# 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.** We consider an agent which learns a relational action model in order to be able to predict the effects of his actions. The model consists of a set of STRIPS-like rules, i.e. rules predicting what has changed in the current state when applying a given action as far as a set of preconditions is satisfied by the current state. Here several rules can be associated to a given action, therefore allowing to model conditional effects. Learning is *online*, as examples result from actions performed by the agent, and *incremental*, as the current action model is revised each time it is contradicted by unexpected effects resulting from his actions. The form of the model allows using it as an input of standard planners. In this work, the learning unit IRALe [1] is embedded in an integrated system able to i) learn an action model ii) select its actions iii) plan to reach a goal. The agent uses the current action model to perform active learning, i.e. to select actions with the purpose of reaching states that will enforce a revision of the model, and uses its planning abilities to have a realistic evaluation of the accuracy of the model.

## 1 Introduction

Adaptive behaviour studies how an autonomous agent can modify its own behaviour so as to adapt to a complex, changing and possibly unknown environment. Any adaptive agent needs to simultaneously learn from its experience, and act so as to fulfill various goals. Thus, an adaptive system needs to integrate some kind of *online* learning together with action selection mechanisms. When the agent's knowledge is constantly revised as new examples are encountered, rather than built from scratch, learning is stated as *incremental*.

In this work, the agent knows which actions it can perform and has a complete representation of the current state, representing both his own state and the environment state. He sequentially performs actions and each action applied will change the current state into a new state, forming a *trajectory* in the space of states. The difference between these two states is considered as the *effect* of performing this action in the current state. Now, we would like the agent to be able to rationally select actions, in order to reach goals. Then, there are basically two ways for selecting actions: either, as in classical Reinforcement Learning, by learning directly to predict in each state what should be the next

[1] This research was supported in part by the French ANR HARRI project *JC*08_313349.

action to perform (the most valuable action given the goal) or, as in *indirect* Reinforcement Learning by learning and using separately an *action model* that predicts the effect of applying a given action in the current state. Here, we adopt the latter setting, so that the action model can be used by a symbolic planner to build a plan to reach the goal. The learning task of the agent addressed here is to revise and maintain along the trajectory of the agent an accurate action model.

In this paper, states and actions are represented using restrictions of first order languages. The corresponding *relational* action model has a concise representation and does not assume that the number and the ordering of the objects in the environment are known *a priori*. As a consequence, straightforward *transfer* of the learned model from simple to complex problems is expected.

Adaptation within relational representations is primarily addressed by Relational Reinforcement Learning (RRL) [7] by extending the classical Reinforcement Learning (RL) problem to first order representations. *Indirect* RL [25] proved to be very efficient with relational representations [4].

We have recently proposed a relational revision algorithm implemented in IRALe [22] which starts from a (usually) empty action model and performs online learning of a deterministic conditional STRIPS-like model. More precisely, given a state space $\mathcal{S}$ and an action space $\mathcal{A}$, learning an action model $T$ consists in learning a transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. In deterministic Markov Decision Processes that we consider here, predicting the effect of performing action $a$ in state $s$ results in a new state $\hat{s'}$. In the following, we state the prediction as *correct* whenever $\hat{s'} = s'$. When $\hat{s'} \neq s'$ we have a prediction error also denoted as a *mistake*. Furthermore, we only consider the *realizable* case, *i.e.*, the case where a perfect action model $T$ exists in the class of models $\mathcal{T}$ that the learning procedure investigates. As a consequence, learning an action model comes down to searching for an element in $\mathcal{T}$ that will never make any prediction error.

The main features of the learning algorithm in IRALe are as follows: *i)* $T$ is represented as a relational rule set, *ii)* IRALe is a revision algorithm: it incrementally revises the model $T$ when $T$ makes a prediction error *iii)* among all the $(s, a, s')$ examples sequentially encountered by the agent, IRALe only memorizes those, denoted as *counter-examples*, associated to a prediction error and that have therefore enforced a revision of the model $T$. As mentioned in [22], storing counter-examples is enough to guarantee that the number of mistakes is bounded in the realizable case.

We study in this work an extension of IRALe, consisting in providing the agent with *active learning* capabilities (see the recent survey by [23]), allowing it to select actions to perform in order to improve its action model. The main motivation for introducing active learning in online action model learning is that when the action model becomes accurate, the probability to encounter counter-examples when performing random actions, and therefore to revise the model, severely decreases. However, if the agent is able to select an action that could result in an unexpected effect, then, while the correct model is expected to be reached after a number of mistakes similar to pure random exploration, this

number of mistakes will be encountered after much less actions have been performed. We propose in the following such an *active* action selection mechanism. Overall, the agent trajectory depends then on how each action is selected. Here, with probability $(1 - \epsilon_a)$, the action is randomly selected, and with probability $\epsilon_a$ the action is selected following the *active* action selection mechanism.

A second contribution of this paper concerns the integration of online learning and *planning*. By introducing planning capabilities, we intend to allow the agent to build plans relying on the current action model, in order to reach goals expressed as goal states or as conjunctions of constraints on the final state. Our autonomous agent has been provided with planning capabilities allowing to evaluate the quality of the current action model by its ability to generate plans to reach variable goals interactively provided to the agent.

Section 2 summarizes the online action model learning mechanism of IRALe. Section 3 provides details on the main components of this architecture, namely the active exploration mechanism added to the basic random exploration mechanism. Finally, in section 4 we experiment our integrated agent and discuss the results of experiments concerning various benchmark action models.

## 2 On-line Learning of a Relational Action Model

In this section, we first investigate related work concerning the task of learning an action model, and then we summarize the IRALe revision algorithm [22]. IRALe is a theory revision algorithm dedicated to relational action rule learning. Overall, the STRIPS-like resulting action model is a set of rules. Several rules are associated to each action, each rule predicting all the effects of the action when triggered in some particular conditions regarding the current state. In this way, the model allows to represent conditional effects. IRALe only stores *counter-examples*, namely examples that have provoked a prediction mistake at some point during the model construction, and may be therefore considered as a *partial memory incremental learner* [15].

### 2.1 Related work

Learning planning operators has been studied intensively, including the problem of learning the effects of actions in the context of RRL. The RRL system most closely related to ours is MARLIE [4], the first Relational RL system integrating incremental action model and policy learning. MARLIE uses TG [6] to learn relational decision trees, each used to predict whether a particular literal is true or false in the resulting state. The decision trees are not restructured when new examples appear that should lead to reconsider the internal nodes of the tree. Other systems [5] integrating such restructuring operators scale poorly. Moreover, this kind of representation of an action model is not very concise and, above all, does not allow direct integration with a symbolic planner: given a state $s$, and the set of trees induced by TG, it is indeed an expensive process to predict what is the resulting state after applying an action $a$.

In the planning field, several works aim at learning action models but show limitations *w.r.t.* the autonomous and adaptive agent framework. Benson's work [2] relies on an external teacher; EXPO [9] starts with a given set of operators to be refined, and cannot start from scratch; OBSERVER [28] knows in advance the number of STRIPS rules to be discovered and which examples are indeed relevant to each rule; LIVE [24] doesn't scale up very well: it proceeds by successive specializations but cannot reconsider early over-specializations.

In the ILP context, learning relational action rules has been studied by Otero et al. in the context of monotonic learning [17, 18]. Action rules in these works predict as in [4] a single effect literal (positive or negative). After a logical formalization for the frame problem, authors restrict examples to predict changes between consecutive states (for instance, a false effect literal in state $s_i$ becoming true in state $s_{i+1}$ forms a positive example for the effect literal). They show that this way of modeling allows to learning action rules in a standard ILP monotonic setting. We address the same problem by restricting action learning to learning the preconditions associated to the DEL and ADD effects in an extended STRIPS like formalism. Note that the ramification problem addressed in [18] does not occur with our action model as triggering one rule predicts all the effects of the corresponding action.

Other works [20, 19, 29] address stochasticity, but are limited to batch learning. Based on the KWIK framework [14], [27] addresses stochastic problems but can hardly be considered as incremental: all examples are stored and a new batch-learning is performed each time. Finally, another line of research is the PELA architecture, which integrates components that learn and use probabilistic action models [11]. In PELA, an initial action model is learned with the TILDE algorithm, a batch relational decision tree.

On the side of ILP revision, there have been quite a number of relational theory revision systems [21, 1], the closest to IRALe being the Inthelex system [8]. Inthelex and IRALe have similar ascending operators, and differ substantially concerning other operators, in particular specialization. Moreover, except the work described in [3], Inthelex uses a full memory storage, and up to our knowledge, has not been adapted to action learning problems.

### 2.2 Incremental Relational Action Model Learning

**States, actions, examples and rules** States and actions are represented by objects and relations between them. Examples are provided online to the agent which minimally revises the action model when needed. Relations between objects are described using predicates applied to constants. In the following, objects are denoted by constants, denoted by a lower-case character ($a$, $b$, $f$, ...). Variables are denoted by an upper-case character ($X$, $Y$, ...), and may instantiate to any object of the domain. A *term* is here a constant or a variable. Actions and relations between objects are denoted by *predicate* symbols. For instance, in a blocks world, if a block $a$ is on top of an unknown block $X$, this is denoted by the literal $on(a, X)$.

Examples are described as sets of conjunctions of ground literals. As usual, we assume that when the agent performs 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 discribing the state, $x.a$ a literal of action and finally an effect part with $x.e.add$ a conjunction of positive literals and $x.e.del$ a conjunction of negated literals, as usual in a STRIPS-like notation. For instance, in the blocks world, Fig..1 shows an example of the action *move*:
$on(a, f), on(b, f), on(c, a)/move(c, b)/on(c, b), \neg on(c, a)$.

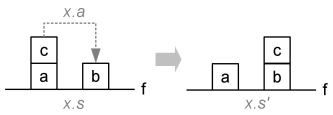


**Fig. 1.** Example of a *move* action in a simple blocks world

IRALe builds an action model $T$ represented as a set of rules $T.R$ and a set of counter-examples $T.X$ 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 $r.p$ / $r.a$ / $r.e$. The precondition is 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. The effect is composed of two sets of literals: $r.e.add$ is the set of literals getting true when the rule is applied, and $r.e.del$ is the set of literals getting false when the rule is applied. According to a rule $r$, 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) $r.e.add \cap r.p = \emptyset$ iii) $r.a \cup r.e$ must be connected[2]. Finally, all variables occurring in $r.a$ should also occur in $r.p$ and $r.e$, but $r.p$ and $r.e$ may reference objects/variables not occurring in $r.a$. (see deitic references [19]). For instance, a well-formed rule is the following:
$on(X, Z), on(Y, W)/move(X, Y)/on(X, Y), \neg on(X, Z)$.

This formalism, to which we refer as *Extended Deterministic* STRIPS (EDS), is more expressive than *Deterministic* STRIPS, considered for instance in [26]: a given action may be represented by several rules.

**Rule covering and contradiction** Matching operations between rules and examples rely on a *generality* relation frequently used in the Inductive Logic Programming (ILP) framework: subsumption under Object Identity, denoted as OI-subsumption [8]. Given the OI Bias, different constants or variables must be distinct. This assumption is natural when learning action rules for planning

[2] Any variable occurring in $r.e$ (resp. $r.p$) should be linked through a path of literals to a variable occurring in the literal of action $r.a$.

[19]. A formula $G$ OI-subsumes a formula $S$ iff there exists an injective substitution[3] $\sigma$ such that $G\sigma \subseteq S$. Two formulas may have several least general generalizations (*lgg*'s) under OI-subsumption, each corresponding to a largest common substructure between the input formulas.

Rule matching is defined as follows.

**Definition 1 (pre-matching $\overset{sa}{\sim}$ and post-matching $\overset{ae}{\sim}$).** *For any rule $r$, state $s$, action $a$ and effect $e$,*

- $r \overset{sa}{\sim} (s, a)$ *iff there exists two injective substitutions $\sigma$ and $\theta$ such that i) $(r.a)\sigma = a$, and ii) $(r.p)\sigma\theta \subseteq s$.*
- $r \overset{ae}{\sim} (a, e)$ *iff there exists two injective substitutions $\sigma$ and $\theta$ such that i) $(r.a)\sigma = a$ ii) $(r.e)\sigma\theta = e$.*

Pre-matching $\overset{sa}{\sim}$ checks whether a given rule may apply to predict the effect of a given example, and post-matching $\overset{ae}{\sim}$ checks whether a given rule may explain a given state modification when a given action is performed.

The question of whether the action model contradicts or is consistent with an example is addressed through the following definitions.

**Definition 2 (covering $\approx$ and contradiction $\nsim$).** *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 $\sigma$ and $\theta$.*
- $x \nsim r$ *if* $r \overset{sa}{\sim} (x.s, x..a)$ *for some injective $\sigma$ and $\theta$ substitutions, and $r$ not* $\overset{ae}{\sim} (x.a, x.e)$ *with the same substitutions.*

Covering checks whether the effect part of an example is accurately predicted by a rule of the model, and contradiction appears when a rule applies but incorrectly predicts the outcomes of the action. A default rule is implicitly added to $T$: whenever no rule applies, the prediction is that the action produces no effect, i.e. $e.del = e.add = \emptyset$.

The model $T$ needs to be revised whenever the current action model fails to predict the effect part of some incoming example. In order to ensure convergence of the action model, the model includes a subset $T.X$ of the examples which have been met since the beginning of the learning session. These examples, denoted as counter-examples and defined hereunder, are all those that have previously enforced a revision of the model:

**Definition 3 (counter-example).** *$x_u$ is a counter-example and is said to contradict the model $T$ iff either there is no rule $r \in T.R$, such that $r \approx x$ or there is some rule $r'$ contradicted by $x_u$.*

[3] Two different variables of the domain of $\sigma$ are assigned to different terms: for instance, $p(X, Y)$ does not OI-subsume $p(a, a)$ because $X$ and $Y$ can't be assigned to the same constant.

Any incoming example $x_u$ may be a counter-example either because no rule pre-matches it (*completeness* issue) or because there are rules that pre-match the counter-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 in $T.X$.

**Online revision of the action model** At first, the action model is an empty rule-set. The interactions between the agent and the environment produce examples, and when an example contradicts the model, the latter has to be revised by modifying or adding one or several rules.

More precisely, when such a new counter-example $x_u$ is encountered, two kinds of modifications may have to be performed, either generalization or specialization. Specialization is not central to this work, we therefore refer to [22] for further details. As far as generalization is concerned, it takes place in order to preserve completeness of the model if no rule of $T.R$ pre-matches $x_u$. The rules $r$ of $T.R$ which are candidates for generalization are such that $r$, up to an inverse substitution $\rho^{-1}$, post-matches $x_u$. The role of $\rho^{-1}$ is to generalize constants that occur in $r$ and more precisely in effects of $r$ into variables, if necessary. Preconditions of $r.\rho^{-1}$ are then generalized with $x_u$ using a least general generalization under OI subsumption operator, denoted as lgg. If such generalization does not contradict any example in $T.X$ (preserving coherence), $r$ is thus replaced by the new minimally generalized rule, otherwise backtracking takes place as several minimal generalizations may exist under OI-subsumption. If no consistent generalization exists, $x_u$ becomes a rule and is added as such to $T.R$. Note that $x_u$, as a counter-example, is stored in $T.X$.

When a generalization is performed, the resulting generalization keeps track of which rules/examples it comes from. This way, each rule in $T.R$ is the top node of a memory tree. This structure is used during specialization which consists in backtracking on ancestors of over-generalized rules.

| | | | |
|---|---|---|---|
| $x_1$ | $boxInCity(b_1, c_1), truckInCity(t_1, c_1)$ | $load(b_1, t_1)$ | $boxOnTruck(b_1, t_1)$<br>$\neg boxInCity(b_1, c_1)$ |
| $x_2$ | $truckInCity(t_1, c_1), boxInCity(b_1, c_2)$ | $drive(t_1, c_2)$ | $truckInCity(t_1, c_2)$<br>$\neg truckInCity(t_1, c_1)$ |
| $x_3$ | $boxInCity(b_1, c_2), truckInCity(t_1, c_2),$<br>$boxOnTruck(b_2, t_1)$ | $load(b_1, t_1)$ | $boxOnTruck(b_1, t_1)$<br>$\neg boxInCity(b_1, c_2)$ |

**Table 1.** Relational representation of actions examples in a logistic problem described in Section 4.

In Example 1, we illustrate the revision process with a sequence of actions and revisions in a small logistic problem composed of one truck, two boxes and two cities (for the sake of readability, types predicates are omitted here and states are simplified).

**Example 1** *Referring to the examples of Table 1, the agent starts with an empty model. When the example $x_1$ is encountered, $x_1$ is added as a rule $r_1$ in T.R, and added to the counter-example memory T.X. Later, the example $x_2$ is encountered, the model is unable to cover this example because $x_2$ doesn't post-match $r_1$. $x_2$ is then also added to the counter-example memory T.X and in T.R as rule $r_2$. Then, $x_3$ occurs and there is no rule in T.R that pre-matches this example, and therefore the model cannot predict the effects of $x_3$. However, turning some constants of $x_1$ into variables (with the inverse substitution $\rho^{-1} = \{c_1/C\}$), and applying a least general generalization on pre-conditions (forgetting $boxOnTruck(b_2, t_1)$) generates the generalized rule $r'_1$ , that post-matches $x_3$ with empty substitutions $\sigma$ and $\theta$. $x_2$ also pre-matches $x_3$ and is stored as the rule $r_2$ (see Table 2). After these three examples have been handled, the model is composed of the rules $r'_1$ and $r_2$. All met examples are stored as counter-examples.*

| | | | |
|---|---|---|---|
| $r'_1$ | $boxInCity(b_1, C), truckInCity(t_1, C)$ | $load(b_1, t_1)$ | $boxOnTruck(b_1, t_1)$<br>$\neg boxInCity(b_1, C)$ |
| $r_2$ | $truckInCity(t_1, c_1), boxInCity(b_1, c_2)$ | $drive(t_1, c_2)$ | $truckInCity(t_1, c_2)$<br>$\neg truckInCity(t_1, c_1)$ |

**Table 2.** $x_1$ and $x_3$ are generalized into $r'_1$; $x_2$ is incorporated in the model as $r_2$.

## 3 Active Learning as action selection

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 $\epsilon$-active exploration: with probability $1-\epsilon_a$, the action to perform is chosen randomly, otherwise the action is selected following an active exploration process. In the random mode, as in MARLIE [4], in a given state, any syntactically correct action can be selected and performed by the agent, and not only the *legal* ones. We consider as *legal* an action that has observable effects (for example, moving a clear block on another clear block), while an *illegal* action has no observable effects, i.e. is such that $s = s'$. 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). Learning an action model is much more difficult when the agent does not know which actions are legal in a given state. In the following, a *syntactically correct* action is an action instantiated with any object of the world satisfying type constraints (when available).

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 state/action/$state_1$/$action_1$/$state_2$/ .../$state_n$ 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 which effects are, in a sense defined below, 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 $a$. As IRALe is mainly bottom-up, founding new opportunities for generalizing the model will decrease the number of examples necessary to converge to the correct model.

More precisely, in the current state $s$, and given the current action model $T$, we consider an action $a$ such that no rule about action $a$ applies to (pre-matches) the current state. This means that, following the model, we expect no effect when applying $a$ in the current state, i.e. we expect that, applying $a$, the agent would observe the example $s, a, s$. However, applying $a$ could result in an effect, i.e. in an example $s, a, s'$ with $s' \neq s$, that would then be a counter-example enforcing a revision of the model. This leads to select an action associated to a rule in the model whose preconditions are *almost* satisfied in the current state. This basically means that we hope that the observed effects will be those predicted by the rule whereas the preconditions turned to be overspecific. This expectation relies on the fact that the precondition part of such a rule is built by least general generalization, and therefore the preconditions are almost never more general than necessary (except if alternative generalization paths are possible). Of course, such an attempt can fail: possibly the effects in $s'$ are not those expected, meaning that the rule cannot be generalized.

**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 $T$

$LA \leftarrow \emptyset$
**for all** $r \in T.R$ s.t $r.p$ does not OI subsume $s$ **do**
  **for all** (injective) post-matching substitutions $\rho_j^{-1}$ and $\sigma_j$ such that $(r.e.del)\rho_j^{-1}\sigma_j \subseteq s$ **do**
    Compute $lgg_j = \text{lgg}_{OI}((r.p)\rho_j^{-1}, s)$ a random lgg given $\sigma_j$ ($lgg_j\sigma_j\theta_j \subseteq s$)
    **if** $r.a.\rho_j^{-1}\sigma_j\theta_j$ is ground **then**
      $LA \leftarrow LA \cup \{((r.a)\sigma_j\theta_j, size(lgg_j))\}$
    **end if**
  **end for**
**end for**
**if** $LA = \emptyset$ **then**
  Randomly select an action $a$ to apply to $s$
**else**
  Select $a_i$ such that $(a_i, size_i) \in LA$ and $size_i$ is max in $LA$
**end if**

Technically, in the current state $s$, the method considers 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-matches $s$, i.e. such that $r.e.del$, generalized with inverse substitution $\rho_j^{-1}$, is included in the current state $s$ up to an injective substitution $\sigma_j$. ACTIVE-SELECT then computes, for all corresponding $\rho_j^{-1}\sigma_j$, a random least general OI-generalization 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 $\rho_j^{-1}\sigma_j$, the action generated with the longest lgg is then selected.

**Example 2** *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:*
$boxOnTruck(b_2, c_a), boxInCity(b_1, c_a), truckInCity(T_b, c_a)/$
$load(b_1, T_b)/boxOnTruck(b_1, T_b), \neg boxInCity(b_1, c_a)$
*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 [13]. This work focuses on the adaptation of the $E^3$(*Explicit Explore or Exploit*) [12] algorithm for the relational case. In order to realize this adaptation, the batch system [19] is used to learn a stochastic action model. This work shows the importance of active exploration in relational worlds. 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 new, 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 as $(s, a, s')$

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 Experiments

The IRALe approach has already been shown more effective than MARLIE [4] 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 [22] which is considered here as a time scale. By integrating learning with planning we can evaluate the model with respect to the actual purpose of such agents, *i.e.* acting so as to fulfill assigned goals. In this paper, we generate random goals so as to evaluate learned models not only using classification error rate, but also considering the accuracy of the model with respect to planning tasks: during the trajectory, the agent periodically builds a plan from the current state to a random goal, using the current action model and the planner FF [10], and then the experimental device simulates the application of the plan and checks whether the plan has succeeded or failed. For that purpose, the goal, domain and action model are translated into an equivalent PDDL [16] planning task. Note that, as in each rule of the action model, all the effects are predicted as a whole, and the translation is straightforward. In systems like [4], a tree is built for each predicate symbol and then the translation into an explicit action model useful for a planner is an open problem.

### 4.1 Problems

We provide experimental results[4] for both *blocks world* and *Logistics* domain, as in [4] and [22]. We consider a variant of the blocks world domain in which color predicates as $b(X)$ (black) and $w(X)$ (white) are introduced. This domain is more challenging, it requires learning disjunctive rules for capturing the impact of color on action move. 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*.

In the logistics domain, the predicates $city/1$, $truck/1$ and $box/1$ indicate the type of the objects. In the $(b, c, t)$-Logistics setting, a state of the world describes $b$ boxes, $c$ cities, and $t$ trucks. Available actions are $load/2$ (load a box on a truck), $unload/2$ (unload a box in a city) and $drive/2$ (move a truck to a city), states are defined using the predicates $boxOnTruck/2$, $truckInCity/2$ and $boxInCity/2$.

### 4.2 Experimental set-up and results

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

[4] we use the same domains as MARLIE since this system isn't available

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* $(1 - v_s)$ *versus* the number of actions performed for various exploration modes. The random exploration mode ($\epsilon_a = 0$) is compared to the $\epsilon_a$-active exploration (where an action is actively explored with probability $\epsilon_a$ or randomly chosen with probability $1 - \epsilon_a$).

In Figures 2 and 3, we experiment with IRALe extended with the active exploration strategy and we display the variational distance versus the number of actions performed by the agent during his trajectory. Two active exploration rates $\epsilon_a = 0.25$ and $\epsilon_a = 0.5$ are investigated.

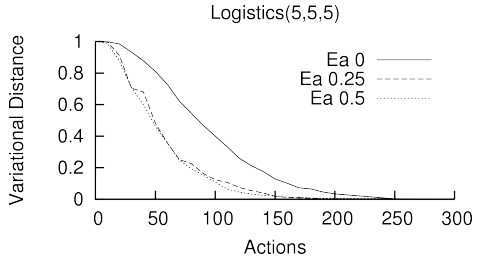


**Fig. 2.** Experiments in the Logistics(5,5,5) problem with increasing values of $\epsilon_a$

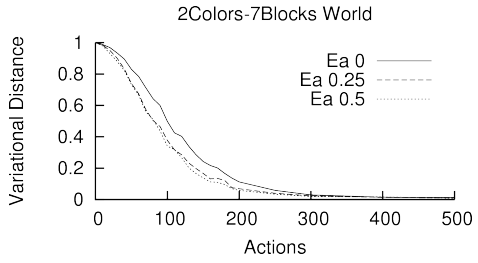


**Fig. 3.** Experiments in the 2-Colors 7-blocks problem with increasing values of $\epsilon_a$

In both domains, adding active learning results in faster convergence to a null variational distance. 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.

The memory of the model contains all the counter-examples encountered during learning. In Figures 4 and 5, we show the amount of counter-examples in the model after each action step. This results are for the same problems and active exploration rates as in Figures 2 and 3.

For a fixed learning step (*i.e.* action), the number of counter-examples in the model is always higher when active learning is used. This means that active learning influences positively space exploration in focusing the action choice on areas where the model is supposed to be incomplete. This results in a gain in action steps for increasing the quality of the model.

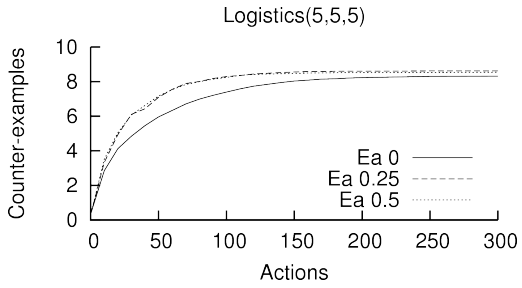


**Fig. 4.** Experiments in the Logistics(5,5,5) problem with increasing values of $\epsilon_a$

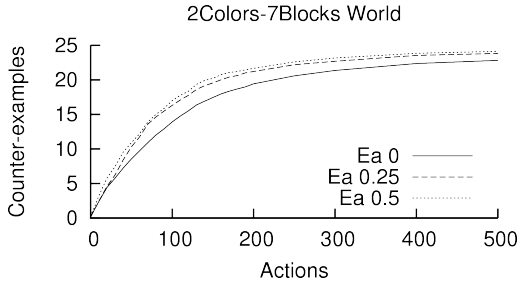


**Fig. 5.** Experiments in the 2-Colors 7-blocks problem with increasing values of $\epsilon_a$

## 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 deterministic: in a given state, the effects of a given action are unknown but determined. The agent uses the revision mechanism of the IRALe system to perform online action model learning as it explores the environment by repeatedly selecting and applying actions. The main contribution of this paper is the action selection strategy. Random selection is replaced, with probability $\epsilon_a$, 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 as to evaluate the quality of the current action model in a realistic way: after the agent has performed a given number of actions, plans are build 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 in the following sense: 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. H. Ade, B. Malfait, and L. De Raedt. Ruth: an ilp theory revision system. In Zbigniew Ras and Maria Zemankova, editors, *Methodologies for Intelligent Systems*, volume 869 of *Lecture Notes in Computer Science*, pages 336–345. Springer Berlin / Heidelberg, 1994.
2. S. Benson. Inductive learning of reactive action models. In *ICML 1995*, pages 47–54, 1995.
3. Marenglen Biba, Stefano Ferilli, Floriana Esposito, Nicola Di Mauro, and Teresa Maria Altomare Basile. A fast partial memory approach to incremental learning through an advanced data storage framework. In *Proceedings of the Fifteenth Italian Symposium on Advanced Database Systems, SEBD 2007*, pages 52–63, 2007.
4. T. Croonenborghs, J. Ramon, H. Blockeel, and M. Bruynooghe. Online learning and exploiting relational models in reinforcement learning. In *IJCAI*, pages 726–731, 2007.
5. W. Dabney and A. McGovern. Utile distinctions for relational reinforcement learning. In *IJCAI*, pages 738–743, 2007.
6. 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.
7. S. Dzeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001.

8. Floriana Esposito, Stefano Ferilli, Nicola Fanizzi, Teresa Maria Altomare Basile, and Nicola Di Mauro. Incremental learning and concept drift in inthelex. *Intell. Data Anal.*, 8(3):213–237, 2004.
9. Y. Gil. Learning by experimentation: Incremental refinement of incomplete planning domains. In *ICML*, pages 87–95, 1994.
10. Jörg Hoffmann. Ff: The fast-forward planning system. *The AI Magazine*, 2001.
11. S. Jiménez, F. Fernández, and D. Borrajo. The pela architecture: integrating planning and learning to improve execution. In *23rd national conference on Artificial intelligence - Volume 3*, pages 1294–1299. AAAI Press, 2008.
12. M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49, 2002.
13. T. Lang, M. Toussaint, and K. Kersting. Exploration in relational worlds. In *ECML/PKDD (2)*, pages 178–194, 2010.
14. 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.
15. Marcus A. Maloof and Ryszard S. Michalski. Incremental learning with partial instance memory. *Artif. Intell.*, 154(1-2):95–126, 2004.
16. Drew McDermott. The 1998 ai planning systems competition. *AI Magazine*, 21(2):35–55, 2000.
17. Ramón P. Otero. Induction of the effects of actions by monotonic methods. In *Inductive Logic Programming: 13th International Conference, ILP 2003*, volume 2835 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2003.
18. Ramón P. Otero. Induction of the indirect effects of actions by monotonic methods. In Stefan Kramer and Bernhard Pfahringer, editors, *Inductive Logic Programming, 15th International Conference, ILP 2005, Bonn, Germany, August 10-13, 2005*, volume 3625, pages 279–294, 2005.
19. H. M. Pasula, L. S. Zettlemoyer, and L. Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research (JAIR)*, 29:309–352, 2007.
20. H. M. Pasula, L. S. Zettlemoyer, and Pack Kaelbling L. Learning probabilistic planning rules. In *ICAPS*, pages 146–163, 2004.
21. B. L. Richards and R. J. Mooney. Automated refinement of first-order horn-clause domain theories. *Machine Learning*, 19:95–131, 1995.
22. C. Rodrigues, P. Gérard, C. Rouveirol, and H. Soldano. Incremental learning of relational action rules. In *ICMLA*, 2010.
23. Burr Settles. Active Learning Literature Survey. Technical Report Technical Report 1648, University of Wisconsin-Madison, 2009.
24. W. M. Shen. Discovery as autonomous learning from the environment. *Machine Learning*, 12(1-3):143–165, 1993.
25. R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2:160–163, July 1991.
26. T. J. Walsh and M. L. Littman. Efficient learning of action schemas and web-service descriptions. In *AAAI*, pages 714–719, 2008.
27. 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.
28. X. Wang. Learning by observation and practice: An incremental approach for planning operator acquisition. In *ICML*, pages 549–557, 1995.
29. 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.