, when an agent emits an action, the effect part completely describes the effects of the action: literals not affected by the action remain unchanged. The effect, as usual in a STRIPS-like notation, is composed of two literal sets: $x.e.add$ is the set of literals getting true when the action is performed, and $x.e.del$ is the set of literals getting false when the action is applied. The examples can be noted $x.s/x.a/x.e.add, x.e.del$, with no negated atoms in $x.s$, $x.a$ and $x.e.add$, and $x.e.del$ described as a conjunction of negated literals.

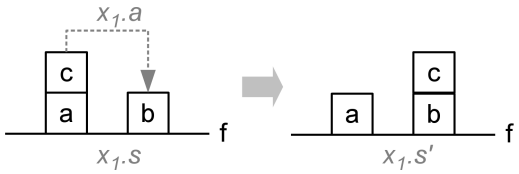

Fig. 2. In a simplified blocks world with only on and $move$ predicates, states and action yielding to example x_1: $on(a, f), on(b, f), on(c, a)/move(c, b)/on(c, b), \neg on(c, a)$

Rules. Most works in RRL use instance based methods [3] or decision trees [4] to simultaneously represent an action model and the value function associated to this model. Existing instance based methods use predefined distances suited to the problems. Decision trees are top-down methods which – in the incremental case — highly rely on the order of presentation of the examples, thus leading to over-specializations.

In this paper, we propose a rule-based representation of the action model T , because inadequate generalization/specialization choices in an incremental context can be easily reconsidered when actions are represented by independent rules and that no ad-hoc prior knowledge is needed, as it is the case for instance based approaches.

Each rule r is composed of a precondition $r.p$, an action $r.a$ and an effect $r.e$. The precondition is represented by a conjunction of positive literals, which have to be satisfied so as to apply the rule. The action is a literal defining the performed action. As for examples, the effect is composed of two literal sets, $r.e.add$ and $r.e.del$. An action $r.a$ has no other effects but those described by $r.e$. In order to be *well formed*, a rule r must be such that i) $r.e.del \subseteq r.p$ ii) $r.e.add \cap r.p \neq \emptyset$ iii) $r.a$ and $r.e$ must be connected. Finally, all variables occurring in $r.a$ should also occur in $r.p$ and $r.e$, but $r.p$ and $r.e$ may refer to objects/variables not occurring in $r.a$. Rules can be denoted $r.p/r.a/r.e.add, r.e.del$. There is no negated literal in $r.p$, and each literal of $r.e.del$ is negated.

This *extended STRIPS* formalism we adopt in this work is more expressive than the regular STRIPS language, as considered for instance in [15]. For a given action and effect, it is possible to associate several preconditions, each expressed as a conjunction of literals. This is not the case in [15], which only accepts conjunctive preconditions for an action. Moreover, our formalism accepts action rules where variables/objects not occurring in the action literals may occur in preconditions and/or effects. Learning here includes learning the "schema" of STRIPS-actions (number of rules, exact variables involved in the action).

1.3 Matching and covering

The following matching relationships all make use of OI-subsumption (subsumption under Object Identity) [6] as a *generality* relation. A formula G OI-subsumes a formula S iff there exists a substitution σ such that $G\sigma \subseteq S$, where σ is an injective substitution (two different variables of the domain of σ are assigned to different terms). For instance, $p(X, Y)$ does not OI-subsume $p(a, a)$ because X and Y cannot be assigned to the same constant.

The *pre-matching* relation $\overset{sa}{\sim}$ allows to decide whether a given rule may apply to predict the outcomes of a given example. For any rule r, state s and action a, $r \overset{sa}{\sim} (s, a)$ iff there exists injective substitutions σ and θ such that i) $r.a\sigma = a$ ii) $r.p\sigma\theta \subseteq s$.

The *post-matching* relation $\overset{ae}{\sim}$ permits to decide whether a given rule may explain a given state modification when a given action is performed. For any rule r, and action a and effect e, $r \overset{ae}{\sim} (a, e)$ 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) $r.e\rho^{-1}\sigma\theta = e$.

The above relations can be extended to matching between rules r and examples x, by adequately taking into account $x.s$, $x.a$ and $x.e$ instead of s, a and e in the definitions.

The *covering* relation $\approx$ permits to check whether an example can be accurately predicted by the model. For any rule r and example 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.

An example x *contradicts* a rule r ($x \nsim r$) if r pre-matches $(x.s, x.a)$ for σ and θ substitutions, and r does not post-match $(x.a, x.e)$ with the same substitutions. In such a case, the rule incorrectly predicts the outcomes of the action.

2 INCREMENTAL RELATIONAL LEARNING OF AN ACTION MODEL

2.1 Sketch of the algorithm

The system takes examples as defined in Section 1. The examples are presented incrementally, each one possibly yielding an update of the action model. The method we propose is example-driven and bottom-up. The starting point is thus an empty rule-set; the interactions between the system and its environment produce rules by computing least general generalization (*l*gg) under Object Identity between examples, and between rules and examples. This approach is different from — descendant — decision trees.

General rules are produced as the *l*gg of two rules/examples. When a *l*gg takes place, the resulting generalization keeps track of which rules/examples it comes from, each rule r therefore has a list of ancestors $r.anc$. Each ancestor rule might have ancestors as well, yielding a hierarchical structure.

This structure is used when specializing, in case an inadequate generalization is detected (see below). In that case, the corresponding rule is removed and replaced by one of its rule ancestors; specialization is called recursively until a trusted rule ancestor is reached.

Example ancestors that have been rejected during specialization are re-injected in the system for learning, and thus may lead to further revisions of the model. So as not to explore the same over-generalizations again, a tabu list is associated with such examples.

Let us suppose we observe a blocks world with three blocks and the floor f. By observing several moving actions, let us suppose the system has learnt the two following (correct) rules:

$$r_1 : cl(X), cl(Y), on(X,Z), bl(X), bl(Y), bl(Z) \;/\; move(X,Y) \;/\; on(X,Y), \neg cl(Y), \neg on(X,Z), cl(Z)$$

for stacking the top X of a two blocks stack on a single block Y, provided that both X and Y are clear of blocks and

$$r_2 : on(X,f), cl(X), cl(Y), on(Y,Z), bl(X), bl(Y) \;/\; move(X,Y) \;/\; on(X,Y), \neg cl(Y), \neg on(X,f)$$

for stacking a block X initially on the floor on another block Y (X and Y should also be clear of blocks). Let us now assume that the following noisy example occurs:

$$x_n : on(a,f), cl(a), cl(c), on(c,b), on(b,f), bl(a), bl(b), bl(c) \;/\; move(a,c) \;/\; on(a,c), \mathbf{cl(f)}, \neg cl(c), \neg on(a,f)$$

The (action,effects) of this example can be post-matched with r_1, yielding the following (incorrect) generalization

$$r_g : cl(X), cl(Y), on(X,Z), bl(Y), bl(X) \;/\; move(X,Y) \;/\; on(X,Y), \neg cl(Y), \neg on(X,Z), cl(Z)$$

while r_2 has to be specialized as it pre-matches the noisy example while not predicting the observed effects.

2.2 Conservative generalizations and specializations

Unlike in the deterministic case, examples may contradict each other, and a rule should not be specialized as soon as it is contradicted a given example (that may be noisy). Noisy examples may also induce over-generalizations. Thus, specializations and generalizations should be "cautious" and conservative. Therefore, we propose in the following an algorithm that delays actual generalizations and specializations until sufficient evidence has been collected. To that end, we attach basic estimates to each rule. They are combined to help decision making about when to generalize/specialize. To each rule r in T, we associate three basic estimates:

- $r.n_{sa}$ the number of examples pre-matched by the rule since its creation
- $r.n_{ae}$ the number of post-matched examples
- $r.n_{sae}$ the number of covered examples

Initial values for $r.n_{sa}$, $r.n_{ae}$ and $r.n_{sae}$ is 1. $(r.n_{ae} - r.n_{sae})$ is the number of times a rule post-matched an example without pre-matching it: if the rule was requested to predict (it wasn't because it is too specific), it would have predicted well. If this value is high *wrt* $r.n_{ae}$, it means that the rule could probably be generalized. The generalization trend $r.gen \in [0, 1]$ of a rule is defined as $\frac{r.n_{ae}-r.n_{sae}}{r.n_{ae}}$. A rule is generalized with an example only if $r.gen > \theta_{gen}$, where θ_{gen} is a threshold parameter of the system.. In addition, such a modification is decided only if the rule has been sufficiently evaluated, *ie* $r.n_{ae} > \theta_{evl}$, where θ_{evl} is another parameter of the system.

Similarly, a rule should be specialized if it doesn't prove to be accurate enough to predict well. Rule accuracy $r.acc$ is defined as $\frac{r.n_{sae}}{r.n_{sa}}$. If accuracy drops below a given threshold θ_{spc}, and if $r.n_{sa} > \theta_{evl}$, then r is specialized.

2.3 Covering and elimination of irrelevant rules

When a new example x is not covered by any rule, the algorithm updates T to make it complete. If no relevant generalization is identified, the example is simply added as a rule in the rule set, without ancestors.

This process ensures covering all examples including noisy ones, and may introduce many irrelevant rules that might be difficult to generalize, and that the algorithm should be able to identify and delete. So as to limit the complexity of the model, we bound the number of rules in T with a parameter N. Any new but supernumerary rule replaces another among most untrusted ones. To that end, a confidence estimate $r.cnf$ is associated with each rule. To be trusted, a rule should have been evaluated often enough, and its accuracy should be high. Therefore, $r.cnf = 0$ if $r.n_{sa} < \theta_{evl}$, and is equal to $r.acc$ otherwise.

2.4 Prediction with the model

The above mechanisms allow for contradictions between rules. As a result, when an example is provided for prediction, more than one rule may pre-match the current state and action. Among these rules, only one among the most confident ones is used to compute the predicted effect.

3 EMPIRICAL STUDY

For the experimental study, we use a blocks world, used as a benchmark in most RRL works. States and actions are described with literals and objects. Objects are either blocks (a, b ...) or the floor f. Predicates $on/2$ and $cl/1$ are used to describe the blocks layout. A predicate $bl/1$ states whether an object is a block. A block is moved on top of an object using the action predicate $move/2$.

Examples are generated in sequence : each episode stops after 10 time steps or when the goal is reached (stacking all blocks), sequential actions are randomly chosen. Every ten learning examples, twenty random examples are generated and used to estimate the accuracy of the model. The *error* is the percentage of unpredicted examples (at least one effect literal is uncorrectly predicted). Results are averaged over five experiments.

So as to introduce indeterminism, a perception noise is added : the states as observed by the system may be randomly altered. With probability ϵ, random predicates are added to or removed from the state. The number of additions/removals is randomly chosen between 1 and n_ϵ. Since the examples are provided in sequence, any noisy state s_t affects two examples : x_t and x_{t+1}. This kind of noise offers a high variety of possible irrelevant observations. This perceptual noise is different from the action-noise studied in [9] where it only consists in sometimes letting a block drop onto the table instead of getting stacked.

Figure 3 shows the evolution of the error along time steps, when the number of blocks grows. The amount of noise is fixed : $\epsilon = 10\%$. Figure 4 shows the evolution of the error along time steps, when the noise grows. The number of blocks is here fixed to 6. System parameters are $\theta_{gen} = \theta_{spc} = 0.9, \theta_{evl} = 3$ and $N = 20$ (max number of rules). Environment parameter n_ϵ is 2.

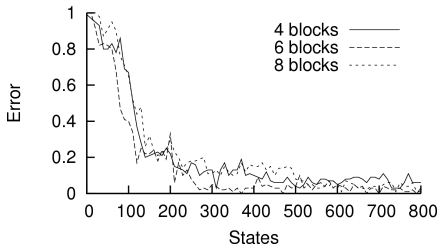

Fig. 3. 10% noise: 4, 6 and 8 blocks

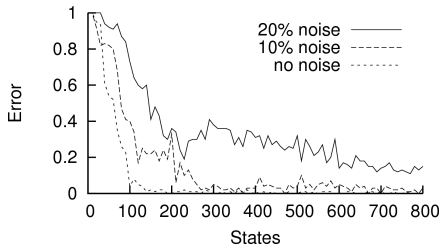

Fig. 4. 6 blocks: 0, 10 and 20% noise

The irregularity of the curves is due to the number of random samples used for evaluation *wrt* the size of the environments. Thanks to good generalization capabilities, error appears loosely dependent on the problem size. Our method is quite resistant to high level of noise, but the convergence speed drops with 20% noise. Indeed, with such an amount of noise, it is highly probable that when an action is observed, both initial and resulting states are corrupted. Together with a noise affecting several literals, very misleading and unlikely examples are presented for learning.

4 CONCLUSION

We have proposed in this paper an algorithm that tackles both problems of noise and full-incrementality for action model learning. Only very few examples are stored and the model might be revised for each new example. Unlike the deterministic case, this work is based on heuristics rather than careful enumerations. Simple estimates help to perform conservative and delayed generalizations and specializations of rules. The presented system proves to be efficient even with a fairly large amount of perception noise, and when the size of the problem grows. Future work will aim at proposing a fully incremental rule based system for regression, so as to approximate value-functions.

References

1. S. Benson. Inductive learning of reactive action models. In *ICML 1995*, pages 47–54, 1995.
2. T. Croonenborghs, J. Ramon, H. Blockeel, and M. Bruynooghe. Online learning and exploiting relational models in reinforcement learning. In *IJCAI*, pages 726–731, 2007.
3. K. Driessens and J. Ramon. Relational instance based regression for relational reinforcement learning. In *ICML*, pages 123–130, 2003.
4. K. Driessens, J. Ramon, and H. Blockeel. Speeding up relational reinforcement learning through the use of an incremental first order decision tree algorithm. In *ECML, LNAI 2167*, pages 97–108, 2001.
5. S. Dzeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001.
6. F. Esposito, G. Semeraro, N. Fanizzi, and S. Ferilli. Multistrategy theory revision: Induction and abduction in INTHELEX. *Machine Learning*, 38(1-2):133–156, 2000.
7. Y. Gil. Learning by experimentation: Incremental refinement of incomplete planning domains. In *ICML*, pages 87–95, 1994.
8. L. Li, M. L. Littman, and T. J. Walsh. Knows what it knows: a framework for self-aware learning. In *ICML*, pages 568–575, 2008.
9. H. M. Pasula, L. S. Zettlemoyer, and Kaelbling L. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research (JAIR)*, 29:309–352, 2007.
10. H. M. Pasula, L. S. Zettlemoyer, and Pack Kaelbling L. Learning probabilistic planning rules. In *ICAPS*, pages 146–163, 2004.
11. C. Rodrigues, P. Gérard, C. Rouveirol, and H. Soldano. Incremental learning of relational action rules. In *ICMLA*. IEEE Computer Society to appear, 2010.
12. W. M. Shen. Discovery as autonomous learning from the environment. *Machine Learning*, 12(1-3):143–165, 1993.
13. R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ICML*, pages 216–224, 1990.
14. M. Van Otterlo. *The logic of adaptive behavior*. PhD thesis, University of Twente, Enschede, 2008.
15. T. J. Walsh and M. L. Littman. Efficient learning of action schemas and web-service descriptions. In *AAAI*, pages 714–719, 2008.
16. T. J. Walsh, I. Szita, M. Diuk, and M. L. Littman. Exploring compact reinforcement-learning representations with linear regression. In *UAI*, pages 714–719, 2009.
17. X. Wang. Learning by observation and practice: An incremental approach for planning operator acquisition. In *ICML*, pages 549–557, 1995.
18. Q. Yang, K. Wu, and Y Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2-3):107 – 143, 2007.