Journal of Machine Learning Research 18 (2017) 1-44 Submitted 6/16; Revised 6/17; Published 8/17

# Relational Reinforcement Learning for Planning with Exogenous Effects

**David Martínez** [1] DMARTINEZ@IRI.UPC.EDU
**Guillem Alenyà** [1] GALENYA@IRI.UPC.EDU
**Tony Ribeiro** [2] TONY.RIBEIRO@LS2N.FR
**Katsumi Inoue** [3] INOUE@NII.AC.JP
**Carme Torras** [1] TORRAS@IRI.UPC.EDU

[1] *Institut de Robòtica i Informàtica Industrial (CSIC-UPC), Barcelona, Spain*
[2] *Laboratoire des sciences du numérique de Nantes (LS2N), Nantes, France*
[3] *National Institute of Informatics, Tokyo, Japan*

**Editor:** George Konidaris

## Abstract

Probabilistic planners have improved recently to the point that they can solve difficult tasks with complex and expressive models. In contrast, learners cannot tackle yet the expressive models that planners do, which forces complex models to be mostly handcrafted. We propose a new learning approach that can learn relational probabilistic models with both action effects and exogenous effects. The proposed learning approach combines a multi-valued variant of inductive logic programming for the generation of candidate models, with an optimization method to select the best set of planning operators to model a problem. We also show how to combine this learner with reinforcement learning algorithms to solve complete problems. Finally, experimental validation is provided that shows improvements over previous work in both simulation and a robotic task. The robotic task involves a dynamic scenario with several agents where a manipulator robot has to clear the tableware on a table. We show that the exogenous effects learned by our approach allowed the robot to clear the table in a more efficient way.

**Keywords:** Learning Models for Planning, Model-Based RL, Probabilistic Planning, Active Learning, Robot Learning

## 1. Introduction

Task planning has been a useful tool to solve complex problems where an agent has to reach a goal by executing actions, including robotic tasks (Kulick et al., 2013; Martínez et al., 2015a) and scheduling (Zhu et al., 2014). Task planners can find solutions that optimize a reward function for any state, even if this state was unexpected. The reward function encodes the priorities when solving the task, and new actions can be added easily to extend tasks.

Traditionally, applications have mostly used deterministic planners (Hoffmann and Nebel, 2001; Helmert, 2006) as they are easier to set-up and can solve complex problems with many variables. Recently the planning community has improved successfully the existing planners to solve more expressive and complex tasks. In the latest International Probabilistic Planning Competition (IPPC 2014) (Vallati et al., 2015), the best planners were able to solve

probabilistically interesting tasks (Little and Thiebaux, 2007) that included both action effects and exogenous effects (Kolobov et al., 2012; Keller and Eyerich, 2012) and many relational variables. We consider that action effects are those that occur when the agent executes an action (e.g. a robot moves to a new position, the robot grasps an object), while exogenous effects are those that do not depend on an action (e.g. people moving in the street, a traffic light turns red, a plant grows).

However, these planners rely on a model, that has to be either handcrafted or learned. In contrast to planners, learners cannot tackle yet the expressive models as planners do, which forces complex models to be mostly handcrafted. Relational action models with uncertain effects from a log of completely observable state-action-state transitions can be learned using the approach by Pasula et al. (2007), but in this paper we propose a new method that, in addition to relational probabilistic action models, it can also learn exogenous effects.

Relational model learners can be integrated in Reinforcement Learning (RL) approaches such as REX (Lang et al., 2012), allowing an agent to learn a task autonomously. However, heuristic model learners may have trouble learning certain domains if exploration is scarce when using REX. Thus we show that our approach can be integrated in V-MIN (Martínez, Alenyà, and Torras, 2015b), an extension of REX that can successfully learn such domains by actively requesting a few teacher demonstrations.

Finally, we will present how this learner can be integrated in a robot to improve its performance in tasks where exogenous effects are important. Specifically, we show a robot that has to help clearing the tableware on a table. Exogenous effects allow us to model the frequency at which different types of tableware arrive, and to coordinate with the waiter robots that take piles of tableware back to the kitchen. When a model with such exogenous effects can be learned, the performance of the task is increased significantly.

To summarize, we propose a novel method that takes as input a set of state-action-state transitions, and learns a relational model with probabilistic and exogenous effects to be used for planning. We also show that the learner can be integrated in a RL framework to learn and solve new tasks. This RL approach can be integrated in a robot decision-maker to improve the performance in the task of clearing the tableware of a table.

This work is an extension of Martínez et al. (2016), where the model learner was presented. The most significant additions are the integration with RL, the experiments with a robot, and showing the way input transitions must be normalized.

## 2. Background

In this section we present the formulation that we will use throughout this paper, as well as the background on reinforcement learning required to understand Section 4.

### 2.1 Formulation

Two types of representations are combined in this work, a relational one for the planning operators, and a propositional one that will be used internally by some parts of the learner. We assume complete observability and uncertain effects.

### 2.1.1 Relational Formulation

Literals $l$ are expressions of the form $(\neg)var(t_1, ..., t_m)$ where $var$ is a predicate symbol that represents a variable, $(\neg)$ means that the atom may be optionally negated, and $t_i$ are the terms. Terms can be variables, which have a preceding "?" symbol (e.g. ?X), and can also be objects, which are represented without an "?" symbol (e.g. box1). We use a relational representation where expressions take objects as arguments to define their grounded counterparts. A state $s$ is defined as a conjunction of grounded literals $l^g$ that follow the closed world assumption $s = l_1^g, ..., l_N^g$.

A planning operator $o \in \mathcal{O}$ defines the probability of a literal taking a value based on a set of preconditions. Operators take the form

$$o(t_1, ..., t_n) = l_h : p_o \leftarrow l_1 \wedge \cdots \wedge l_m, (a) \tag{1}$$

where $l_h$ is the head of the operator, $p_o$ is the probability of $l_h$ being true in the next state given that the body and the action are satisfied, $l_1 \wedge \cdots \wedge l_m$ are the literals in the body, $(a)$ is an optional action, and $t_i$ are the terms that may appear in the head, body and action. The action is optional so that operators can capture both action effects when there is an action, and exogenous effects when there is no action. Operators are not Horn clauses as negation can appear in both the body and the head. Also note that this representation does not model correlated effects, as each operator can only have one effect. There can be two operators with the same body and different heads, but they would be independent.

**Example 1** *An example operator for an action is:*
$o1(?X, ?Y, ?Z) = \textit{robot-at}(?X, ?Y) : 0.82 \leftarrow \textit{robot-at}(?X, ?Z) \wedge \textit{adj}(?Z, ?Y) \wedge \textit{move}(?X, ?Y)$.
*If the "move" action is applied, the robot would move to* $(?X, ?Y)$ *with a probability of* 0.82 *if it is on an adjacent position.*

A grounded operator only has objects as terms. If an operator $o$ has $n$ variables, its groundings $Gr(o)$ are a set of operators, each taking one of the possible combinations of $n$ objects.

**Example 2** *Having the objects* $\{x1, y1, y2\}$ *and the operator* $o1(?X, ?Y)$*, the possible groundings can be obtained by substituting* $?X$ *and* $?Y$ *for every permutation of 2 objects. One possible grounding is:*
$o1(x1, y2, y1) = \textit{robot-at}(x1, y2) : 0.82 \leftarrow \textit{robot-at}(x1, y1) \wedge \textit{adj}(y1, y2) \wedge \textit{move}(x1, y2)$.

The transition dynamics are defined by a set of planning operators $\mathcal{O}$. A grounded operator $o_g$ is said to cover a state-action pair $(s, a)$ when the literals of the body are in $s$, and the optional action of the operator is either $a$, or the operator has no action:

$$cov(o_g, s, a) = (body(o_g) \subset s) \wedge ((action(o_g) = a) \vee (action(o_g) = \varnothing)).$$

Likewise, a non-grounded operator $o$ covers $(s, a)$ if one of its groundings does:

$$cov(o, s, a) = \exists\, o_g \in Gr(o) \mid cov(o_g, s, a).$$

Transitions $T$ are defined as triples $t = (s, a, s')$ where $s'$ is the successor state of $s$ after executing $a$. A successor state $s'$ is obtained by applying all groundings of all operators to $(s, a)$. When a grounded operator $o_g$ is applied to $(s, a)$, its head is added to the state $s'$ with a probability $p_{o_g}$ if $cov(o_g, s, a)$.

**Example 3** *An example transition is:*
$s$*:* $adj(y1, y2) \wedge adj(x1, x2) \wedge$ *robot-at*$(x1, y1) \wedge$ *moving-obstacle*$(x1, y2)$
$a$*:* *move*$(x2, y1)$
$s'$*:* $adj(y1, y2) \wedge adj(x1, x2) \wedge$ *robot-at*$(x2, y1) \wedge$ *moving-obstacle*$(x2, y2)$
*Where the action "move" modifies the literal "robot-at", and an exogenous effect modifies "moving-obstacle".*

We require operators to be mutually exclusive, there cannot be two operators with the same head atom that cover the same $(s, a)$ as their heads may conflict. One example of such conflict would be $[o_{g,1}(r1) = at(r1) : 0.8 \leftarrow ...]$ and $[o_{g,2}(r1) = \neg at(r1) : 0.6 \leftarrow ...]$ where both heads cannot hold at the same time as one contradicts the other. The objective of this work is to learn models that can be used by planners, and planners require conflict-free operators. A planner has to know precisely the expected effects of applying a planning operator. If two different operators were to make conflicting changes, the effects would be undefined. This constraint is similar to the one in Pasula et al. (2007)'s learner, but in this case a different operator can be applied to each head.

If $s'$ is a successor state of $s$, we define $changes(s, s')$ as the set of literals $\{c \in s', c \notin s\}$. Given a transition $t = (s, a, s')$ and a grounded operator $o_g$, a transition change $c \in changes(s, s')$ has a likelihood

$$P(c \mid o_g) = \begin{cases} p_{o_g}, & cov(o_g, s, a) \wedge (c = head(o_g)) \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

And a set of non-grounded operators $\mathcal{O}$ gives the following likelihood to a change $c$:

$$P(c|\mathcal{O}) = \begin{cases} P(c|o_g), & \exists!\, o_g \in Gr(\mathcal{O}) \mid P(c|o_g) > 0 \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where $\exists!$ is the operator for uniqueness quantification. If more than one operator covers the same change given the same state-action pair, there is a conflict and the behavior is undefined, so a likelihood of 0 is given.

### 2.1.2 Propositional Formulation

On the propositional level, atoms are multi-valued variables that take the form $p^x$, where $p$ is a predicate symbol, $x$ is the value, and atoms have no parameters. A *state* is a conjunction of propositional atoms $s = var_1^{val_1} \wedge \cdots \wedge var_n^{val_n}$ such that $var^{val'}, var^{val''} \in s$ implies $val' = val''$.

We consider a *probabilistic multi-valued logic program* as a set of *propositional rules* $r$ of the form

$$r = var_0^{val_0} : p_r \leftarrow var_1^{val_1} \wedge \cdots \wedge var_n^{val_n} \tag{4}$$

where $var_i^{val_i}$ for all $i \geq 0$ are atoms and $p_r$ is a probability. A propositional rule $r$ is interpreted as follows: with probability $p_r$ the variable $var_0$ takes the value $val_0$ in the next state if all variables $var_i$ have the value $val_i$ in the current state. For any rule $r$, $var_0^{val_0}$ is called the *head* and $var_1^{val_1} \wedge \cdots \wedge var_n^{val_n}$ are the literals in the *body*.

**Example 4** *Let's consider the following rules,*
$r_1 = a^1 : 0.7 \leftarrow b^1$,
$r_2 = b^2 : 1.0 \leftarrow a^1 \wedge b^0$,
$r_3 = a^0 : 0.3 \leftarrow b^2$,
*then the logic program* $P = \{r_1, r_2, r_3\}$ *is a probabilistic multi-valued logic program.*

Finally, propositional transitions $E$ are defined as pairs of states $e = (s, s')$ where $s'$ is the successor state of $s$.

### 2.2 Reinforcement Learning

Reinforcement learning (RL) is a framework for sequential decision-making problems where the dynamics of the environment are not known in advance. At each time step, the agent takes an observation, decides which action to execute, and is given a reward. The agent's goal is to maximize the total sum of discounted rewards over time. In model-based RL the agent maintains a model that is updated with every transition, and a planner uses this model to select the action to be executed.

Formally, fully-observable problems with uncertainty are represented with Markov Decision Processes (MDP). A finite MDP is a five-tuple $\langle S, A, M, R, \gamma \rangle$ where $S$ is a set of states, $A$ is the set of actions that an agent can perform, $M : S \times A \times S \rightarrow [0, 1]$ is the transition model, $R : S \times A \rightarrow \mathbb{R}$ is the reward function and $\gamma \in [0, 1)$ is the discount factor.

The goal of RL algorithms is to find a policy $\pi : S \rightarrow A$ that chooses the actions to maximize the value function. The sum of expected rewards is the value function $V^\pi(s) = E[\sum_t \gamma^t R(s_t, a_t) | s_0 = s, \pi]$ which is to be maximized.

As the model is not known in advance, RL approaches have to balance exploration (visiting state-action pairs to learn the dynamics related to unknown parts of the model, thereby allowing better policies to be obtained in the future) and exploitation (executing actions to maximize rewards based on the current model).

## 3. Model Learner

In this section we show how to learn a relational probabilistic model with exogenous effects from a log of input state-action transitions. The learned model will consist of a set of planning operators that define both action effects and exogenous effects.

There exist previous works that tackle the problem of learning models. Some of them estimate the parameters of a model (Moldovan et al., 2012; Thon et al., 2011), but the structure of the model has to be provided and only its parameters are learned. The work by Jiménez et al. (2008) can capture the preconditions and uncertainty in the models given that the possible effects are known in advance. A more complete approach by Sykes et al. (2013) learns probabilistic logic programs, but the restrictions for the initial set of candidate rules need to be manually coded. In contrast, we learn the complete model and no restrictions are required.

Most approaches that learn complete models handle deterministic tasks, and although they can tackle partial observability (Molineaux and Aha, 2014; Mourão et al., 2012; Zhuo and Kambhampati, 2013) or apply transfer learning (Zhuo and Yang, 2014), they do not consider uncertain effects. In this work we focus on models with uncertain effects.

The most similar approaches to ours are those that learn relational action models with uncertain effects (Pasula et al., 2007; Deshpande et al., 2007; Mourão, 2014). They learn the effects that each action may have for each set of preconditions, which are then represented with probabilistic STRIPS-like models. However, they do not learn exogenous effects that are not related to any action.

These methods cannot be easily extended to learn exogenous effects. Pasula et al. (2007) and Deshpande et al. (2007)'s approach use a local search algorithm that works well when transitions are explained by one rule, but faces many local minima when tackling domains with exogenous effects, as two or more new rules may have to be added to properly explain a transition. Mourão (2014)'s approach exhibits a similar problem since it learns one rule per transition. To favor a better understanding of this problem, let's assume that we have a transition that is optimally modeled by 2 rules, one of which relates to an exogenous effect. These 2 rules are likely to have a very low score individually as they only cover part of the transition. The local search in Pasula's learner would choose a rule that covered the complete transition to maximize the score, even if the rule had a lot of noise and low probability to guess the effect. As a local search is used, the learner can only make one change at a time, and selecting only one of the 2 optimal rules would decrease the score, so they would never be chosen.

Note that although those methods cannot learn exogenous effects, they have other advantages. Pasula et al. (2007)'s approach can learn correlated effects while Mourão (2014)'s learner can also tackle partial observability. Therefore the best method will depend on the type of domain being learned.

The problem of learning minimal effects from a log of input data transitions is known to be NP-Hard (Walsh, 2010). The approaches shown before, as well as our approach, apply heuristics to find solutions with any number of input experiences. Optimal approaches that learn complete probabilistic models have also been proposed (Walsh et al., 2009), but they require too many input experiences or assumptions not made in the current work.

The learned model will consist of a set of planning operators that define the different effects. The proposed method can be divided into two parts:

- *Candidate planning operator generation.* Candidates are generated with the LFIT (Learning From Interpretation Transitions) framework (Inoue et al., 2014). LFIT induces a set of propositional rules that realize the given input transitions. Specifically, an algorithm that guarantees to learn the set of minimal rules is used (Ribeiro and Inoue, 2014).

- *Planning operator selection.* To select the best subset of candidates, we define a score function that is maximized by candidates that explain input transitions while being general enough. Based on this score function, a search optimization method guided by an heuristic function is proposed. Moreover, suboptimal solutions to make complex tasks tractable are provided.

Our approach combines (a) LFIT on the propositional level to ensure that candidates are minimal, (b) an optimization method that works on the relational level to apply relational generalizations when selecting subsets, and (c) grounded input data. Since, as mentioned, the approach requires three different types of data (grounded, relational and propositional), data transformation methods are needed.

### 3.1 LFIT

The LFIT framework (Inoue et al., 2014) is used to obtain the set of probabilistic candidate rules that model the dynamics. Given a batch of propositional transitions $(s, s')$, LFIT induces a *normal logic program* that realizes the given transitions. This framework has been extended (Ribeiro and Inoue, 2014) with a new algorithm that guarantees that the learned rules are minimal: the body of each rule constitutes a prime implicant to infer the head. It is based on a top-down method that generates hypotheses by *specialization* from the most general rules. Moreover, the framework has been adapted to capture also probabilistic dynamics (Martínez et al., 2015c).

In this section we explain briefly how to obtain probabilistic logic programs with minimal rules. For more details please refer to the work by Ribeiro and Inoue (2014) and Martínez et al. (2015c).

#### 3.1.1 Rule Specialization

To learn multi-valued logic programs with minimal rules we need to define the ground resolution and the least specialization for multi-valued propositional variables.

**Definition 1 (Subsumption)** *Let $r_1$ and $r_2$ be two rules. If $head(r_1) = head(r_2)$ and $body(r_1) \subseteq body(r_2)$ then $r_1$* subsumes *$r_2$. Let $P$ be a logic program and $r$ be a rule. $P$* subsumes *$r$ if there exists a rule $r' \in P$ that subsumes $r$.*

We say that a rule $r_1$ is *more general* than another rule $r_2$ if $r_1$ subsumes $r_2$ and $body(r_1) \subset body(r_2)$. In particular, a rule $r$ is *most general* if there is no rule $r'(\neq r)$ that subsumes $r$ ($body(r) = \emptyset$).

**Example 5** *Let $r_1$ and $r_2$ be the two following rules: $r_1 = (a^1 \leftarrow b^1)$, $r_2 = (a^1 \leftarrow a^0 \wedge b^1)$, $r_1$ subsumes $r_2$ because $(body(r_1) = \{b^1\}) \subset (body(r_2) = \{a^0, b^1\})$. When $r_1$ appears in a logic program $P$, $r_2$ is useless for $P$, because whenever $r_2$ can be applied, $r_1$ can be applied.*

**Definition 2 (Complement)** *Let $r_1$ and $r_2$ be two rules, $r_2$ is a* complement *of $r_1$ on $var^{val}$ if $var^{val} \in body(r_1)$, $var^{val'} \in body(r_2), val \neq val'$ and $(body(r_2) \setminus \{var^{val'}\}) \subseteq (body(r_1) \setminus \{var^{val}\})$.*

**Definition 3 (Ground resolution)** *Let $r$ be a rule, $P$ be a logic program and $\mathcal{B}$ be a set of atoms, $r$ can be* generalized *on $var^{val}$ if $\forall var^{val'} \in \mathcal{B}, val \neq val'$, $\exists r' \in P$ such that $r'$ is a complement of $r$ on $var^{val}$:*

$$generalise(r, P) = head(r) \leftarrow body(r) \setminus \{var^{val}\}$$

**Definition 4 (Least specialization)** *Let $r_1$ and $r_2$ be two rules such that $head(r_1) = head(r_2)$ and $r_1$ subsumes $r_2$. Let $\mathcal{B}$ be a set of atoms. The least specialization $ls(r_1, r_2, \mathcal{B})$ of $r_1$ over $r_2$ w.r.t $\mathcal{B}$ is*

$$ls(r_1, r_2, \mathcal{B}) = \{head(r_1) \leftarrow body(r_1) \wedge var^{val'}\} \text{ such that}$$
$$var^{val} \in body(r_2) \setminus body(r_1),\ var^{val'} \in \mathcal{B},\ val' \neq val$$

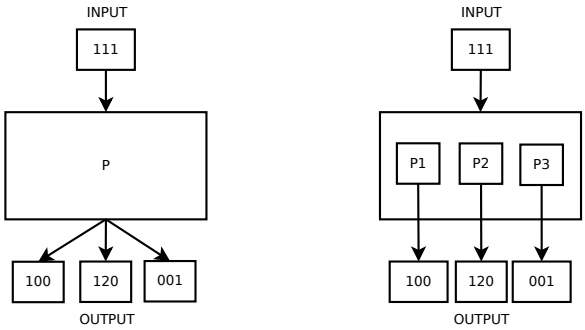


Figure 1: **Left:** A black-box probabilistic system where one program has multiple possible next states. **Right:** The LFIT algorithm uses a set of deterministic programs that form an equivalent probabilistic system. Each deterministic program outputs one of the possible next states.

Least specialization can be used on a rule $r$ to avoid the subsumption of another rule with a minimal reduction of the generality of $r$. By extension, least specialization can be used on the rules of a logic program $P$ to avoid the subsumption of a rule with a minimal reduction of the generality of $P$. Let $P$ be a logic program, $\mathcal{B}$ be a set of atoms, $r$ be a rule and $S$ be the set of all rules of $P$ that subsume $r$. The least specialization $ls(P, r, \mathcal{B})$ of $P$ by $r$ w.r.t $\mathcal{B}$ is as follows:

$$ls(P, r, \mathcal{B}) = (P \setminus S) \cup (\bigcup_{r_P \in S} ls(r_P, r, \mathcal{B}))$$

### 3.1.2 Probabilistic LFIT

To learn probabilistic systems, LFIT represent them as a set of deterministic programs. Given a state $s$, we compute the set of possible next states by querying each program. Figure 1 (left) shows a probabilistic system $P$ where the possible successors of the state 111 can be 100, 120 or 001. This behavior can be captured by a set of three deterministic programs $\{P1, P2, P3\}$ (right), such that $P1$ obtains a transition $e_1 = (\{a^1, b^1, c^1\}, \{a^1, b^1, c^1\})$, $P2$ obtains $e_2 = (\{a^1, b^1, c^1\}, \{a^1, b^2, c^0\})$, and $P3$ obtains $e_3 = (\{a^1, b^1, c^1\}, \{a^0, b^0, c^0\})$. Each program will also encode the behavior of the system for all possible other states. This representation allows us to capture any probabilistic system dynamics. It provides a model than can reproduce the possible behavior but no information about which transition is more likely to occur. To output such information, probabilities have to be extracted from observed transitions and reflected in the model. There are many ways of encoding likelihood and probabilities in logic programs, hence here we choose to simply consider rules likelihood independently for each rule. For each rule, we simply check how many times it could be applied in the input transitions and how many times the next state could effectively be the result of the use of this rule. This provides us with a simple likelihood for each rule, a ratio matches/realizes that can be used to give the likelihood of the possible next states. Other methods and normalization of the probabilities could be performed with the same sets of programs.

In this work we use an extension of *LFIT* (Martínez et al., 2015c) to learn a set of deterministic logic programs that can model probabilistic domains. The main idea is that when two transitions are not consistent, we need two different programs to realize them. The first program will realize the first transition and the second one will realize the second transition. The algorithm will output a set of logic programs such that every transition given as input is realized by at least one of those programs.

**Probabilistic LFIT**

- **Input**: a set of propositional transitions $E$ and a set of atoms $\mathcal{B}$ .
- Step 1: Initialize a set of logic programs $P$ with one program $P_1$ with fact rules for each atom of $\mathcal{B}$.
- Step 2: Pick $e = (s, s')$ in $E$, check consistency of $e$ with all programs of $P$:
- If there is no logic program in $P$ that realizes $e$ then
  - Copy one of the logic programs $P_i$ into a $P_i'$ and add rules in $P_i'$ to realize $e$.
  - Use ground resolution to generalize $P_i'$.
- Step 3: Revise all logic programs that realize $e$ by using least specialization.
- Step 4: If there is a remaining transition in $E$, go to step 2.
- Step 5: Compute the probability of each rule of all programs $P_i$ according to $E$.
- **Output**: $P$ a set of multi-valued logic programs that realize $E$.

The detailed pseudo-code of the *Probabilistic LFIT* is given in Algorithm 1. The algorithm starts with one logic program that contains all fact rules (lines 1-7). Each input transition $e$ is analyzed one by one. If no program can realize the observed transition (lines 10-20), one is copied and rules are added into this copy so that it realizes the transition (lines 21-28). The programs that realize $e$ are then revised using least specialization like in LFIT (Ribeiro and Inoue, 2014) (lines 29-36). The programs that do not realize the transition realize another one previously observed that is not consistent with the new one because of the non-determinism of the system. Those programs cannot be specialized by this transition because we would lose the information of the previous transition they realize. Finally, the likelihood of each rule is computed by just counting the number of observations covered/realized (lines 39-53). The algorithm outputs a set of logic programs, each one of them realizes some transitions of $E$, and all transitions of $E$ are realized by at least one program. The complexity belongs to $O(d \cdot |E| \cdot nv^n)$ for run time and $O(d \cdot 2^n)$ for memory, with $d$ the maximal out-degree of an observed state (degree of non-determinism), $E$ the set of input transitions, $n$ the number of variables and $v$ the maximal domain of a variable (the maximal number of values a variable may have).

Finally, all the learned programs $P$ are combined in one set that will become the set of planning operator candidates.

**Algorithm 1** Probabilistic LFIT($E, \mathcal{B}$)

**Input:** a set of pair of states $E$ and a set of atoms $\mathcal{B}$
**Output:** $P$ a set of logic programs

$E' := \emptyset$
$P := \emptyset$
// Initialize $P$: one program with the most general rules
$P_1 := \emptyset$
**for** each $var^{val} \in \mathcal{B}$ **do**
  $P_i := P_i \cup \{var^{val} \leftarrow\}$
**end for**
$P := P \cup P_1$
// 2) Revise $P$ to realize every transition
**while** $E \neq \emptyset$ **do**
  Pick $e = (s, s') \in E$; $E := E \setminus \{e\}$
  // 2.1) Check if $e = (s, s')$ is realizable
  **for** each logic program $P_i$ of $P$ **do**
    $realize\_e := true$
    **for** each $var^{val} \in s'$ **do**
      **if** $\nexists r \in P_i \mid body(r) \subseteq s$, $head(r) \in s'$ **then**
        realize_e := false
      **end if**
    **end for**
    **if** $realize\_e = true$ **then**
      **break**
    **end if**
  **end for**
  // 2.2) Construct a logic program that realizes $e$
  **if** $realize\_e = false$ **then**
    **for** each $var^{val} \in s'$ **do**
      $r := var^{val} : 1.0 \leftarrow \bigwedge_{B_i \in s} B_i$
      choose a $P_i \in P$
      $P'_i := P_i$
      $P :=$ **AddRule**$(P'_i, r, \mathcal{B})$
    **end for**
  **end if**
  // 3) revise logic programs that realize e
  **for** each logic program $P_i$ of $P$ **do**
    **for** each $var^{val} \in s'$ **do**
      **for** each $var^{val'} \in \mathcal{B}, val' \neq val$ **do**
        $r^s_{var^{val'}} := var^{val} : 1.0 \leftarrow \bigwedge_{m_i \in s} m_i$
        $P :=$ specialize($P$,$r^s_{var^{val'}}$)
      **end for**
    **end for**
  **end for**
  // 4) Remember e and continue
  $E' := E' \cup e$
**end while**
// 5) Compute the likelihood of each rule
**for** each logic program $P_i$ of $P$ **do**
  **for** each rule $r \in P_i$ **do**
    // $r = var^{val} : p_r \leftarrow body(r)$
    $i := 0$, $j := 0$
    **for** each $e = (s, s') \in E'$ **do**
      **if** $body(r) \subseteq s$ **then**
        $j := j + 1$
        **if** $head(r) \in s'$ **then**
          $i := i + 1$
        **end if**
      **end if**
    **end for**
    // Update the probability $p_r$ of the rule $r$
    $p_r = i/j$
  **end for**
**end for**
**return** $P$

### 3.2 Candidate Planning Operator Generation

The input of the method is a set $T$ of training transitions which are triples $t = (s, a, s')$ where $s'$ is a successor state of $s$ when the action $a$ was executed. The state $s'$ would have the effects of the action and all the applicable exogenous effects. The output is a set of planning operators $\mathcal{O}$ that define the model. Figure 2 shows the data transformations and processing done in the algorithm, which are described below:

- Transform grounded transitions to relational (non-grounded) ones. The objective is to learn relational operators that can take objects as arguments to generalize. Note that grounded literals are different from propositional ones because the objects and the variables in their terms can be identified. The transformation to relational transitions requires to substitute objects with variables.

- Obtain candidate operators. LFIT is used to obtain all possible candidate operators for a given set of transitions. The main advantage of using LFIT is that it obtains the set of minimal rules that model both action effects and exogenous effects. To use LFIT, first the relational transitions have to be transformed to propositional ones, and later, the output propositional rules to planning operators.

- Select the subset of candidate planning operators that best models the training transitions. This is detailed in Section 3.3.

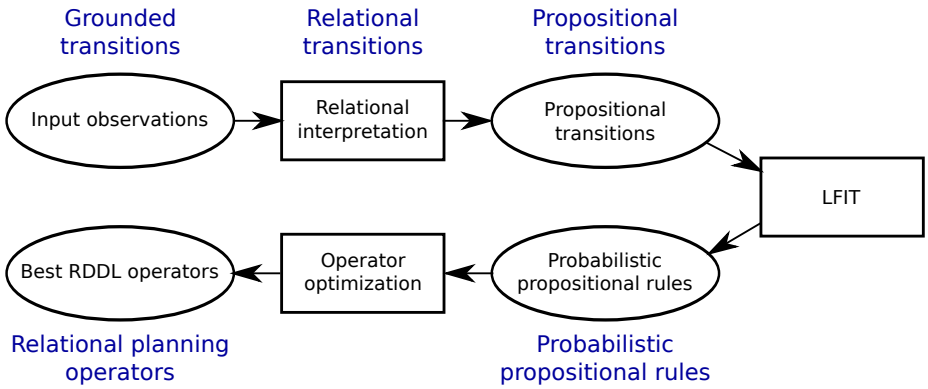


Figure 2: Data representation used for each module. The input and output data are shown in ellipses, and the processing modules are shown in boxes. The data representation used is indicated at each step.

The aim of these transformations is to generalize better, a relational representation is more compact and general as an infinite number of objects can be represented by each variable. By replacing objects with variables, LFIT learns rules that model relational data. The trade-off of learning relational generalizations is that the number of generated relational transitions is larger than the number of input grounded transitions, which increases the learning time.

#### 3.2.1 Input Normalization

The first step is to normalize the input so that all state-action pairs $(s, a)$ in the training data get the same number of transitions. Given a transition $t = (s_t, a_t, s'_t)$, the coverage of a pair $(s, a)$ by $t$ is defined as $cov(t, s, a) = (s_t = s) \wedge (a_t = a)$.

This step is specially important if the learner is integrated in a RL method, as state-action pairs with high expected values will be visited much often than those with low expected values. Without normalizing the input, when using heuristic learners, the selection of the best operators (Section 3.3) would be biased to perfectly model repeated state-action pairs and ignore rare ones.

The normalization is done by repeating transitions so that every $(s, a)$ gets a similar number of covering transitions:

- *Input*: grounded transitions $T$.
- Get maximum coverage $n_{max} = \max_{(s,a)} |\{t \mid cov(t, s, a),\ t \in T\}|$.
- For each $(s, a)$ covered by $T' \subset T$ (with $|T'| > 1$):
  - Number of times each transitions should be repeated: $n_{rep} = n_{max}/|T'|$.
  - Add $n_{rep}$ times each $t' \in T'$ to $T_{out}$.
- *Output*: normalized transitions $T_{out}$.

Note that this input normalization is used exclusively to calculate the likelihood of the sets of operators (Sec. 3.3), and it is not used for the RL exploration nor the confidence in the operators. Moreover, it does not have an impact on performance if implemented correctly. Every transition should have a counter representing the number of times it is repeated to avoid extra computation.

#### 3.2.2 Grounded to Relational Transitions

The goal is to obtain a relational representation that can generalize to different objects. If the dynamics of an object are learned, the same dynamics can be applied to other objects, not requiring examples of every possible grounding.

Since generating all possible relational combinations of every transition would be highly inefficient, we limit the number of relational variables to a fixed number $\omega$, which imposes a limit on the maximum number of variables that learned operators will have. This type of constraint has been frequent in previous works to tackle complex domains (Mourão et al., 2012; Walsh et al., 2009; Amir and Chang, 2008).

Selecting the right value of $\omega$ is important. To learn effects that involve $n$ objects, a value $\omega \geq n$ is required. However, the number of relational transitions scales exponentially with $\omega$, thus a large value of $\omega$ is intractable.

This module generates all possible relational transitions with at most $\omega$ variables. For every transition $t \in T$, the following method is applied:

- *Input*: grounded transition $t = (s, a, s')$, max variables $\omega$. We define the objects in $s$ as $b_{s,i}$ and the objects in $a$ as $b_{a,i}$. The action $a$ has $m$ objects.
- Initialize an empty set $V_{obj}$ of object combinations.
- Obtain combinations of $\omega$ objects. For each combination of $\omega - m$ objects $(b_{s,1}, \dots b_{s,\omega-m})$ that are not in the action $(b_{s,i} \neq b_{a,j}, \forall\, i, j)$ do:
  - Create $v_{obj} = (b_{a,1}, \dots b_{a,m}, b_{s,1}, \dots b_{s,\omega-m})$ where $(b_{a,1}, \dots b_{a,m})$ are the objects in the action $a$.
  - Add $v_{obj}$ to $V_{obj}$.
- For each $v_{obj} \in V_{obj}$:
  - A new transition $t' = t$ is created.
  - Replace in $t'$ all objects in $v_{obj}$ for variables.
  - Remove from $t'$ any remaining literal with objects.
  - Add the new transition $t'$ to $T'$.
- *Output*: a set of relational transitions $T'$.

**Example 6** *Given a grounded transition $(s, a \rightarrow s')$ :*

$$at(r1) \wedge road(r2, r3) \wedge road(r1, r3), move(r3) \rightarrow$$
$$road(r1, r3) \wedge road(r2, r3) \wedge at(r3),$$

*the following relational transitions are generated ($\omega$=2):*

$$\begin{aligned} v_{obj} = (r3, r1): \ & at(?Y) \wedge road(?Y, ?X), move(?X) \rightarrow \\ & road(?Y, ?X) \wedge at(?X); \\ v_{obj} = (r3, r2): \ & road(?Y, ?X), move(?X) \rightarrow \\ & road(?Y, ?X) \wedge at(?X). \end{aligned}$$

### 3.2.3 Relational to Propositional Transitions

To create the input that LFIT requires, which are pairs $(s, s')$ of propositional states, a library $L_{conv}$ that converts between relational and propositional atoms is created. For each relational atom $d_r$, a new propositional atom $d_p$ is created and the pair $(d_r, d_p)$ is added to $L_{conv}$. Using $L_{conv}$, everything is substituted by its propositional counterpart:

- Relational literals are represented with propositional atoms that take the values 1 (true) or 0 (false).

- Relational transitions are triples $(s, a, s')$, while propositional transitions are pairs $(s, s')$. Therefore, an additional multi-valued atom is added to propositional transitions to represent the action (if the agent executed an action during that transition). This atom takes as value the corresponding action name in $L_{conv}$, or "*noaction*" if there is no action. In the input, the "*noaction*" value will only be used in those transitions where the robot did not execute any action. In the output, it identifies exogenous effects. Note that literals are binary, so the multi-valued representation is only required to model the action, as there may be more than 2 different actions, and only one action can be executed per transition.

**Example 7** *Using the relational transitions in example 6, the following library is created* $L_{conv} = \{(at(?Y), b1), (at(?X), b2), (road(?Y, ?X), b3), (move(?X), b4)\}$. *The propositional transitions obtained by using* $L_{conv}$ *are:*

$$\begin{aligned} & (b1{=}1) \wedge (b3{=}1) \wedge (action{=}b4) \rightarrow (b3{=}1) \wedge (b2{=}1); \\ & (b3{=}1) \wedge (action{=}b4) \rightarrow (b3{=}1) \wedge (b2{=}1). \end{aligned}$$

### 3.2.4 Generation of Rules

The LFIT framework is used to obtain the set of probabilistic candidate rules from a set of proposition transitions $(s, s')$. LFIT induces a set of *normal logic programs* that realize the given transitions. Our approach uses the specialization and probabilistic algorithm of LFIT, so it learns the sets of minimal probabilistic rules that model all effects appearing in the input transitions. The learned sets are combined into a single set before converting them to planning operators. LFIT learns both action effects and exogenous effects because the action is just another atom that may or may not appear in the body of a rule.

**Example 8** *Using a larger set of propositional transitions such as the ones obtained previously, LFIT could obtain a set of rules such as:*

$$(b2{=}1) : 0.8 \leftarrow (b1{=}1) \wedge (b3{=}1) \wedge (action{=}b4);$$
$$(b2{=}1) : 0.1 \leftarrow (b1{=}1) \wedge (b3{=}0) \wedge (action{=}b4);$$
$$(b1{=}1) : 0.6 \leftarrow (b2{=}1);$$
$$\ldots$$

#### 3.2.5 Propositional Rules to Planning Operators

Planning operators (eq. 1) can be reconstructed from probabilistic rules (eq. 4) by using the library $L_{conv}$ created before. For each propositional rule:

- The atoms in the body and head of the rule are transformed to relational ones (using $L_{conv}$), and added to the body and head of a planning operator.
- The action is extracted from the multi-valued action atom in the rule body. If the atom is present, the corresponding action in $L_{conv}$ is added to the operator.

**Example 9** *Given that LFIT had learned the following rule*

$$(b2{=}1) : 0.8 \leftarrow (b1{=}1) \wedge (b3{=}1) \wedge (action{=}b4),$$

*using the library $L_{conv}$ generated in example 7, the resulting operator $o(?X, ?Y)$ is*

$$at(?X) : 0.8 \leftarrow at(?Y) \wedge road(?Y, ?X), move(?X).$$

Traditionally, PPDDL (Younes and Littman, 2004) has been the standard language to model probabilistic domains, but it cannot model exogenous effects directly (PPDDL1.0 rules model actions, not exogenous effects). Therefore, our approach uses the RDDL language (Sanner, 2010), which has been the standard for the latest probabilistic planning competitions (IPPC 2011 and 2014). Writing our planning operators with RDDL is straightforward, and this language can be used directly by state-of-the art planners. RDDL objects and variables have types, and a variable can only be substituted by an object of the same type. However, for clarity and simplicity, we assume through the paper that there are no types, as adding them is trivial.

### 3.3 Planning Operator Selection

LFIT provides the set of minimal rules (that have been transformed to planning operators) that describe all the transitions. Note that LFIT learns the set of minimal rules, and not the minimal set of rules, so many operators may model the same changes and underfit or overfit. The subset of operators to model the transition dynamics is selected as follows:

- A score function is defined to evaluate the planning operators.
- A heuristic search algorithm selects the set of operators that maximizes the score. Note that this set may differ from the actual model, as it depends on the coverage of the input transitions and the quality of the score function.
- The subsumption tree is used to improve efficiency by partitioning the set of candidates into smaller subsets.

### 3.3.1 Score Function

The score function values the quality of a set of operators. The following functions are used by the score function:

- The likelihood is the probability that a transition $t = (s, a, s')$ is covered by a set of planning operators $\mathcal{O}$:

$$P(t \mid \mathcal{O}) = \prod_{c \in \text{changes}(t)} P(c \mid \mathcal{O}, s, a). \tag{5}$$

- The penalty term $Pen(o) = |body(o)|$ is the number of atoms in the operator bodies.
- The confidence $Conf(T, \widehat{o}, \epsilon)$ is obtained from (Hoeffding, 1963)'s inequality. The probability that an estimate $\widehat{o_{prob}}$ is accurate enough, i.e. $|\widehat{o_{prob}} - o_{prob}| \leq \epsilon$, is bounded by $Conf(T, \widehat{o}, \epsilon) \leq 1 - e^{-2\epsilon^2 |T_{\widehat{o}}|}$, where $|T_{\widehat{o}}|$ is the number transitions covered by $\widehat{o}$.

Finally, the proposed score function is defined as

$$s(\mathcal{O}, T) = \underset{t \in T}{E}[\log(P(t|\mathcal{O}))] - \alpha \sum_{o \in \mathcal{O}} \frac{Pen(o)}{Conf(T, o, \epsilon)}, \tag{6}$$

where $\alpha > 0$ is a scaling parameter for the penalty term. This score function is based on Pasula et al. (2007)'s one, where the likelihood is maximized to obtain operators that explain the transitions well, and the penalty term is minimized to prefer general operators when specific ones have very limited contributions. In contrast to Pasula et al.'s approach, the confidence term is added so that the penalty is increased when few transitions are available, as the estimates are less reliable.

### 3.3.2 Heuristic Search

Given a set of operators $\mathcal{O}$ with the same head, a heuristic search method is used to find the best subset of operators that maximizes the score function. To that end, we define the heuristic version of the change likelihood (eq. 3) as:

$$P_h(c|\mathcal{O}) = \begin{cases} P(c|o_g), & \exists! o_g \in Gr(\mathcal{O}) \,|P(c|o_g) > 0 \\ 1 - \delta, & \nexists\, o_g \in Gr(\mathcal{O}) \,|cov(o_g, s, a) \\ 0, & \text{otherwise } (|Gr(\mathcal{O})| > 1), \end{cases} \tag{7}$$

where $\delta$ is a parameter that can trade quality for efficiency. This heuristic modifies the change likelihood (eq. 3) when no operator covers the change, giving a likelihood of $1 - \delta$ instead of 0. The heuristic score function $s_h(\mathcal{O}, T)$ is defined as the score function (eq. 6) but replacing the standard change likelihood (eq. 3) with this heuristic likelihood.

This heuristic gets the expected likelihood that can be obtained by adding new operators to $\mathcal{O}$. When $\delta = 0$, it works as an admissible heuristic (prop. 1) as it gives the maximum likelihood $= 1$ to uncovered changes. When $\delta > 0$ but close to 0, then the heuristic penalizes very specific operators when more general operators with a high likelihood are also

available. The practical result is that the algorithm usually runs faster, but the heuristic is not admissible anymore.

Algorithm 2 selects the best subset of operators to explain the input transitions. In line 1, the candidate list $\Gamma$ is initialized by creating one separate subset for each operator in the input set of candidates $\mathcal{O}_{input}$. Note that $\Gamma$ is a set of sets of planning operators, which is initialized to $\Gamma = \{\{o_1\}, \ldots, \{o_n\}\}$ assuming that $\mathcal{O}_{input} = \{o_1, \ldots, o_n\}$. Afterwards, lines 2-3 find the best subset in $\Gamma$ (which is the best set with only one operator). From that point, the candidate sets in $\Gamma$ are iteratively joined together to find the best set with any number of operators, until none has $s_h(\mathcal{O}, T) > max_{score}$ (lines 4-17). In lines 5-6, the candidate $\mathcal{O}$ with the largest heuristic score is selected and removed from $\Gamma$. Then, in lines 7-9, new candidates are generated by combining the selected subset $\mathcal{O}$ with every subset in $\Gamma$. The *IsNew* method checks that the new candidate has not been already analyzed. If any of the new candidates has a new best score, it is saved as the best candidate (lines 10-13). Finally, the new candidates are added to $\Gamma$.

This method works as a search algorithm guided by an heuristic. The nodes to be analyzed are the subsets of operators stored in $\Gamma$, where they are ordered by the heuristic score value. The search tree is expanded by joining one subset with every other subset. Finally, the algorithm continues until no subset has a heuristic score larger than the best score so that the solution has been found.

The search can be used as an *anytime* algorithm, it can be stopped at any point to get the best solution found so far. Moreover, there are two options to limit in advance the processing time of the algorithm in complex problems: set a time limit, or set a limit in the size of $\Gamma$ (only maintain the $\kappa$ sets with the highest score in $\Gamma$). Experimental tests showed that in some domains the heuristic leads quickly to the best set, and subsequent processing is done only to confirm that no other set is better.

**Property 1** *If $\delta = 0$, then the heuristic $s_h$ is admissible: $\forall \mathcal{O}, s_h(\mathcal{O}) \geq s(\mathcal{O}^*) \mid \mathcal{O}^* = \operatorname{argmax}_{\mathcal{O}' \supset \mathcal{O}} s(\mathcal{O}')$. Therefore the optimal set will be found.*

**Proof** The two parts of the score function in eq. 6 (likelihood and regularization) can be analyzed separately. Note that subsets of operators may only increase in size, as they start with one operator and can only be combined with other subsets.

- When adding operators to a set, the likelihood only increases when transition changes that were not covered before are covered by the added operators. In the heuristic score (eq. 7) all non-covered transition changes are already set to the maximum value of 1, so adding new operators cannot improve the result over the heuristic.

- The regularization part of the score function ($Reg(\mathcal{O}) = -\alpha \sum_{o \in \mathcal{O}} \frac{Pen(o)}{Conf(T,o,\epsilon)}$) is monotonically decreasing with respect to adding operators. If new operators are added, the new penalty $\frac{Pen(o)}{Conf(T,o,\epsilon)}$ will be positive as $Conf \in (0, 1]$, $Pen \geq 0$ and $\alpha > 0$.

■

**Algorithm 2** OperatorSelection($\mathcal{O}_{input}$, $T$)

Current candidates $\Gamma \leftarrow \{o\}, \forall\, o \in \mathcal{O}_{input}$
$max_{score} = \max_{\mathcal{O} \in \Gamma} s(\mathcal{O}, T)$
$\mathcal{O}_{best} = \operatorname{argmax}_{\mathcal{O} \in \Gamma} s(\mathcal{O}, T)$
**while** $\max_{\mathcal{O} \in \Gamma} s_h(\mathcal{O}, T) > max_{score}$ **do**
    $\mathcal{O} = \operatorname{argmax}_{\mathcal{O} \in \Gamma} s_h(\mathcal{O}, T)$
    Remove $\mathcal{O}$ from $\Gamma$
    **for** $\mathcal{O}' \in \Gamma$ **do**
        **if** `IsNew`$(\mathcal{O} \cup \mathcal{O}')$ **then**
            $\mathcal{O}_{new} = \mathcal{O} \cup \mathcal{O}'$
            **if** $s(\mathcal{O}_{new}, T) > max_{score}$ **then**
                $max_{score} = s(\mathcal{O}_{new}, T)$
                $\mathcal{O}_{best} = \mathcal{O}_{new}$
            **end if**
            Add $\mathcal{O}_{new}$ to $\Gamma$
        **end if**
    **end for**
**end while**
**Output** $\mathcal{O}_{best}$

**Property 2** *When relaxing the admissibility criterion with $\delta > 0$, the solution found by Algorithm 2 is bounded to be no worse than $\mathcal{C} \cdot \log(1-\delta)$ plus the optimal score, where $\mathcal{C}$ is the average number of literals with the same predicate that change in a transition.*

**Proof** Operators with different head predicates are analyzed separately as their effects are independent, so $\mathcal{C}$ only depends on the average changes to one predicate literals. Also note that $\delta \in [0, 1)$, and thus, $\log(1-\delta) \leq 0$. Let $\mathcal{O}_n$ be the optimal solution with $n$ operators and $s(\mathcal{O}_n) = opt$, then $\forall i < n$, at least one predecessor of $i$ operators $\mathcal{O}_i$ and $s_h(\mathcal{O}_i) \geq opt + \mathcal{C} \cdot \log(1-\delta)$ will exist.

- The regularization term is monotonically decreasing (see explanation of property 1), so $Reg(\mathcal{O}_n) \leq Reg(\mathcal{O}_i)$.
- The maximum difference between $P(\mathcal{O}_n)$ and $P_h(\mathcal{O}_i)$ is $P(t|\mathcal{O}_n) = 1$ and $P_h(t|\mathcal{O}_i) = (1-\delta)^x$, which is the case where $\mathcal{O}_n$ has perfect coverage, $\mathcal{O}_i$ has no coverage (all the coverage is obtained from $\mathcal{O}_n \setminus \mathcal{O}_i$) and the transition has $x$ changes. Then, if we take the worst case for all transitions, and an average of $\mathcal{C}$ changes per transition, $P(\mathcal{O}_n) - P_h(\mathcal{O}_i) = E[\log(P(T|\mathcal{O}_n))] - E[\log(P_h(T|\mathcal{O}_i))] = \log(1) - \log(1-\delta)^{\mathcal{C}} = \mathcal{C} \cdot \log(1-\delta)$.

Therefore, the predecessors of the optimal solution are checked (and thus the optimal solution found) unless a solution with $s(\mathcal{O}) \geq opt + \mathcal{C} \cdot log(1-\delta)$ is found before. ■

**Algorithm 3** OperatorSelectionSubsumption($Tree_{\mathcal{O}}, T$)

1: **do**
2: $\quad \mathcal{O}_L = leaves(Tree_{\mathcal{O}})$
3: $\quad \mathcal{O}_{L,best} = \texttt{OperatorSelection}(\mathcal{O}_L, T)$
4: $\quad$ Remove $(\mathcal{O}_L \setminus \mathcal{O}_{L,best})$ from $Tree_{\mathcal{O}}$
5: $\quad \mathcal{O}_P = \mathcal{O}_{L,best} \cup parents(Tree_{\mathcal{O}}, \mathcal{O}_{L,best})$
6: $\quad \mathcal{O}_{P,best} = \texttt{OperatorSelection}(\mathcal{O}_P, T)$
7: $\quad$ Remove $(\mathcal{O}_L \setminus \mathcal{O}_{P,best})$ from $Tree_{\mathcal{O}}$
8: **while** $Tree_{\mathcal{O}}$ changed
9: **Output** $= leaves(Tree_{\mathcal{O}})$

A value of $\delta > 0$ can speed up the algorithm considerably at the expense of optimality. When two operators have similar likelihoods, $\delta > 0$ penalizes the most specific one. This results in general operator sets being analyzed first, and thus models with better likelihoods are obtained earlier.

### 3.3.3 Subsumption Tree

In this section we present a method to speed up the approach by partitioning the set of candidates into smaller groups. The idea is to organize the operators in a tree using the OI-subsumption (Esposito et al., 2004), then start with the specialized operators, and iteratively check if more general operators yield better scores.

**Definition 5 (OI-subsumption relation)** *First, we define the extended body of an operator as* $body_{ext}(o) = body(o)(\wedge a)$ *where* $(\wedge a)$ *only appears if* $o$ *has an action. Let* $o_1$ *and* $o_2$ *be two planning operators with* $head(o_1) = head(o_2)$, $o_1$ *is OI-subsumed by* $o_2$ *if* $(body_{ext}(o_1) \supseteq body_{ext}(o_2))$, *given that different terms* $t_i$ *cannot take the same object.*

**Definition 6 (OI-subsumption tree)** *The OI-subsumption tree* $Tree_{\mathcal{O}}$ *of a set of planning operators* $\mathcal{O} = \{o_1, ..., o_n\}$ *is a directed graph with arcs* $(o_i, o_j)$ *when* $o_i$ *OI-subsumes* $o_j$ *and* $|body_{ext}(o_j)| - |body_{ext}(o_i)| = 1$. *We call the set of leaves* $L(Tree_{\mathcal{O}})$. *Figure 3 shows an example of a OI-subsumption tree.*

The OI-subsumption tree orders the operators in levels that represent the generality of the operators: the less literals the more general the operator. Based on this tree, Algorithm 3 selects the operators. The idea is to start by identifying the best specific operators, and then check if their generalizations improve the results. In lines 2-4, the best subset of leaves $\mathcal{O}_{L,best}$ is identified, and all leaves not in $\mathcal{O}_{L,best}$ are removed from the tree. Then, in lines 5-7, a new set of operators $\mathcal{O}_P$ is created that includes $\mathcal{O}_{L,best}$ and the operators that OI-subsume them (their parents in the tree). The best subset $\mathcal{O}_{P,best}$ in $\mathcal{O}_P$ is selected, and $\mathcal{O}_L \setminus \mathcal{O}_{P,best}$ are removed. This is repeated until nothing is changed in the tree.

The performance is improved by using the OI-subsumption tree: it divides the candidates into subsets, and the planning operator selection is much faster with smaller sets of candidates. Although it sacrifices optimality, experimental tests showed that the results obtained were in many cases optimal or near optimal. This happens due to the fact that

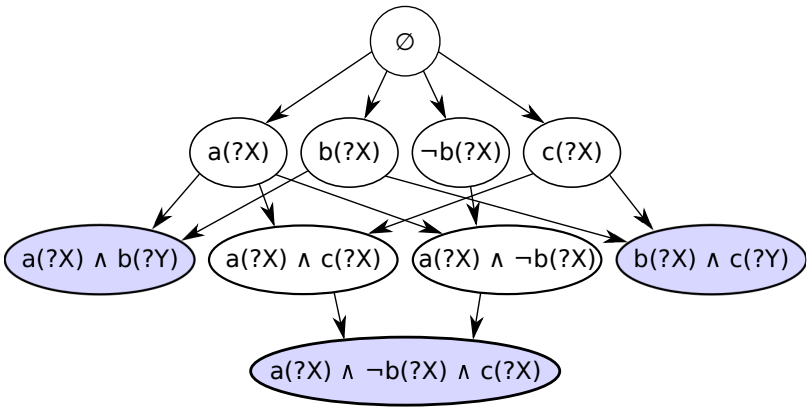


Figure 3: Example of a OI-subsumption tree. Each letter (a, b, c) represents a literal in the extended body of a planning operator. The leaves are the nodes painted in blue.

in most cases $P(\mathcal{O}) >> Reg(\mathcal{O})$. The operators in the leaves maximize $P(\mathcal{O})$ as they are more specialized, while the operators near the root maximize $Reg(\mathcal{O})$ as they are more general. Thus, the subset of leaves selected in the first iteration usually is near optimal, and afterwards the method only has to find the right level of generalization.

Note that if the OI-subsumption tree is used, the best operators may be close to the root of the tree (and thus they will be analyzed at the end), so the learner shouldn't be used as an anytime algorithm. However, the processing time can still be bounded with satisfactory results by limiting the size of $\Gamma$ to $\kappa$ sets in Algorithm 2.

## 4. Integration with RL

In this section we present how to integrate the model learner with a RL approach. The objective is to allow an agent to learn and solve a task from scratch by using a task planner and the learner presented before.

Model-based RL has proven to be an excellent tool to learn and solve a task that is unknown initially (Littman, 2015). There are several RL approaches that can learn complete relational models for planning (Džeroski et al., 2001; Diuk et al., 2008; Walsh et al., 2009), but they require a large amount of samples to make sure that they can solve the problem (Walsh, 2010). Other approaches use heuristics to reduce the number of samples needed to make the problem tractable, such as TEXPLORE (Hester and Stone, 2013) and REX (Lang et al., 2012). Specifically, REX uses Pasula et al. (2007)'s model learner internally, and proposes an exploration method based on R-MAX (Brafman and Tennenholtz, 2003) and $E^3$ (Kearns and Singh, 2002) that takes advantage of the relational representation to generalize and reduce the exploration required to learn relational tasks. However, REX can obtain models that are a local minimum as they cut exploration.

Our objective is to learn models without requiring too many input transitions. Therefore we choose to use the V-MIN algorithm (Martínez, Alenyà, and Torras, 2015b) as the RL framework. V-MIN extends the REX (Lang et al., 2012) with active learning: it explores

to learn the basic dynamics of the actions, but when it cannot find a plan with value larger than a certain threshold $V_{min}$, it requests help from a teacher. In contrast to REX, V-MIN can ensure that it will solve the task with a value larger than $V_{min}$ (given that the internal learner can obtain the model for the task). Moreover, it also performs well with scarce exploration, because even if it misses the exploration of an important state-action pair, V-MIN will visit it later (if it is needed to get $V_{min}$) by requesting demonstrations.

Many RL approaches focus on learning specific tasks, but have trouble when parts of the problem change. Model-based approaches have been proposed to learn models that generalize between different states and number of objects (Diuk et al., 2008), while others have applied transfer learning in combination with RL to reuse knowledge in similar tasks (Konidaris et al., 2012). The REX and V-MIN algorithms learn relational operators that can be used by standard planners, which provides a great flexibility in the changes that can be made to the task without having to relearn or adapt the model. Using Taylor and Stone (2009)'s transfer learning categorization, the following changes can be done:

- The initial state, the goal state, and the number of objects can be changed without requiring further learning.

- The reward function can change without having to modify the rest of the model.

V-MIN in addition can incorporate new actions easily to the model through demonstrations if they are needed to get a value $> V_{min}$. This results in an approach with good adaptation capabilities when compared to other works (Taylor and Stone, 2009), and it also has the advantage of learning models with less exploratory actions. The drawback is that a teacher is required.

The inclusion of a teacher in the learning loop has been also tackled by other approaches. In some methods the teacher issues corrective demonstrations when the robot doesn't perform as desired (Meriçli et al., 2012; Walsh et al., 2010), which implies that the teacher has to actively monitor the agent. The teacher has also been used to provide approval or disapproval of the agent performance (Knox and Stone, 2012). When using V-MIN, the agent is the one that actively requests help whenever it is needed, and thus the teacher is free from monitoring the agent. Other approaches also require help directly from the agent (Grollman and Jenkins, 2007; Chernova and Veloso, 2009), but they only learn from demonstrations, so they have to request a larger number of demonstrations to learn. Agostini et al. (2017)'s approach requests demonstrations from the teacher when the planner cannot find a solution with its current set of planning operators. However, this approach only works in goal-driven deterministic problems, and the interaction with the teacher lacks flexibility as it does not consider rewards. Finally, Walsh et al. (2011) proposed a similar approach to V-MIN where the agent communicates the expected value to the teacher, and if it is lower than the optimal value then the teacher demonstrates an action. This approach can learn STRIPS rules efficiently, but it would require many input experiences to learn exogenous effects in the same way because the number of possible rules would be much larger.

### 4.1 The V-MIN Algorithm

The REX algorithm (Lang et al., 2012) is devised to apply relational generalizations to R-MAX, reducing the exploration needed. State-action pairs are grouped in contexts that have the same dynamics (i.e. that are covered by the same relational planning operator). Based on these contexts, exploration is reduced by defining a context-based density function that considers all possible state-action pairs in a context to be equivalent when deciding if a state is known. Although exploration generalizations improve the learning performance, they also have drawbacks. Using a context-based count function implies that not all states are explored before considering them as known, as we assume that all states within a context behave likewise. Thus, state-action pairs needed to attain the best policy may not be visited and their contexts (i.e. planning operators) not learned.

The V-MIN algorithm requests teacher demonstrations to learn new actions and contexts, and improve performance. It uses the concept of $V_{min}$, which is the minimum expected value. If the planner cannot obtain a value $V^{\pi}(s) \geq V_{min}$, it actively requests help from a teacher, who will demonstrate the best action $a = \pi^*(s_i)$ for the current state $s_i$. Therefore, when exploration and exploitation can no longer yield the desired results, demonstrations are requested to learn yet unknown actions, or to obtain demonstrations of actions whose effects were unexpected (i.e. new unknown contexts). The result is that good models can be learned even if exploration is scarce. The $V_{min}$ parameter provides flexibility to the system:

- A high $V_{min}$ forces the system to learn good policies at the cost of a higher number of demonstrations, whereas a lower $V_{min}$ leads to a faster and easier learning process, but worse policies are learned.
- The teacher can change $V_{min}$ online until the system performs as desired, forcing it to find either better or easier policies.
- Demonstrations can be used to learn new actions that weren't required before.

#### 4.1.1 Algorithm Details

The V-MIN algorithm uses a model $M$ that consists of a set of planning operators $\mathcal{O}$, which define the contexts. The algorithm creates an extension $M_{vmin}$ of the model $M$ to implicitly plan exploration and demonstration requests. As in the R-MAX version of the REX algorithm, every unknown context (those that have been explored less than a threshold $\zeta$) gets the maximum reward $R_{max}$, implicitly following the principle of "optimism under uncertainty". Moreover, a special action "TeacherRequest()" is added which leads to an absorbing state with a $V_{min}$ value. "TeacherRequest()" is also limited to be the only action in a plan when selected, as otherwise a value of $V(s') + V_{min}$ could be obtained by combining it in a plan.

Algorithm 4 shows a pseudocode for V-MIN. This algorithm maintains a model $M$ that represents the robot actions with a set of relational planning operators. The model is updated in every iteration (line 3) to adapt to newer experiences using the learner presented in Section 3. The $M_{vmin}$ model is then created to plan the action to execute, implicitly selecting exploratory actions until all the relevant contexts become known, and requesting teacher demonstrations whenever a value $V^{\pi}(s) > V_{min}$ cannot be obtained. Finally a new state is observed, and a transition is added to the log of experiences.

**Algorithm 4** V-MIN

**Input:** Reward function $R$, confidence threshold $\zeta$
**Updates:** Set of input transitions $T$

Observe state $s_0$
**loop**
    Update transition model $M$ according to $T$
    Create $M_{vmin}(M, R, \zeta)$
    Plan an action $a_t$ using $M_{vmin}$
    **if** $a_t$ ==“TeacherRequest()” **then**
        Request demonstration
        $a_t$ = demonstrated action
    **else**
        Execute $a_t$
    **end if**
    Observe new state $s_{t+1}$
    Add $\{(s_t, a_t, s_{t+1})\}$ to $E$
**end loop**

The V-MIN algorithm provides the framework that selects which actions to explore and when a demonstration should be requested, but the most important piece is the learner used to obtain the model. Using the learner presented in Section 3 implies that a probabilistic relational model with exogenous effects will be learned.

Another important piece is the planner used, as it has to support planning with the learned model. PROST (Keller and Eyerich, 2012) is the planner used as it can obtain good results with probabilistic models containing exogenous effects.

Note that in this work we consider the reward function to be known. A KWIK algorithm can learn a reward function with a polynomial number of samples (Li et al., 2011), but it remains as future work to show how to effectively combine that reward learner with the V-MIN algorithm.

## 5. Experiments

This section describes the experimental evaluation of our approach. Three domains of the 2014 International Probabilistic Planning Competition (IPPC) (Vallati et al., 2015) were used in the experiments. The standalone learner introduced in Section 3 was used to learn from a batch of given input transitions, while the RL approach presented in Section 4 was used to solve tasks.

### 5.1 Domains

Three IPPC 2014 domains were used in the experiments. Note that they were slightly modified to remove redundancy (e.g. a *north(?X,?Y)* literal is equivalent to *south(?Y,?X)*, so one can be replaced by the other).

- *Triangle Tireworld.* This domain is the easiest one, it has uncertain effects, but no exogenous effects. It is modeled with 5 different predicates, 3 actions, 7 operators, and operators require at most 2 terms ($\omega = 2$). This domain serves as a good baseline to compare with the state of the art as there are not exogenous effects.

- *Crossing Traffic.* This domain has an intermediate difficulty. It has uncertain effects and exogenous effects, which makes it more challenging, but the complexity of the model is still moderate: 8 predicates, 4 actions, 6 operators, and operators take at most 3 terms.

- *Elevators.* This is the most challenging domain. It has uncertain effects and exogenous effects. It is modeled with 10 predicates, 4 actions, 17 operators, and operators take at most 3 terms.

### 5.2 Evaluation of the Model Learner

To evaluate the learner presented in Section 3 the scheme used by Pasula et al. (2007) was followed. The learners had to obtain models from sets of input transitions $(s, a, s') \in T$ that were generated randomly. To create a transition, first the state $s$ is constructed by randomly assigning a value (positive or negative) to every literal, but ensuring that the resulting state is valid (e.g. in the elevators domain, an elevator cannot be in two different floors at the same time). Then, the action $a$ arguments are picked randomly, and the state $s'$ is obtained by applying all operators to $(s, a)$. The distribution used to construct $s$ is biased to guarantee that, in at least half of the examples, the operators that contain $a$ have a chance of changing the state.

The evaluation of the learned models is carried out by calculating the *average variational distance* between the true model $\mathcal{O}$ and the estimate $\hat{\mathcal{O}}$. This evaluation uses a new set of similarly generated random transitions $T'$:

$$D(\mathcal{O}, \hat{\mathcal{O}}) = \frac{1}{|T'|} \sum_{t \in T'} \left| P(t|\mathcal{O}) - P(t|\hat{\mathcal{O}}) \right|. \tag{8}$$

As the *average variational distance* may be difficult to interpret, here we give an intuition about the utility of the learned models. A planner usually ($p > 0.9$) yields a plan that can solve the task (optimally or suboptimally) when the *average variational distance* is below:

- 0.09 in the *Triangle Tireworld* domain.

- 0.15 in the *Crossing Traffic* domain.

- 0.1 in the *Elevators* domain.

As the *average variational distance* becomes lower, the planner will obtain better solutions.

We analyze experimentally the proposed algorithm, its parameters, and how it compares with the state of the art. The learner uses the following parameters: $\alpha$, $\epsilon$, and $\delta$, which are the score function parameters; $\kappa$, which is the size limit of $\Gamma$ (Algorithm 2); and "*tree*" to denote that the subsumption tree is being used.

The difficulty to learn a domain is mostly given by:

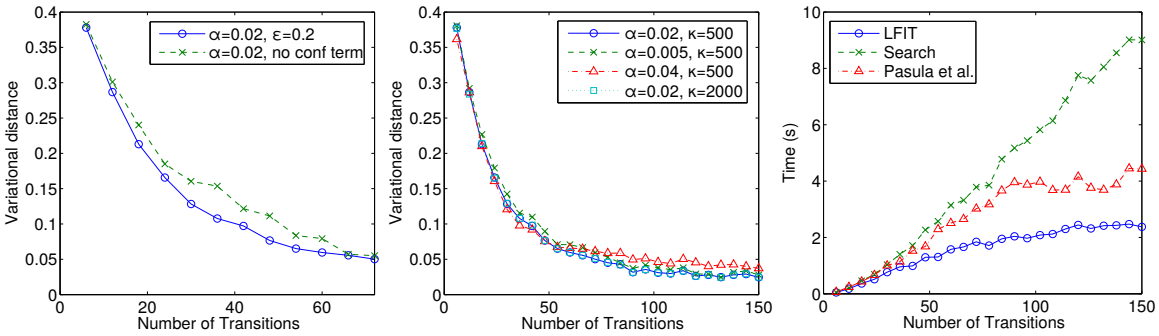


Figure 4: Evaluation of different configurations in the Triangle Tireworld domain, MDP-2. Unless stated otherwise, the following parameters were used: ($\alpha = 0.02$, $\epsilon = 0.2$, $\omega = 2$, $\delta = 0$, $\kappa = 500$, no-tree). The results shown are the means obtained from 50 runs. The evaluation was done with 3000 transitions. **Left:** Influence of the confidence term. **Center:** Comparing different values of $\alpha$ and $\kappa$. **Right:** Execution time of the search (optimization) and the LFIT modules. The total learning time would be the sum of LFIT plus one of the search configurations.

- The maximum number of terms $\omega$ that operators may have. The number of terms increases exponentially the number of relational transitions generated from the input grounded transitions (Section 3.2.2), and therefore the number of candidate rules. If the value of $\omega$ is larger than the number of terms that operators actually require, the learning time increases while the quality of the models remains the same.

- The number of predicates, both constant and variable, used to represent the states. The candidates that LFIT generates consider all combinations of predicates that are consistent with the transitions.

- The number of uncertain and exogenous effects. LFIT generates all candidates that may explain an effect, including operators that overfit and underfit, and all combinations of action effects and exogenous effects.

### 5.2.1 Configuration Parameters

Here we discuss the impact of the different configuration parameters on the quality of the models learned.

**Triangle Tireworld domain:**

- As seen in Fig. 4-left, the confidence term in the score function improves the quality of the models learned when few input transitions are available. This confidence term penalizes very specialized operators in cases where there is a large uncertainty in the predictions. Once many transitions are available, the impact of this term disappears. Note that only probabilistic operators are improved as deterministic ones are completely specialized anyway. The impact of this term is relatively small in the experiments because it only affects 1 probabilistic operator.

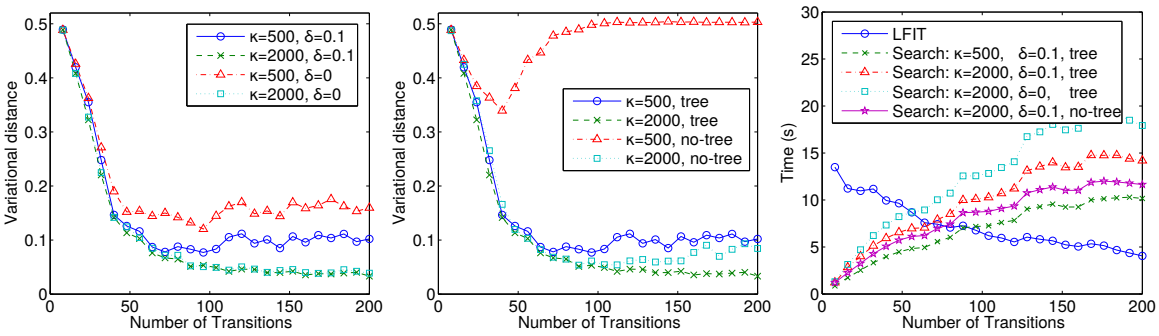


Figure 5: Evaluation of different configurations in the Crossing Traffic domain, MDP-1. Unless stated otherwise, the following parameters were used: ($\alpha = 0.02$, $\epsilon = 0.2$, $\omega = 3$, $\delta = 0.1$, $\kappa = 500$, tree). The results shown are the means obtained from 50 runs. The evaluation was done with 4000 transitions. **Left:** Influence of the $\kappa$ and the $\delta$ terms. **Center:** Influence of $\kappa$ and the subsumption tree. **Right:** Execution time of the search (optimization) and the LFIT modules. The total learning time would be the sum of LFIT plus one of the search configurations.

- Figure 4-middle compares different values of $\alpha$. Small changes of the $\alpha$ term produce almost the same results, and any value $\sim 0.02$ yielded good results in all the domains analyzed in this paper. Moreover, as the Triangle Tireworld domain is simple, a value of $\kappa = 500$ already produces good models, and increasing it does not improve the results.

- The execution time is shown in Fig. 4-right. Even with 150 input transitions only a few seconds are required to learn a model. Most of the execution time is spent in obtaining the likelihood of the operators as there are a lot of possible candidates in this domain.

**Crossing Traffic domain:**

- As can be seen in all the experiments, the number of operator candidates that the algorithm analyzes ($\kappa$) has a great influence on both the quality of the learned domain and the learning time. The best value of $\kappa$ will depend on the quality required for the domains and the time available for learning.

- Figure 5-left shows the advantages of a non-admissible heuristic ($\delta > 0$). A small value of $\delta$ prioritizes operators that explain many transitions with a high likelihood, and very specific operators that may not be interesting are given a smaller heuristic score. Although the same results could be obtained with a larger value of $\kappa$, the non-admissible heuristic speeds up the search (Fig. 5-right).

- Figure 5-center shows the advantages of the subsumption tree. This tree divides the search problem in smaller ones, and thus a much lower $\kappa$ is enough to learn good models, specially when the number of input transitions is large. Since to get the same model quality a much larger $\kappa$ would be required, the learning time is reduced.

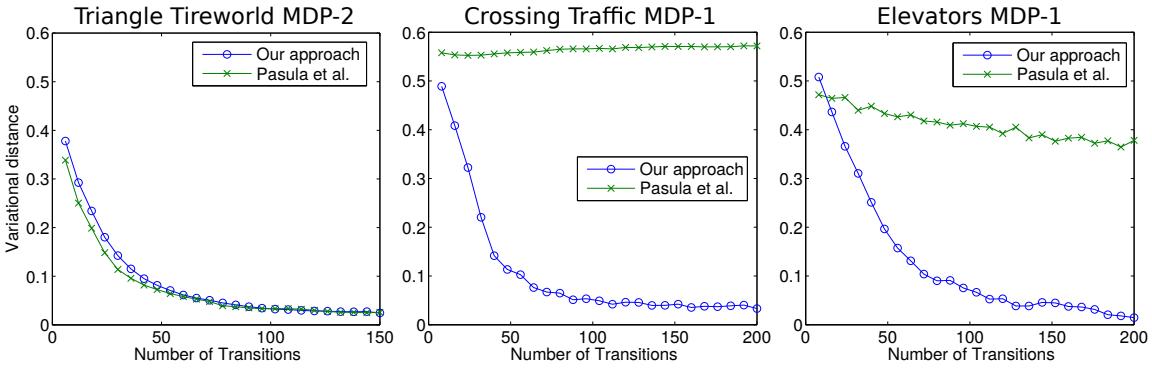


Figure 6: Comparison with Pasula et al. The results shown are the means obtained from 50 runs. The evaluation was done with 5000 random transitions. The parameters used were $\alpha = 0.02$, $\epsilon = 0.2$, $\delta = 0.1$, tree. **Left:** Triangle Tireworld ($\kappa = 500$). **Center:** Crossing Traffic ($\kappa = 2000$). **Right:** Elevators ($\kappa = 2000$).

### 5.2.2 Comparison with Pasula et al.

Comparison is performed only with Pasula et al. (2007)'s learner, as Deshpande et al. (2007)'s learner is an extension to include transfer learning, and Mourão (2014)'s learner yields similar results to Pasula et al.'s approach in completely observable problems. The experiments were carried out with the implementation by Lang and Toussaint (2010) and the fix by Mourão (2014).

Figure 6-left shows the results of learning the *Triangle Tireworld* domain. It can be easily learned by both, ours, and Pasula et al.'s approach, as there are no exogenous effects. With this experiment we want to show that our method can learn domains without exogenous effects as well as state-of-the-art learners. Pasula et al.'s learner gets slightly better results with few transitions as one of the actions is easier to learn with correlated effects.

Figures 6-center and 6-right refer to two domains with exogenous effects. As Pasula et al.'s learner cannot learn exogenous effects, it tries to build overcomplicated operators. These operators try to explain every possible combination of action effects and exogenous effects at the same time instead of learning each effect separately, and thus it is not able to yield a good general model. In contrast, our approach is able to distinguish action effects from exogenous effects once enough transitions are given as input.

### 5.2.3 Limitations of the Learner

The main limitation of the algorithm is the scalability. If the number of generated propositional predicates is large (either because the domain is represented with a large number of predicates, or because $\omega$ takes a high value), the problem may become intractable. The *Elevators* domain has the highest complexity that our approach can learn within a reasonable time. With 100 input transitions, the planning operator selection took an average of 2 minutes, and LFIT took up to 2 hours.

Moreover, in comparison with previous approaches, our learner cannot model correlated effects so it is not able to learn domains such as slippery gripper (Pasula et al., 2007). Our learner considers all effects as candidates, and thus, it would have to consider all possible

combinations of effects to learn correlated effects, which would be intractable. In contrast, approaches that do not learn exogenous effects can afford to find correlated effects because they search for a single effect per transition.

Finally, many IPPC 2014 domains cannot be directly learned. The following features would be needed to support all domains:

- The universal quantifier (*forall*) and the negative existential quantifier ($\sim$ *exists*) are not supported. When doing the translation from propositional rules to relational planning operators, new candidates could be generated that considered a positive atom in the preconditions as a (*forall*), or a negative atom as a ($\sim$ *exists*). It remains as future work to analyze the effectiveness of such candidates, and to check if the operator selection process would still be efficient with the addition of new candidates.

- Some IPPC domains have operators whose probabilities are encoded as a function. Unfortunately, our approach can only consider real numbers as probabilities.

### 5.3 Evaluation of the RL Integration

Here we evaluate how the learner can be integrated in a RL approach to learn models from scratch as explained in Section 4. The learner proposed by (Pasula et al., 2007) was also integrated in the V-MIN approach to compare against our approach.

#### 5.3.1 Triangle Tireworld

In this domain, a car has to move to its destination, but it has a probability of getting a flat tire while it moves. The car starts with no spare tires but it can pick them up in some locations. The actions available in this domain are: a "Move" action to go to an adjacent position, a "Change Tire" action to replace a flat tire with a spare tire, and a "Load Tire" action to load a spare tire into the car if there are any in the current location. The main difficulty in the *Triangle Tireworld* domain is the dead end when the agent gets a flat tire and no spare tires are available. Safe and long paths exist with spare tires, but the shortest paths do not have any spare tires.

It should be noted that in the IPPC 2014 representation of the *Triangle Tireworld* domain there is one exogenous effect: when the goal reward is received, the "goal-reward-received" literal becomes true. As this is modeled as an exogenous effect and Pasula et al.'s approach cannot learn it, that planning operator is given to it at the beginning. Our approach can learn this effect easily because it appears isolated after the goal is reached, so it doesn't have a significant impact on the performance.

Figure 7 shows the results of learning the *Triangle Tireworld* domain. Initially the robot chooses the shortest path and thus 50% of the times gets an irrecoverable flat tire. The learners have to learn that the robot cannot move with a flat tire, and that a spare tire is required to recover from a flat tire before the planner opts to go through the safe but longer path. MDP-4 is learned in less episodes because more actions are executed per episode. As expected, the performance of our learner is similar to Pasula et al.'s one, both learners can obtain the model of the task with few transitions so they also work well when integrated in V-MIN. Pasula et al.'s learner performs slightly better in the MDP-2 and our learner got slightly better results in the MDP-4, but the differences are not significant.

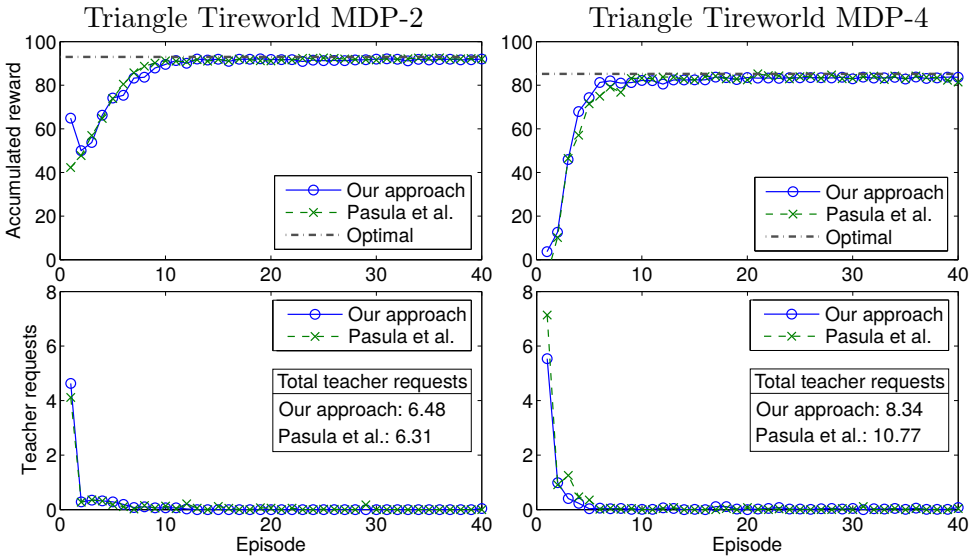


Figure 7: Learning the Triangle Tireworld domain with V-MIN. Comparison of using V-MIN with our learner and Pasula et al.'s learner. The difference between MDP-2 and MDP-4 is that in the latter the state space is larger, and more actions are required to reach the goal. The results shown are the means obtained from 100 runs. The exploration threshold is $\zeta = 3$ and $V_{min}$ is 85 in MDP-2, and 65 in MDP-4.

### 5.3.2 Crossing Traffic

The Crossing Traffic domain is a grid where a robot must get to a goal and avoid cars arriving randomly and moving left. The goal is located on the top right of the grid, and the robot starts on the bottom right position. If a car overlaps with the robot, the robot disappears and can no longer move around. The robot can "duck" underneath a car by deliberately moving right when a car is to the right of it. The robot receives $-1$ for every time step it has not reached the goal. The best strategy is to move first to the left to be able to see if a car is coming, and crossing whenever there are no cars in the way.

The difficulty in this domain is learning that if the robot ends in the same position as a car, it will disappear independently of the action executed. Our model learner has to see one transition of the disappear effect for every action until it can learn the disappear effect completely. Figure 8 shows the results of V-MIN and REX with our model learner. Pasula et al.'s learner was not compared here because, as mentioned previously, it cannot learn the exogenous effects in this task.

The main problem for REX in this domain is that some action effects can only be learned if the robot has moved previously and has not collided with a car. For example, the "move-right" action can only be learned if the robot moved first to the left (because it starts in the bottom-right corner) and if it did not collide before with any car. Thus, if the exploration threshold is low, REX may start to exploit with an incomplete model that provides suboptimal results.

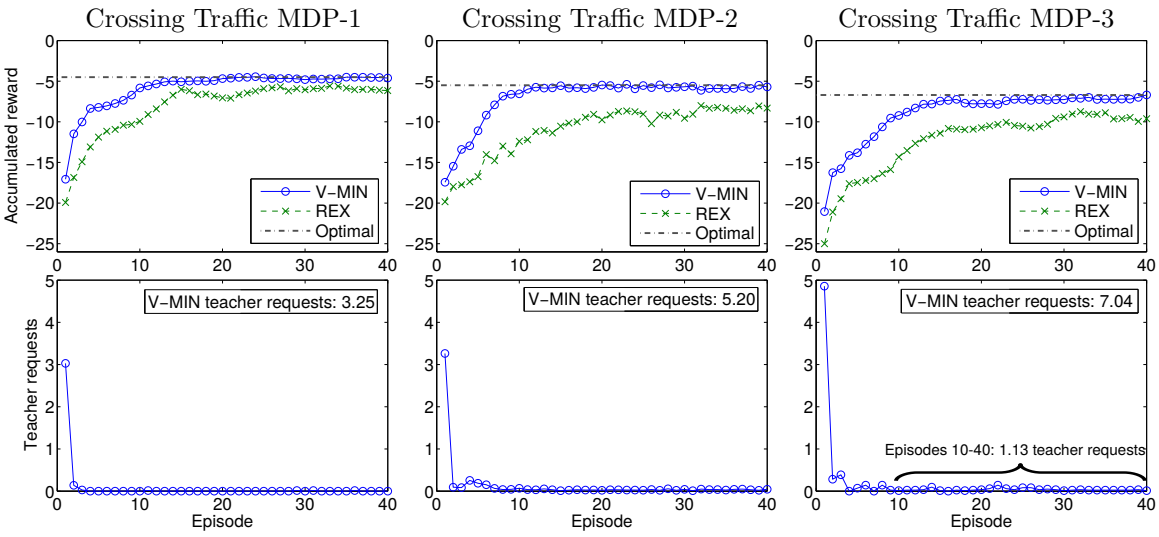


Figure 8: Learning the Crossing Traffic domain with V-MIN and REX. The results shown are the means obtained from 300 runs. MDP-1 is the easiest set up: its state space is small and the probability that a car arrives to each middle lane is 0.3. MDP-2 has also a small state space but cars arrive with a probability of 0.6. MDP-3 has a larger state space and a 0.3 probability of car arrivals. The exploration threshold is $\zeta = 3$ and $V_{min}$ is $-8$ in MDP-1, $-12$ in MDP-2, and $-15$ in MDP-3.

In contrast, V-MIN requests demonstrations when the models do not provide good enough policies, and the demonstrations quickly teach the missing action effects. Below we analyze each MDP separately:

- In MDP-1, the naive strategy of going directly to the goal has a high success ratio (70%), and thus it may take some episodes until the robot collides with a car. The robot always takes the shortest route until it learns that it disappears after a collision with a car independently of the action taken. This in an easy problem so V-MIN only requires a few teacher demonstrations during the first episodes to get good results. REX takes longer to get a high reward because it has to explore more to obtain a good model.

- In MDP-2, the success ratio of reaching the goal with the shortest route is only 40%. In this problem V-MIN requires a few extra demonstrations in episodes 2-6 if it could not learn the "move-right" action before. In contrast, REX failed to obtain good models in a few iterations because it had already collided or was in an edge every time it tried to explore an important action.

- MDP-3 works similarly to MDP-1, but as it has a larger state space, the probability of reaching the goal without dodging cars is around 50%. The larger state space also implies that new problematic situations may appear. For example, the robot may

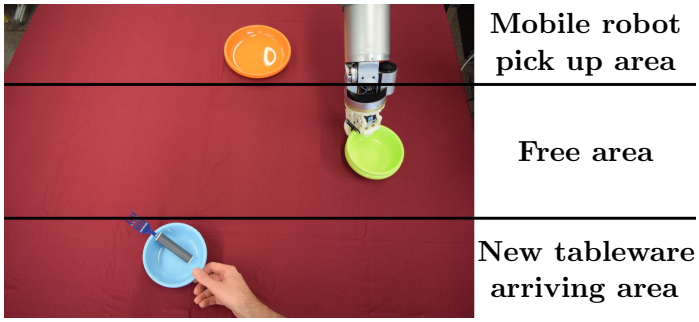


Figure 9: Table clearing task. On the bottom area, people leave the used tableware. On the top area, the robot has to prepare piles for the mobile robots that take tableware to the kitchen. The middle space is free for the robot to store objects while piling them up and waiting for mobile robots.

start to go up, and if too many cars appear so that the robot cannot dodge them, it has to go back down by executing a "move-south" action. The "move-south" action is not needed in MDPs 1 and 2, but it is needed in some cases in MDP-3. After the 10th episode, most of the teacher demonstrations requested by V-MIN are to learn how to solve these unexpected problems that didn't appear in the simpler problems. As REX cannot request teacher demonstrations, it has a harder time dealing with these unexpected situations, and this is reflected in a lower accumulated reward mean when compared to MDP-1.

## 6. Robot Table Clearing

This section describes a task where a robot has to clear the tableware laying on a table. To that end, V-MIN with the learner presented previously was used as the decision-maker of the robot to solve the task.

This task represents a robotized restaurant. The manipulator robot that we control has to clear the tableware on a table. It has to cooperate with mobile robots that can take a pile of tableware from the table to the kitchen to be cleaned. The number of mobile robots is limited, so our robot should pile tableware together to minimize the number of piles to be taken. The difficulty of the task comes from the fact that people will continuously bring new used tableware to the table. The robot has to organize continuously the tableware on the table so that there is always enough space for people to leave new tableware, and also to ensure that the mobile robots always have a prepared pile of tableware when they come to the table.

This task is illustrated in Fig. 9. People leave used tableware on the bottom area, the robot has the central area to organize tableware and create piles, and the top area is where the mobile robots pick a pile up when they come to the table. The tableware being used includes plates, cups and cutlery.

The robot system has three modules, the decision-maker module that includes the planner and the learner, the perception module that obtains a representation of the scene that can be used by the decision-maker, and the manipulation module that executes actions.

The perception module uses a camera that is located on top of the table (hanging from the ceiling) to update continuously a symbolic state that represents the table. This state consists of a set of literals that describe the different locations on the table. These literals are:

- "ArrivingLocation(?loc)" indicates that ?loc is a location where people will leave new tableware. This literal is constant as people always leave objects on the bottom side of the table.
- "PickUpLocation(?loc)" indicates that ?loc is a location where mobile robots will pick up piles to take them to the kitchen. This literal is constant as mobile robots always pick up piles from the top side of the table.
- "mobileRobotPickingUp(?loc)" indicates that a mobile robot will pick up a pile during the next iteration.
- "plate(?loc)" indicates that there is at least one plate at ?loc.
- "cup(?loc)" indicates that there is at least one cup at ?loc.
- "cutlery(?loc)" indicates that there is at least one fork, knife or spoon at ?loc.
- "stable(?loc)" indicates that the pile at ?loc is stable. An unstable pile cannot be picked up by a mobile robot.

The objects in this domain are the locations. The perception module generates the location objects dynamically by assigning one location to each pile of tableware, and then generates extra locations for (large enough) empty areas.

The manipulator module executes the actions planned. It takes the 3D perceptions obtained by the camera and generates the arm trajectories and gripper movements required to move the objects as desired. The symbolic actions that can be planned are:

- The "put(?loc1, ?loc2)" action to put the top object from a pile in location ?loc1 into location ?loc2.
- The "movePile(?loc1, ?loc2)" action that drags a whole pile from ?loc1 to ?loc2 if ?loc2 is empty. This action has high failure rates with unstable piles.

Piles may become unstable if objects are not piled properly. For example, if a plate is placed on top of a pile containing a cup and a fork, there is a high probability that it will become unstable. To obtain stable piles, in general, plates should be placed at the bottom, cups on top of them, and cutlery at the top. Unstable piles are harder to move, and mobile robots cannot pick them successfully.

Finally the decision-maker module uses the V-MIN algorithm (Section 4) in combination with the learner presented in Section 3 to learn, and the PROST planner (Keller and Eyerich, 2012) to plan.

Figures 10 and 11 show examples of the Table Clearing task. The robot plans the optimal action to execute based on the state of the table.

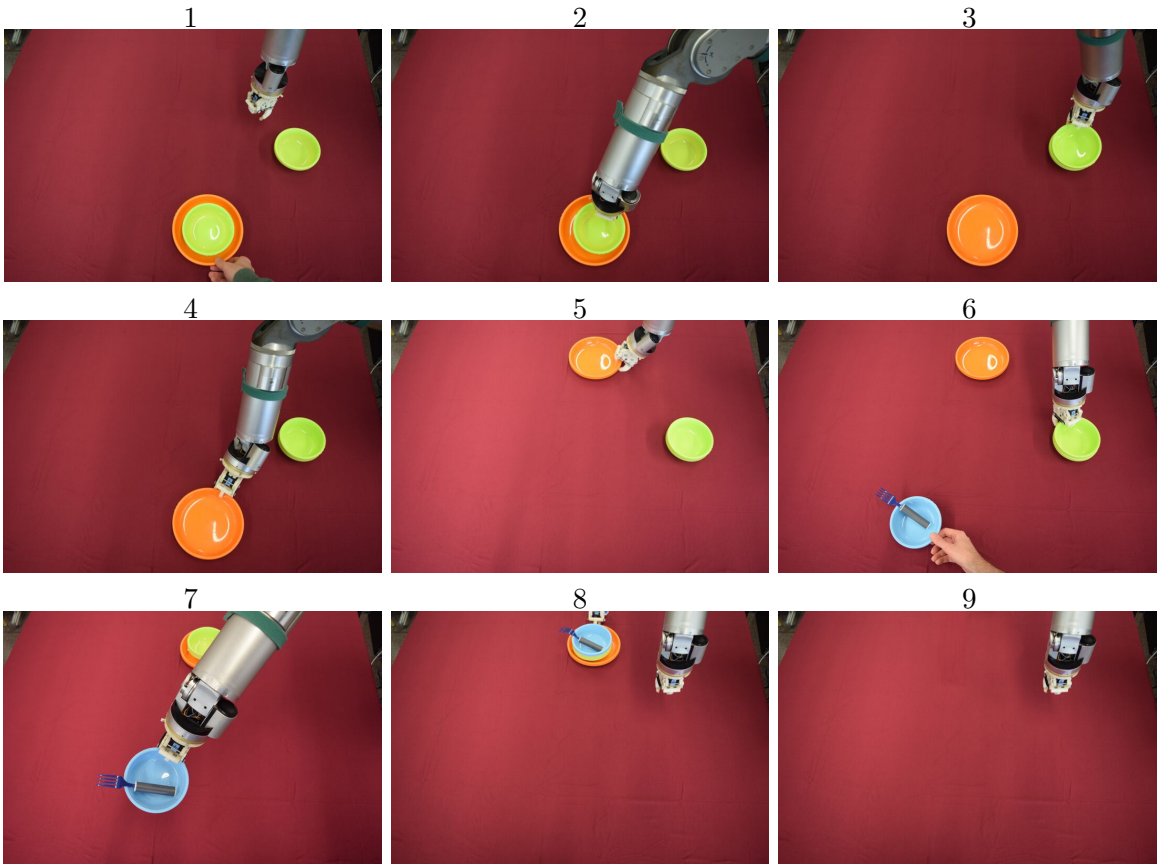


Figure 10: Table clearing with a learned model. As the robot does not know if new tableware will arrive shortly, it decides first to place the two cups together (images $1-3$), and then moves the plate to the *pick up* area (images $4-5$). Afterwards, a new pile arrives that contains no plates (image 6), so the robot decides to move the cups to the picking up area (image 7), and finally, as a mobile robot is arriving, the robot places the last pieces of tableware on the *pick up* area (image 8). Image 9 shows the final state after a mobile robot picked up the pile.

### 6.1 Learning a Model with RL

Using the setup described above, we executed the learner to obtain a model of the task from scratch while solving it. To make the experimentation easier, instead of having mobile robots picking up piles from the *pick up* area, a person did it. The robot had to solve experiments of increasing difficulty, and it transferred the learned knowledge from one experiment to the next. All experiments lasted 30 iterations during which piles were picked up 5 times from the *pick up* area.

1. The easiest experiment was executed once. People left 10 objects on the table during the experiment, including only plates and cups.

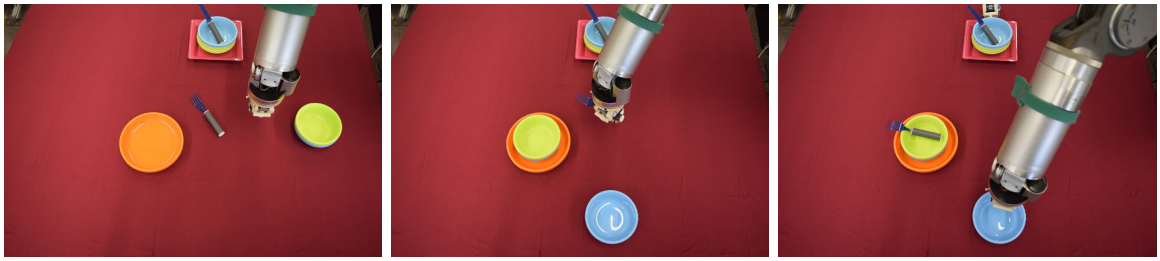

Figure 11: Table clearing with a learned model. **Initially**, the robot has already one pile prepared for the mobile robots. The planner opts to prepare a new pile with the tableware on the table as it expects that at some point a mobile robot will pick up the pile on top. **In the second image**, the new pile is halfway done, and a new cup arrives. The planner knows that in the next iteration a mobile robot is coming, so it decides to finish the new pile. **In the third image**, a mobile robot picks the top pile, and the manipulator robot plans that the best action is to move the new cup to the center free area to make more space in the arriving area. Once the mobile robot takes the top pile, the manipulator robot will plan the action of taking the center pile to the *pick up* area.

2. The medium experiment was executed twice. People left 13 objects on the table that included plates, cups and cutlery.

3. The difficult experiment was executed twice. People left 18 objects on the table that included plates, cups and cutlery.

The goals of the robot were to minimize the number of objects on the table, and to have empty space in the *arriving* area so that people always had space to place new tableware. Therefore big enough piles had to be given to the mobile robot at the same time that the arriving area had to be cleared quickly.

The learner parameters used were $\alpha = 0.01$, $\epsilon = 0.1$, $\omega = 2$, $\delta = 0.05$, $\kappa = 1000$, and the subsumption was enabled. The V-MIN exploration threshold was $\zeta = 3$ and $V_{min}$ was selected and updated by the teacher depending on the robot performance.

### 6.1.1 Learning Results

Table 1 summarizes the performance of the manipulator robot. During the easy experiment, 7 teacher demonstration requests were required to complete the task. The robot had to learn the "put" action to pick and place objects, and also the operators defining how mobile robots took piles from the *pick up* area.

In the second experiment, as cutlery appeared for the first time, and piles that have cutlery in the middle are usually unstable, the robot needed 6 extra demonstrations to learn how to manipulate cutlery and how unstable piles could be unpiled to make them stable again. The third experiment needed only 3 more demonstrations as the robot already knew most of the dynamics.

| | Easy 1 | Medium 1 | Medium 2 | Difficult 1 | Difficult 2 |
|---|---|---|---|---|---|
| Number of objects | 10* | 13 | 13 | 18 | 18 |
| Sum of reward | −48 | −90 | −82 | −148 | −130 |
| Optimal rewards | −41 | −64 | −65 | −97 | −101 |
| Teacher requests | 7 | 6 | 3 | 4 | 1 |

*In the easy setup no cutlery was used (only plates and cups).

Table 1: Learning results in the robot table clearing task. The columns show the different learning experiments, which were executed in order of increasing difficulty. The rows show the number of objects that arrived in each experiment, the sum of rewards, and the number of teacher demonstration requests.

Finally, in the last difficult experiments, 5 teacher demonstrations were needed once many objects accumulated on the table. The robot had to learn that it could assemble a pile in the middle area, and then move it directly to the *pick up* area by using the action "movePile(?loc1, ?loc2)". In previous experiments this action was not needed as less tableware arrived and the robot could just make piles directly on the *pick up* area with the "put" action.

Appendix A explains in more detail the execution of the learning process as well as the behavior of the manipulator robot during the easy experiment and the first medium experiment.

It should be noted that during the learning experiments we observed that the effectiveness of the robot actions was very important. If actions failed often, many more action executions were needed to obtain a proper model. Therefore, before running the experiments presented here, the robot actions were improved so that they succeeded in most cases to make the task simpler.

### 6.2 Evaluation of the Learned Model

Finally, an experiment was carried out to evaluate the usefulness of learning exogenous effects. Here we compare the performance when using the model learned in the previous section, with another model learned with no exogenous effects (Pasula et al., 2007). The exogenous effects allow the planner to know the probabilities with which each type of tableware arrives, and the probability with which mobile robots come to pick piles up. There was a reward of −1 for every object on the table at each step, and a reward of −2 if new tableware was arriving but the arriving area was full. The results are shown in Table 2.

- An optimal action sequence obtained a reward of −99. Note that this optimal action sequence was not obtained in a fair way: it was created knowing in advance when and which type of tableware was arriving at each step, and when a mobile robot approached. In contrast the manipulator robot did not have this information, it could only know the probabilities of these effects.

- The model containing exogenous effects obtained a reward of −119. In this experiment the manipulator robot performed well in general, but it created some inefficient piles

| | Optimal | Model with exogenous | Model with no exogenous |
|---|---|---|---|
| Sum of rewards | −99 | −119 | −136 |
| Operators (actions) | - | 8 | 20 |
| Operators (exogenous) | - | 11 | 0 |

Table 2: Execution results with the learned models. The columns show the results with an optimal sequence of actions, a learned model considering exogenous effects, and a learned model not considering exogenous effects. The rows show the sum of rewards and the number of planning operators that represent actions and exogenous effects.

as a mobile robot would take shorter or longer than expected to arrive. Moreover, the manipulator robot took a conservative strategy and it didn't complete fast enough some piles, as it did not know if new tableware would be arriving shortly. The model consisted of 8 planning operators for action effects, and 11 planning operators for exogenous effects.

- The model without exogenous effects obtained a reward of −136. This model had more trouble to complete the task. The manipulator robot did not know that new tableware would arrive because it did not have the exogenous effect for it. Thus, it tried to make the best piles with the given objects, which resulted in the robot redoing the piles whenever new tableware arrived (e.g. if a new cup arrived, the robot would put it on top of the other cups in an existing pile, and below the cutlery). The model had 20 operators that represented action effects. As exogenous effects were not learned separately, each operator actually represented combinations of an action effect with exogenous effects. However, these operators were less effective to represent the model than separate action and exogenous operators, so the planner selected worse policies.

Analyzing the results we can see that the proposed learner allows a robot to learn tasks including exogenous effects. Given that only 5 episodes of 30 action executions were used for training, the results were good. The experiment also shows that a model considering exogenous effects improves the performance when external agents interact. Here the planner was able to find much better plans when it anticipated that new tableware or a mobile robot were arriving. However, extending this task with a richer representation and more external agents would be costly. For every literal and exogenous effect that gets added, the number of input transitions and computational time required to learn a model increases significantly.

## 7. Conclusions

We have introduced a new method that, given a set of input transitions, learns a general model explaining them. In contrast to previous approaches, it can learn exogenous effects (effects not related to any action), while still being similarly good at obtaining a relational representation of the problem and at learning uncertain effects. Moreover optimal and suboptimal search methods are provided, so the best approach can be chosen depending on

the quality requirements, the difficulty of the problem, and the learning time available. The main limitation of the algorithm is scalability when the number of generated propositional predicates becomes large.

This learner can be combined with a RL algorithm to learn models from scratch. We validated experimentally that the integration in the V-MIN algorithm allowed an agent to learn models with a relatively small number of actions and teacher demonstration requests.

The learner was also integrated in a robot to perform the task of clearing the tableware on a table. In this task external agents interacted, people brought new tableware continuously and the manipulator robot had to cooperate with mobile robots to take the tableware to the kitchen. The learner was able to learn a usable model in just 5 episodes of 30 action executions. Finally, the model was used to complete the task, and learning models with such exogenous effects proved to increase the obtained reward significantly.

Many robotic problems cannot be learned yet. As future work, the learner could be extended to tackle correlated effects and partial observations.

- The challenge with correlated effects is to identify the candidates. A possible path to explore is to replace LFIT by Ribeiro et al. (2015)'s approach, in which preconditions can be literals in both the previous state ($s_t$) and the resulting state ($s_{t+1}$). In this case, an effect (a literal in $s_{t+1}$) can be a precondition for another effect, so they would be correlated effects.

- Partial observations would require to change the heuristic search and the cost function to consider that the given state transitions may be wrong or incomplete.

## Acknowledgments

This work has been supported by the MINECO project RobInstruct TIN2014-58178-R and the European Union's Horizon 2020 research and innovation programme under grant agreement H2020-ICT-2016-1-731761 IMAGINE. D. Martínez is also supported by the Spanish Ministry of Education, Culture and Sport via a FPU doctoral grant (FPU12-04173).

## Appendix A. Learning the Table Clearing Task

In Sec. 6.1 we explained how the manipulator robot learned the table clearing task. This appendix shows the detailed execution of the easy experiment and the first medium experiment to provide a deeper understanding of the learning process.

### A.1 Easy Experiment

The states at each iteration of the easy experiment are shown in Fig. 12. The robot starts with no prior knowledge and has to learn the complete model. Below is an explanation of the actions taking place at each iteration:

1. Initially the robot does not have prior knowledge and its model is empty, so no action is taken. A person brings a plate (exogenous effect).

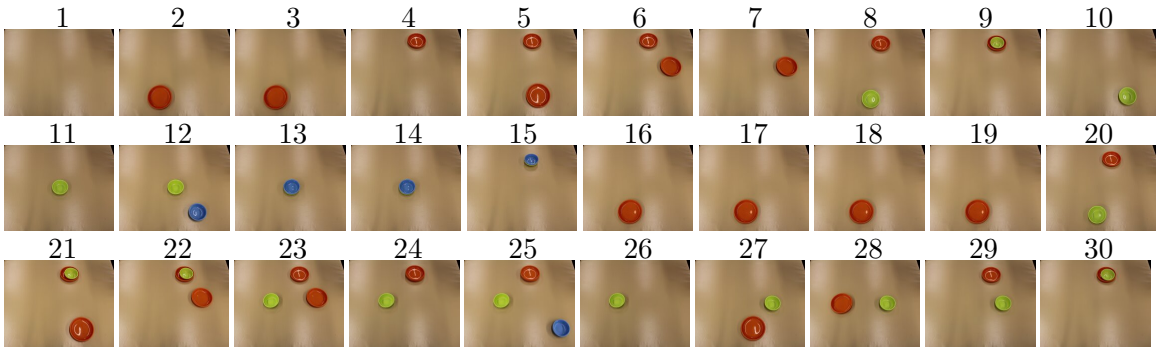


Figure 12: The state at the beginning of each iteration during the easy experiment.

2. No action is taken.

3. The manipulator robot learns that plates remain in the same location if no action is taken. A demonstration is requested to learn how to deal with plates, and the teacher demonstrates the "put" action to move the plate to the *pick up* area.

4. The manipulator robot executes the "put" action with an empty location as parameter, so it does not move anything. A new plate arrives.

5. The new plate is moved to the middle of the table.

6. The manipulator robot executes "put" with an empty location. Meanwhile, a mobile robot comes and picks the plate on the *pick up* area (another type of exogenous effect).

7. After learning that mobile robots pick up plates on the *pick up* area, the manipulator robot moves the plate on the middle to the *pick up* area. A cup arrives to the table.

8. A demonstration is requested, and the teacher moves it to the *pick up* area with a "put" action.

9. It is not known yet that mobile robots also pick cups up, so a demonstration is requested. The teacher does not execute any action because a mobile robot is already arriving to take the pile. A new cup appears.

10. The learned model is not good enough yet, so a demonstration is requested. The teacher puts the cup in the center of the table.

11. An exploratory "put" action with an empty location is taken. A new cup arrives.

12. A demonstration is requested. The teacher shows that both cups can be piled together.

13. No action is taken.

14. As a mobile robot is arriving, the cups are moved to the *pick up* area.

15. The mobile robot takes the cups and a new plate appears.

16. The robot executes a "put" action that fails.
17. An exploratory "put" action is executed with an empty location (a new planning operator with the action "put" and the precondition "cup(?X)" was generated and it is explored).
18. An exploratory "put" action is executed with an empty location.
19. The plate is moved to the *pick up* area. A cup appears.
20. The new cup is moved to the *pick up* area. A new plate appears.
21. A demonstration is requested (the robot does not know yet that it can pile plates together). The teacher decides to put the new plate on an empty location because piling it on the *pick up* area could be unstable.
22. As an exploratory action, the cup is moved to an empty location.
23. A demonstration is requested, and the teacher piles the two plates together.
24. No action was taken (exploration is not needed, so it decides to wait until new objects arrive or a mobile robot approaches). A new cup arrives.
25. The new cup is moved to the pile in the *pick up* area because a mobile robot is arriving.
26. As an exploration action, the cup is moved to a different location on the center of the table. A plate arrives.
27. The plate is moved to the center.
28. The plate is moved to the *pick up* area.
29. The cup is put on top of the plate.
30. A mobile robot comes and takes the last pile.

### A.2 First Medium Experiment

Here the robot reuses the knowledge that it obtained during the easy experiment. The states at each iteration of the first medium experiment are shown in Fig. 13. Below is an explanation of the actions taking place at each iteration:

1. Initially there is no tableware, so no action is taken. A plate arrives.
2. The plate is "put" on the *pick up* area. A cup arrives.
3. An exploratory "put" action that does nothing is executed (the first parameter is an empty location and the second one is the cup, so it does not move anything).
4. The operators learned are not complete enough, so a demonstration is requested. The teacher places the cup in the middle of the table. A plate arrives

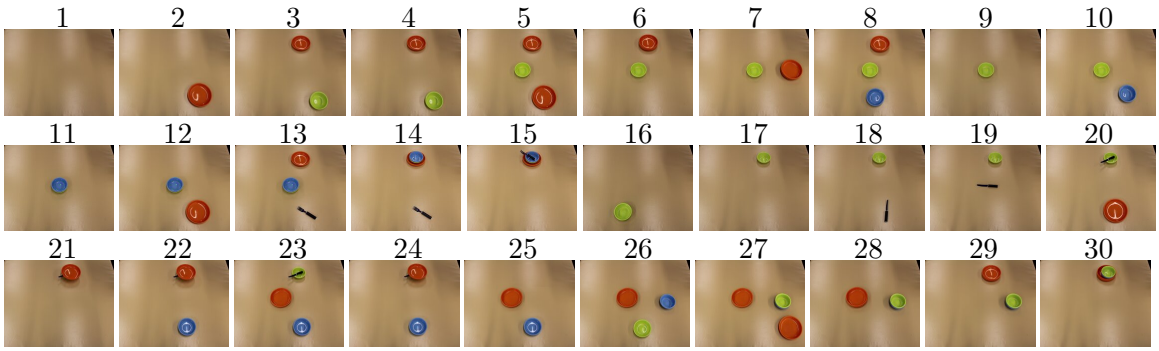


Figure 13: The state at the beginning of each iteration during the first medium experiment.

5. The new plate is moved to the *pick up* area.
6. As an exploratory action, the plates are moved to the center of the table.
7. The plates are moved back to the *pick up* area. A cup arrives.
8. As a mobile robot is arriving, it is piled up directly in the *pick up* area. The pile on the *pick up* area is taken.
9. No action is taken (exploration is not needed, so it decides to wait until new objects arrive or a mobile robot approaches). A new cup arrives
10. The new cup is piled with the other one.
11. No action is taken and a plate arrives.
12. The new plate is moved to the *pick up* area.
13. Cutlery appears for the first time. A demonstration is requested and the teacher decides to move the cups to the *pick up* area. Note that the teacher shows the best action, and not the action that would make the robot learn the most. In this case it is better to first put the cup on the plate, and later put the cutlery on the cup.
14. As the actions to manipulate cutlery have not been learned yet, another demonstration is requested. This time the teacher executes a "put" action to pile up the fork.
15. The manipulator robot has still to learn that mobile robots also take cutlery, so another demonstration is requested. As a mobile robot is arriving to take the pile, the teacher does not execute any action. A cup arrives.
16. The new cup is moved to the *pick up* area.
17. No action is taken. A fork arrives.
18. The new fork is moved to the center of the table.
19. The fork is put on top of the cup at the *pick up* area. A plate arrives.

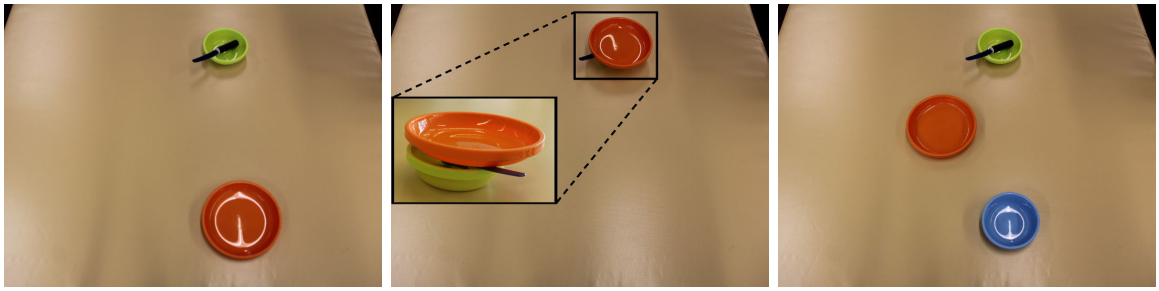

Figure 14: Unstable pile during the first medium experiment. **Left:** the state before the manipulator robot placed the plate on the top pile. **Center:** the pile became unstable, and thus it was not taken by mobile robots. **Right:** to make the pile stable again, the plate was positioned back in the table.

20. The new plate is moved to the pile at the *pick up* area. However, the pile becomes unstable as a plate is on top of a cup with a knife (see Fig. 14).

21. A mobile robot comes, but it does not take the pile because it is unstable. A cup arrives.

22. The manipulator robot has learned that mobile robots do not take unstable piles, so a demonstration is requested. The teacher shows that putting the plate back to the table makes the pile stable again.

23. As an exploratory action, the plate is put again on the pile, making it unstable.

24. The manipulator robot makes the pile stable again by removing the plate. A mobile robot comes and takes the pile on the *pick up* area as it is stable.

25. The cup on the *arriving* area is moved to the center of the table. A new cup arrives.

26. The model has updated its probabilities with which new objects and mobile robots arrive, and the resulting plan has a value $V(s) < V_{min}$ even though it is a good plan. The problem is that $V_{min}$ was too high, and the teacher updates it after piling the two cups together. A plate arrives.

27. The plate is piled on top of the other plate.

28. The plates are moved to the *pick up* area.

29. The cups are piled on top of the plates.

30. A mobile robot comes and takes the last pile.

## References

Alejandro Agostini, Carme Torras, and Florentin Wörgötter. Efficient interactive decision-making framework for robotic applications. *Artificial Intelligence*, 247:187–212, 2017.

Eyal Amir and Allen Chang. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33:349–402, 2008.

Ronen I Brafman and Moshe Tennenholtz. R-max - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2003.

Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34(1):1–25, 2009.

Ashwin Deshpande, Brian Milch, Luke S Zettlemoyer, and Leslie Pack Kaelbling. Learning probabilistic relational dynamics for multiple tasks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 83–92, 2007.

Carlos Diuk, Andre Cohen, and Michael L Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 240–247, 2008.

Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43(1-2):7–52, 2001.

Floriana Esposito, Stefano Ferilli, Nicola Fanizzi, Teresa Maria Altomare Basile, and Nicola Di Mauro. Incremental learning and concept drift in INTHELEX. *Intelligent Data Analysis*, 8(3):213–237, 2004.

Daniel H Grollman and Odest Chadwicke Jenkins. Dogged learning for robots. In *Proceedings of the International Conference on Robotics and Automation*, pages 2483–2488, 2007.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

Todd Hester and Peter Stone. TEXPLORE: real-time sample-efficient reinforcement learning for robots. *Machine Learning*, 90(3):385–429, 2013.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, pages 253–302, 2001.

Katsumi Inoue, Tony Ribeiro, and Chiaki Sakama. Learning from interpretation transition. *Machine Learning*, 94(1):51–79, 2014.

Sergio Jiménez, Fernando Fernández, and Daniel Borrajo. The PELA architecture: integrating planning and learning to improve execution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1294–1299, 2008.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

Thomas Keller and Patrick Eyerich. PROST: Probabilistic Planning Based on UCT. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 119–127, June 2012.

W Bradley Knox and Peter Stone. Reinforcement learning from simultaneous human and MDP reward. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 475–482, 2012.

Andrey Kolobov, Peng Dai, Mausam, and Daniel S Weld. Reverse iterative deepening for finite-horizon MDPs with large branching factors. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 146–154, 2012.

George Konidaris, Ilya Scheidwasser, and Andrew G Barto. Transfer in reinforcement learning via shared features. *The Journal of Machine Learning Research*, 13(1):1333–1371, 2012.

Johannes Kulick, Marc Toussaint, Tobias Lang, and Manuel Lopes. Active learning for teaching a robot grounded relational symbols. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1451–1457, 2013.

Tobias Lang and Marc Toussaint. Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research*, 39:1–49, 2010.

Tobias Lang, Marc Toussaint, and Kristian Kersting. Exploration in relational domains for model-based reinforcement learning. *Journal of Machine Learning Research*, 13:3691–3734, 2012.

Lihong Li, Michael L Littman, Thomas J Walsh, and Alexander L Strehl. Knows what it knows: a framework for self-aware learning. *Machine Learning*, 82(3):399–443, 2011.

Iain Little and Sylvie Thiebaux. Probabilistic planning vs. replanning. In *Proceedings of the ICAPS Workshop on IPC: Past, Present and Future*, 2007.

Michael L Littman. Reinforcement learning improves behaviour from evaluative feedback. *Nature*, 521(7553):445–451, 2015.

David Martínez, Guillem Alenyà, and Carme Torras. Planning robot manipulation to clean planar surfaces. *Engineering Applications of Artificial Intelligence*, 39:23–32, 2015a.

David Martínez, Guillem Alenyà, and Carme Torras. V-MIN: Efficient reinforcement learning through demonstrations and relaxed reward demands. In *Proceedings of The AAAI Conference on Artificial Intelligence*, pages 2857–2863, 2015b.

David Martínez, Tony Ribeiro, Katsumi Inoue, Guillem Alenyà, and Carme Torras. Learning probabilistic action models from interpretation transitions. In *Technical Communication of the International Conference on Logic Programming, CEUR Workshop Proceedings*, volume 1433(30), 2015c.

David Martínez, Guillem Alenyà, Carme Torras, Tony Ribeiro, and Katsumi Inoue. Learning relational dynamics of stochastic domains for planning. In *International Conference on Automated Planning and Scheduling*, pages 235–243, 2016.

Çetin Meriçli, Manuela Veloso, and H Levent Akın. Multi-resolution corrective demonstration for efficient task execution and refinement. *International Journal of Social Robotics*, 4(4):423–435, 2012.

Bogdan Moldovan, Plinio Moreno, Martijn van Otterlo, José Santos-Victor, and Luc De Raedt. Learning relational affordance models for robots in multi-object manipulation tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4373–4378, 2012.

Matthew Molineaux and David W Aha. Learning unknown event models. In *Proc. of the AAAI Conference on Artificial Intelligence*, pages 395–401, 2014.

Kira Mourão. Learning probabilistic planning operators from noisy observations. In *Proceedings of the Workshop of the UK Planning and Scheduling Special Interest Group*, 2014.

Kira Mourão, Luke S Zettlemoyer, Ronald Petrick, and Mark Steedman. Learning STRIPS operators from noisy and incomplete observations. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 614–623, 2012.

Hanna M Pasula, Luke S Zettlemoyer, and Leslie Pack Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29(1):309–352, 2007.

Tony Ribeiro and Katsumi Inoue. Learning prime implicant conditions from interpretation transition. In *Proceedings of the International Conference on Inductive Logic Programming, LNAI*, volume 9046, pages 108–125, 2014.

Tony Ribeiro, Morgan Magnin, Katsumi Inoue, and Chiaki Sakama. Learning multi-valued biological models with delayed influence from time-series observations. In *Proc. of the International Conference on Machine Learning and Applications*, pages 25–31, 2015.

Scott Sanner. Relational dynamic influence diagram language (RDDL): Language description. *Unpublished ms. Australian National University*, 2010.

Daniel Sykes, Domenico Corapi, Jeff Magee, Jeff Kramer, Alessandra Russo, and Katsumi Inoue. Learning revised models for planning in adaptive systems. In *Proceedings of the International Conference on Software Engineering*, pages 63–71, 2013.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

Ingo Thon, Niels Landwehr, and Luc De Raedt. Stochastic relational processes: Efficient inference and applications. *Machine Learning*, 82(2):239–272, 2011.

Mauro Vallati, Lukáš Chrpa, Marek Grześ, Thomas L McCluskey, Mark Roberts, and Scott Sanner. The 2014 international planning competition: Progress and trends. *AI Magazine*, 36(3):90–98, 2015.

Thomas J Walsh. *Efficient learning of relational models for sequential decision making*. PhD thesis, Rutgers, The State University of New Jersey, 2010.

Thomas J Walsh, István Szita, Carlos Diuk, and Michael L Littman. Exploring compact reinforcement-learning representations with linear regression. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 591–598, 2009.

Thomas J Walsh, Kaushik Subramanian, Michael L Littman, and Carlos Diuk. Generalizing apprenticeship learning across hypothesis classes. In *Proceedings of the International Conference on Machine Learning*, pages 1119–1126, 2010.

Thomas J Walsh, Daniel K Hewlett, and Clayton T Morrison. Blending autonomous exploration and apprenticeship learning. In *Advances in Neural Information Processing Systems*, pages 2258–2266, 2011.

Håkan LS Younes and Michael L Littman. PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects. *Technical Report CMU-CS-04-162*, 2004.

George Zhu, Dan Lizotte, and Jesse Hoey. Scalable approximate policies for Markov decision process models of hospital elective admissions. *Artificial Intelligence in Medicine*, 61(1): 21–34, 2014.

Hankz Hankui Zhuo and Subbarao Kambhampati. Action-model acquisition from noisy plan traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 2444–2450, 2013.

Hankz Hankui Zhuo and Qiang Yang. Action-model acquisition for planning via transfer learning. *Artificial Intelligence*, 212:80–103, 2014.