Relational Reinforcement Learning with Guided Demonstrations

David Martínez, Guillem Alenyà and Carme Torras

Institut de Robotica i Informatica Industrial (CSIC-UPC), Llorens i Artigas 4-6, 08028 Barcelona, Spain
{dmartinez,galenya,torras}@iri.upc.edu

Abstract

Model-based reinforcement learning is a powerful paradigm for learning tasks in robotics. However, in-depth exploration is usually required and the actions have to be known in advance. Thus, we propose a novel algorithm that integrates the option of requesting teacher demonstrations to learn new domains with fewer action executions and no previous knowledge. Demonstrations allow new actions to be learned and they greatly reduce the amount of exploration required, but they are only requested when they are expected to yield a significant improvement because the teacher's time is considered to be more valuable than the robot's time. Moreover, selecting the appropriate action to demonstrate is not an easy task, and thus some guidance is provided to the teacher. The rule-based model is analyzed to determine the parts of the state that may be incomplete, and to provide the teacher with a set of possible problems for which a demonstration is needed. Rule analysis is also used to find better alternative models and to complete subgoals before requesting help, thereby minimizing the number of requested demonstrations. These improvements were demonstrated in a set of experiments, which included domains from the international planning competition and a robotic task. Adding teacher demonstrations and rule analysis reduced the amount of exploration required by up to 60% in some domains, and improved the success ratio by 35% in other domains.

Keywords: active learning, learning guidance, planning excuse, reinforcement learning, robot learning, teacher demonstration, teacher guidance

1. Introduction

Learning tasks with robots is a very interesting topic, where impressive results may be obtained without having to design specific algorithms for each task. However, learning high-level tasks can be very time consuming because considerable experience is required to learn in real-world domains where stochastic actions with multiple effects can be performed. In general, learning from scratch in such domains requires that hundreds of actions are executed even for simple tasks. Robots require several seconds, or even minutes, to execute high-level actions, which means that the total time required to learn a task can be excessively large. However, learning with the help of a human teacher can reduce this learning time greatly, although human time is usually considered to be more valuable than that of a robot, and thus the robot should only request help from a teacher a limited number of times.

This is the underlying concept of the learning approach in the European project IntellAct [1], the goal of which is to exploit the semantics of manipulations in terms of objects, actions, and their consequences to reproduce human actions using robots. This project provides a framework where a robot should learn manipulation tasks with no prior knowledge, as well as performing high-level reasoning to adapt to changes in the domain. The robot starts without any knowledge of the available actions and it has to request demonstrations from a teacher whenever new actions are required to complete a task. These actions are learned at a low level using programming by teleoperation [2], as well as at a high level with a decision maker. Moreover, a 3D object recognition algorithm [3] and a particle filter tracker [4] are used to obtain a state for reasoning and to generate semantic event chains [5], which encode the demonstrated action sequences. Using this information, the decision maker can learn about the domain constantly while completing the assigned tasks [6]. In this study, we propose a new reinforcement learning algorithm for the decision maker called Relational Exploration with Demonstrations (REX-D), which can learn to perform manipulation tasks based on only a small number of action executions with the help of a teacher guided by rule analysis.

In some robotic systems, such as those that involve complex manipulation sequences, the decision maker has little input data and long periods of time to process it, because the actions take a long time to execute. Therefore, a good approach for learning such high-level tasks is model-based reinforcement learning (RL) [7]. This approach allows a model to be obtained that represents the actions that the robot can execute. The model is generated from the experiences obtained when the robot executes the actions.

To learn tasks as rapidly as possible, we need a highly compact representation of the model, and thus we use relational models. These models generalize over different objects of the same type, thereby reducing the learning complexity of domains to the number of different types of objects in them. Several approaches apply RL to obtain good results in specific robotic tasks using relational domains [8, 9], as well as with general relational models [10, 11]. A fundamental problem in RL is balancing exploration and exploitation. To reduce the number of experiences required, Lang et al. [12] proposed the REX algorithm, which used relational count functions to apply relational generalization to the exploration-exploitation dilemma. In our approach, we learn general relational models using the relational generalization of REX, but we also include teacher demonstrations and rule analysis to reduce the number of experiences required even further, as well as facilitating generalization over different tasks.

In general, there is some uncertainty about the effects of an action executed by a robot. This uncertainty is important in some tasks, thus the RL algorithm should be able to handle stochastic effects. In the KWIK framework [13], a method was proposed for learning the probabilities associated with a given set of action effects using linear regression [14], as well as an extension for learning the action effects themselves [15]. However, a large number of samples are needed because the problem of learning action effects is NP. In our proposed method, we use the learner proposed by Pasula et al. [16], which employs a greedy algorithm to obtain rule sets that optimize a score function. Although this does not obtain the optimal solution, it generates good rule sets based on only a few experiences. Furthermore, it generates rules with deictic references and noisy effects, which make models more compact and tractable.

Learning from Demonstration (LfD) is a supervised learning paradigm, where a teacher transfers knowledge of tasks or skills to a robotic system by performing a demonstration of the task or skill [17]. Thus, we propose the combination of RL with LfD to obtain a system that learns new tasks without needing to know the actions in advance. Demonstrations are also very useful for improving the learning process, as well as for adapting to different tasks when new unknown actions need to be introduced.

The problem of integrating demonstrations with RL-like algorithms has been addressed previously. Meriçli et al. [18] used a teacher to improve the robot policies, where corrective demonstrations were issued whenever the robot did not perform as well as the teacher expected. Walsh et al. [19] expanded the apprenticeship protocol of Abbel and Ng [20] by proposing a system where the teacher reviews the actions selected, and a demonstration of the optimal action is performed whenever they are not optimal. TAMER [21] is a framework where the teacher provides reinforcement rewards that evaluate the performance of the robot, and the system exploits its current model by choosing the actions that are expected to be the most highly reinforced. In these approaches, the teacher has to intervene to improve the robot behavior whenever it is not sufficiently satisfactory. By contrast, our algorithm actively requests demonstrations from the teacher whenever help is needed, thereby releasing the teacher from having to monitor the system continuously.

Active demonstration requests have been included in algorithms with confidence thresholds [22], which request demonstrations for a specific part of the state space whenever the system is not sure about the expected behavior. A confidence-based method was also described in [23], which was combined with supplementary corrective demonstrations in error cases. Agostini et al. [24] request demonstrations from the teacher when the planner cannot find a solution with its current set of rules. Our approach combines active demonstration requests with autonomous exploration. Because the teacher's time is considered to be very valuable, demonstration requests should be limited and replaced with exploration whenever possible.

When a demonstration is requested from the teacher, he does not know which parts of the model are already known. In many tasks, several actions may be selected at a given time to complete the task. However, if no guidance is provided to the teacher, he may demonstrate an action that is already known by the system. In the model, the actions are represented as a set of action rules, which can be applied over a state space. Therefore, we propose the use of a rule analysis approach to provide some guidance to the teacher so he can demonstrate the unknown parts needed by the decision maker. To explain failures when planning such models, Göbelbecker et al. [25] designed a method for finding "excuses", which are changes made to the state that make the planner find a solution. Based on these excuses, we analyze the rules to provide guidance to the teacher. Moreover, because we use a greedy learning algorithm, excuses are also useful for finding alternative models that explain unexpected states.

In summary, we propose a RL algorithm that can request demonstrations from a teacher whenever it requires new unknown actions to complete a task, which significantly reduces the number of experiences required to learn. This approach is very useful for addressing two fundamental problems in robotics: generalizing to different tasks and the large amount of time required for learning. Adding new actions allows the robot to apply previous knowledge to different tasks that may require new actions, while optimal demonstrations provide more information to the learner than random exploration. Moreover, we also employ model analysis to guide the teacher, thereby making his demonstrations as helpful as possible and improving the model when using a greedy learner.

The remainder of this paper is organized as follows. First, we review the background related to our study, before presenting our REX-D algorithm. Next, the model analysis and its advantages are explained, followed by an experimental evaluation of our system. Finally, we give our conclusions and some suggestions for future research.

2. Background on relational RL

In this section, we present the background related to our RL algorithm. First, we describe Markov Decision Processes (MDPs), which we use to formulate the problem. Next, we explain the relational representation used for a compact state space. Finally, we explain the learner and how to apply relational generalization to RL.

2.1. MDP

MDPs are used to formulate fully observable problems with uncertainty. A finite MDP is a five-tuple $\langle S, A, T, R, \alpha\rangle$ where:

- S is a set of discrete states.
- A is a set of actions that an agent can perform.
- $T : S \times A \times S \to [0, 1]$ is the transition function that describes the probability of obtaining a successor state by executing an action from a given state.
- $R : S \times A \to \mathbb{R}$ is the reward function.
- $\alpha \in [0, 1)$ is a discount factor that measures the degradation of future rewards.

The goal is to find a policy $\pi : S \to A$ which selects the best action for each state to maximize the future rewards. The sum of rewards is $V^\pi(s) = E[\sum_t \alpha^t R(s_t)|s_0 = s, \pi]$ which is the function to maximize.

2.2. Relational representation

Relational domains represent the state structure and objects explicitly. These domains are described using a vocabulary of predicates P and actions A, and a set of objects C_π. Predicates and actions take objects as arguments to define their grounded counterparts.

The state is defined with a set of positive grounded predicates. An example of a predicate is *on(X,Y)*, and some of its groundings could be *on(box1, box2)* and *on(box3, box1)*.

In a relational domain, actions are represented as grounded functions. The arguments of an action are the objects with which the action interacts. An example of an action is *pickup(X)*, while a grounded action would be *pickup(box1)*.

2.3. Model

The system represents the model as a set of rules Γ, which define the preconditions and effects of the actions. Because we want to address stochastic environments, we use Noisy Indeterministic Deictic (NID) rules [16]. A NID rule r is defined as

$$a_r(\chi) : \phi_r(\chi) \to \begin{cases} p_{r,1} & : \Omega_{r,1}(\chi) \\ \dots & : \\ p_{r,n_r} & : \Omega_{r,n_r}(\chi) \\ p_{r,0} & : \Omega_{r,0} \end{cases},$$

where

- a_r is the action that the rule represents.
- $\phi_r(\chi)$ are the preconditions. When the action a_r is executed, only the rule r with preconditions present in the state will be applicable.
- $\Omega_{r,i}$ are the effects, which have an associated probability of $p_{r,i} \in [0, 1]$. The sum of probabilities is always $\sum_i p_{r,i} = 1$. Effects define the set of predicates that are changed in the state when the rule is applied.
- $\Omega_{r,0}$ is called the noise effect, which represents all the other rare and complex effects that are not represented by any $\Omega_{r,i}$ and that are not covered explicitly for compactness and generalization reasons.
- χ is the set of variables for the rule. To obtain a grounded action, each one of these variables is set as an object, thereby yielding a set of grounded predicates in the preconditions and the effects of the rule. There are two types of variables: the parameters of the action and the deictic references.
- An action has a set of parameters χ_a, which represent the objects with which the action interacts.
- Actions also have deictic references $DR = \chi \setminus \chi_a$, which identify objects related to the action that is being performed. These variables are referenced in the preconditions and their possible groundings depend on the context of the action.

A NID rule represents only one action. However, each action may be represented by several rules, where each has different preconditions.

2.4. Learner

A learner is used to obtain the rules that represent the model. The learner uses experiences obtained from executing previous actions to infer the rules.

Experiences comprises a set of triples $E = (s, a, s')$, which represent the new state s obtained after executing the action a in state s. We say that a rule covers an experience when the rule preconditions are satisfied by s, the rule represents the action executed $a_r = a$, and the effects of the rule are the changes between the initial and the final state for the experience $\Omega_{r,i} = s' \setminus s$.

Pasula et al.'s learning algorithm [16] is used to learn the NID rules. The problem of learning stochastic rule sets is NP-hard [16], so a greedy heuristic search is used to obtain the rule sets. The heuristic generates rule candidates and a score function is employed to select the best options. Pasula et al.'s score function for optimizing the trade-off between the accuracy of the rules and their complexity is:

$$S(\Gamma) = \sum_{(s,a,s')\in E} log(\hat{P}(s'|s, a, r_{s,a})) - \alpha \sum_{r\in\Gamma} PEN(r), \tag{1}$$

where $r_{s,a}$ is the rule that covers the experience when a is performed in s, $\hat{P}$ is the likelihood of the experience, $PEN(r)$ is a complexity penalty, and α is a scaling parameter.

Complete observability is also assumed throughout this study. REX-D can address partially observable domains if the learner and the planner used are able to learn them. Other approaches have been proposed that learn complex models in partially observable domains [26]. However, we use Pasula et al.'s learner because its generalization capabilities and its ability to tackle uncertain action effects are better suited than partial observability learners to the tasks considered in the present study.

2.5. RL in relational domains

In a RL problem, the transition distribution T is unknown a priori, and thus it has to be learned. In particular, we use a model-based RL, where a model is estimated based on experiences using the learner described earlier and the policies that the robot executes are obtained from this model. The agent can employ two strategies to solve the task, as follows.

- Exploration: Executing actions to explore the model and to estimate T, thereby allowing better policies to be obtained in the future.

- Exploitation: Executing actions to obtain high rewards.

In RL, a very important problem is the exploration-exploitation dilemma, which involves finding a balance of sufficient exploration to obtain a good model without consuming too much time addressing the low-value parts of the state.

Lang et al. [12] proposed a solution to the exploration-exploitation dilemma in relational worlds, which employs the relational representation to reduce the number of samples before treating a state as known. Their proposed method uses the following context-based density formula for state-action pairs:

$$k(s,a) = \sum_{r \in \Gamma} c_E(r) I(r = r_{s,a}), \text{ with } c_E(r) := |E(r)|, \tag{2}$$

where $c_E(r)$ counts the number of experiences that cover the rule with any grounding and $I()$ is a function, which is 1 if the argument is evaluated as true and 0 otherwise.

Using equation 2, the algorithm (REX) is specified to obtain near-optimal solutions in relational domains. Relational generalization is applied to well-known RL algorithms to obtain considerable improvements in relational domains. It has two variants: one based on R-MAX [27] and another based on E^3 [28].

2.6. Teacher demonstrations

When a demonstration is requested, the teacher demonstrates how to perform an action, its name, and its arguments. In the present study, it is assumed that the teacher knows the optimal policy π^*, but the system can also work with suboptimal policies if they also achieve the goal successfully. Inferior demonstrations may lead to worse policies being learned, but the system will request further demonstrations and explore until it has learned how to meet its goal. In addition, even if the teacher knows the optimal (or near optimal) policy, implementing a policy in a robot is a very tedious task, which may take a much longer time to model and to test properly compared with learning it using the proposed algorithm. Moreover, RL algorithms also facilitate adaptation to unexpected changes and different tasks.

3. Robot platform

As an initial step, all of the modules related to high-level reasoning were integrated into a virtual reality (VR) system with a simulated robot [29, 30], which was very useful for testing the whole system before moving it onto the real robot platform because the same interfaces were used. The VR system provided the decision maker with symbolic states, the actions executed by the robot, and the information supplied to generate semantic event chains [5], which were used to recognize the actions executed by the teacher.

The VR setup comprised a multi-screen stereoscopic rear projection installation for showing the scenario and a wireless data glove with a tracking system for interacting with it. This system could keep track of hand movements, which were used to interact with objects in the scenario in the same manner as if they were in the real world. This VR system also simulated robot executions, including noise and action failures. The decision maker could request actions from this simulated robot in the same manner as it would request them from a real robot. Figure 1 shows the teacher demonstrating an action in the VR system.

The same high-level reasoning skills were successfully integrated in a real robot platform [6]. The robot could learn new actions from demonstrations, recognize the actions executed by the teacher, and provide REX-D with the skills to generate symbolic states that represented the scenario.

To demonstrate an action, the teacher provided the action name and its arguments, before moving the robot arm to perform the action. When a new unknown action was demonstrated, programming by teleoperation [2] was used to learn the demonstrated action at a low level, thereby allowing it to be reproduced. The action was associated with the symbolic name given to the action. Transferring the knowledge to a high-level cognitive system could also be achieved with a symbolic representation that captured the interaction between objects and actions [31, 32]. However, in the present study, we assumed that the teacher supplied the name of the action that he executed for the robot.

The predicates that represented the state used for high-level reasoning were obtained using the robot cameras. A 3D object recognition module [3] determined the different objects in the scene and their positions in the very first frame, which were then tracked as they moved [4], before transforming them into semantic event chains [5] to generate a symbolic state for planning and learning. Semantic event chains also facilitate the recognition of the actions executed

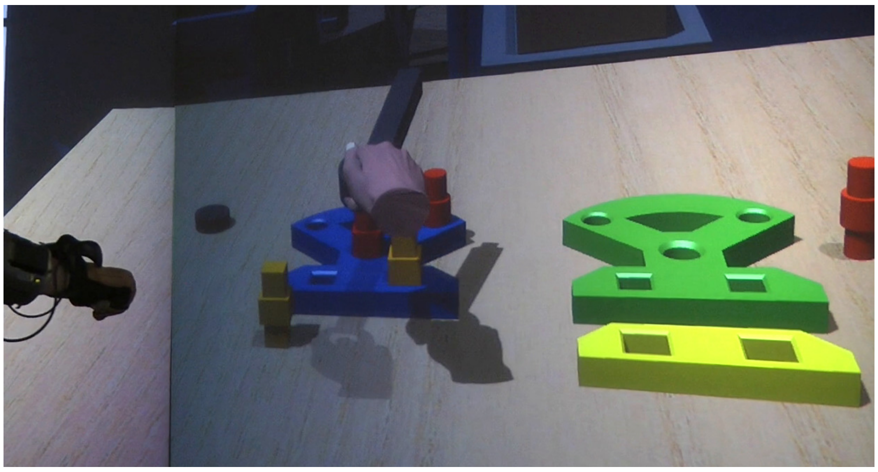

Figure 1: Cranfield benchmark scenario in virtual reality. On the left, a teacher moves the data glove, which is used to reproduce the movements of the simulated hand on the screen. In this case, the teacher demonstrates how to place the pendulum in the assembly.

by the teacher because they encode the changes in the relationships between objects at decisive time points during a manipulation procedure. Although the underlying perception algorithms work with continuous perceptions, REX-D only uses the information available before and after an action is executed because its cognitive processing is based on the state changes introduced by actions.

Although REX-D was finally integrated and tested in the real robot, the development process and experiments were mostly performed in the VR setup because it allowed us to conduct large numbers of replicates to obtain meaningful statistics, thereby facilitating the evaluation of the algorithm. The main limitations of using a real robot platform compared with a VR system are as follows,

- In a state, the objects need to be recognized perfectly, and thus they should be easily distinguishable.
- Objects need to be tracked continuously while the robot or the teacher interacts with the scenario. This process requires at least two cameras, which point in different angles to avoid occlusions, and the objects should not move faster than the tracker's frame rate.'
- The robot has to learn actions in a manner that allows them to be generalized to different object positions. In this study, these skills were limited to simple pick-and-place actions, which were sufficient to solve the tasks assigned.

3.1. Cranfield benchmark

The Cranfield benchmark, which is a standardized assembly task in robotics, was the scenario used for the real-world experiments. This scenario involves assembling an industrial item, the parts of which are shown in Fig. 2. There are precedence constraints in the assembly that restrict the order in which the parts have to be assembled. Square pegs have to be placed before the separator, the shaft before the pendulum, and all other pieces before the front faceplate.

Moreover, some variants of this scenario are used in more difficult and interesting problems, as follows.

- Pegs are difficult to place when they are in a horizontal position. Therefore, actions are required to place them in a vertical position.
- The standard initial state is shown in Fig. 2. To add more complexity to the domain, other initial states can be used, e.g., an initial state where the separator is placed before the pegs. In this case, the robot first has to remove the separator in order to place the square pegs.
- Changing some parts for new ones, such as replacing the front faceplate with another that is not compatible with a pendulum. This is an interesting problem for learning when part of the domain is already known.

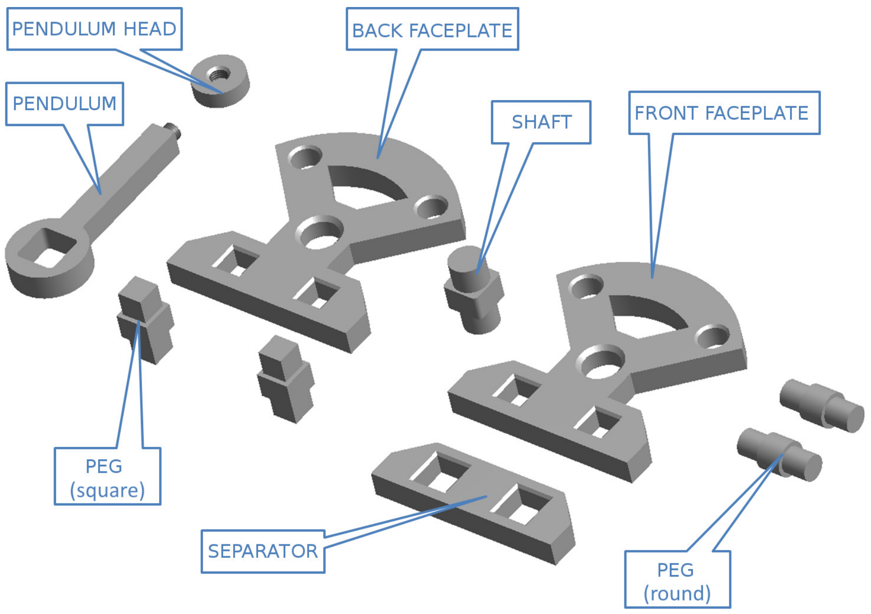

Figure 2: Cranfield benchmark assembly. The different parts have to be placed in a specific order to assemble the structure: the separator requires the square pegs to be placed, the pendulum must be placed after the shaft, and the front faceplate requires that the pegs, the separator, and the pendulum are in place.

4. Incremental reinforcement learning with demonstrations

Our environment contains a robot, which can perform different actions, a perception system to obtain the state, and a human teacher who can demonstrate actions to the robot when requested. The robot starts with no previous knowledge of the transition function T or the set of available actions A. When new goals are assigned to the robot, it tries to solve the tasks and it requests demonstrations if they were needed.

Although the number of teacher interactions should be low, it is considered beneficial to request a demonstration if it eliminates numerous automatic robot executions, which may take a very long time. The teacher is considered to employ an optimal policy π^* so he always demonstrates the best action for the current state.

4.1. Reinforcement learning with demonstrations

To incorporate demonstrations into RL, we propose the use of a variant of the E^3 version of the REX algorithm. The E^3 algorithm is based on the concept of known states: a state is considered to be known when all the state-action pairs for that state have been visited at least the confidence threshold ζ number of times. Depending on the experiences of the algorithm, it proceeds as follows.

- Whenever the robot enters an unknown state, it performs the action that requires the fewest times to explore.
- If it enters a known state, and a valid plan through known states is found, then that plan is executed (exploitation).
- If it enters a known state but no plan is found, it plans using a MDP where unknown states are assumed to have a very high value (planned exploration).

The REX algorithm generalizes this to a relational domain, but it maintains the same concepts. It uses equation 2 as a count function to decide whether a state is already known.

We propose the algorithm REX-D (Algorithm 1), where we modify REX by adding the option to request demonstrations from a teacher. REX-D explores the list of available actions until they are considered to be known in the same manner as REX, but after it enters a known state where no solution is found, the planned exploration step is substituted with a demonstration request, where the teacher executes the best action for the current state. When the teacher demonstrates an action, its treatment depends on the current knowledge of the robot, as follows.

Algorithm 1 REX-D

```
Input: Reward function $R$, confidence threshold $\zeta$
 1: Set of experiences $E = \phi$
 2: Observed state $s_0$
 3: loop
 4:     Update transition model $T$ according to $E$
 5:     if $\forall a \in A : k(s, a) \geq \zeta$ then        ▷ If the state is known
 6:         Plan using $T$
 7:         if a plan is obtained then
 8:             $a_t$ = first action of the plan
 9:         else
10:             Request demonstration
11:             $a_t$ = action demonstrated
12:         end if
13:     else
14:         $a_t = argmin_a k(s_t, a)$
15:     end if
16:     Execute $a_t$
17:     Observe new state $s_{t+1}$
18:     Add $\{(s_t, a_t, s_{t+1})\}$ to $E$
19: end loop
```

- If the demonstration is the first execution of the action, the robot learns how to execute that action and adds it to the set of actions that the learner must learn.
- If the action has already been executed, its rules are simply refined by the learner.

The ability to make a few requests of a teacher inside the RL loop has two main advantages: faster learning and the ability to add actions as they are required.

4.1.1. Faster learning

Executing actions takes a long time for robots, so it is recommended to request help from the teacher to save large periods of learning time.

Exploration is performed until a known state is reached. However, once a known state is entered, the decision maker has to decide where to explore. Exploring the whole state space requires a large number of samples; thus, a demonstration may be requested to guide the system through the optimal path.

The problem when learning a model only by exploration (in the case where a list of actions is available) is finding examples where the actions are successful. As noted by Walsh [15], when an action fails because its preconditions are not satisfied (which we will call *negative examples*), these experiences are highly uninformative because the learner cannot determine the reason for the failure. However, *positive examples* where an action changes predicates, provide very useful information because a superset of the literals that belong to the precondition is identified.

Proposition 1. In the worst case, learning the preconditions of a rule r needs $\Omega(2^{|P_r|})$ examples, where $|P_r|$ is the maximum number of grounded predicates that may appear in the rule preconditions, and $|P_r| = \sum_{p \in P} |\chi_r|^{|\chi_p|}$ where χ_r are the variables of the rule r and χ_p are the parameters of the predicate p.

Proof. If the rule preconditions have a grounded predicate set P_r which is either positive or negative, then only one combination that uses all the predicates satisfies the preconditions and $2^{|P_r|}$ action executions are needed to test all possible combinations. In the worst case, the last combination tested is the only valid combination, thus $2^{|P_r|}$ examples are needed to find it.

Example. Consider that we have to open a box with the action *open(X,Y)*, but it requires that the objects X and Y are connected in a particular way (using the predicates *connected(X,Y)* and *on(X)*). In this case, $|P_r| = 2^2 + 2^1 = 6$, and the possible preconditions are:

$$\pm connected(X, Y), \pm connected(Y, X), \pm connected(X, X), \pm connected(Y, Y), \pm on(X), \text{and } \pm\ on(Y).$$

In total, there are: $2^{|P_r|} = 2^6 = 64$ combinations. In the worst case, where the correct combination is the last one tested, the $open(X, Y)$ action has to be executed in 64 different states until a suitable one is found. □

Proposition 2. With only one positive example of a rule $e_r = (s, a_r, s')$, the action rule preconditions can be learned with $\Omega(|s|)$ examples.

Proof. Because the preconditions comprise a conjunction of predicates, we know that only the predicates in the state s where the action was executed may be present in the rule preconditions. Thus, setting each one of these predicates to its opposite to test whether it is actually one of the preconditions requires $|s|$ examples. □

Proposition 3. Adding a new positive examples of a rule $E_r = \{e_1, ..., e_n\}$ reduces the complexity of learning its preconditions to $\Omega(|P_{E_r}|)$ examples, where $P_{E_r} = \{s_1 \cap ... \cap s_n\}$.

In real-world scenarios, learning rules requires far fewer experiences than the worst cases described above. In general, most predicates are irrelevant to the execution of an action, which increases the probability of obtaining a positive example. Furthermore, robots operate in small subsets of the overall state space and their actions are designed to be executed only in the parts of the state space that they can reach. Although actions can be learned with fewer experiences, the previously described propositions provide a general view of the complexity of the learning process and the advantages of using demonstrations.

4.1.2. Online action additions

Initially, the learning system does not know which actions are available because the teacher has to demonstrate them only when they are needed. Therefore, exploration alone is not a valid strategy because a previously unknown action may be required to complete the task. When the current state is known (using only the list of actions that have already been learned) and no plan is found, a demonstration is requested by REX-D. The teacher will execute the best action for that state, which may be a new and previously unlearned action.

4.2. Generalizing to different tasks

A model is generated to represent the actions required to complete a task. If these actions are also used in another task, previous experiences are useful for this new task. Moreover, relational representations allow generalization over different objects of the same type, and thus new tasks that use the same types of objects but different layouts will also reuse previous models.

In addition to transferring the model to other tasks, the ability to request demonstrations from the teacher gives the system much more flexibility. Thus, actions do not have to be defined in advance, but the system will request demonstrations from the teacher when they are needed if unknown actions are required during the execution.

5. Rule analysis

The system generates rule sets that represent the model, and thus they contain the known information about the task. The weaknesses of the model can be assessed by analyzing these rules, as well as the relationships between different states. This information is useful for providing some guidance to the teacher, for improving the model, and for trying to complete partial tasks via subgoals.

5.1. Explaining planning failures

Whenever the system fails to plan, information can be extracted about the failure using excuses [25, 33]. Excuses are changes to the initial state that would make the task solvable, and thus they indicate the important predicates that cause the planner to fail.

Definition 1. (Excuse) Given an unsolvable planning task, using a set of objects C_π and an initial state s_0, an excuse is a pair $\varphi = \langle C_\varphi, s_\varphi \rangle$, which makes the task solvable, where C_φ is a new set of objects and s_φ is a new initial state.

Excuses can be classified as acceptable, good, or perfect, as follows.

- Acceptable excuses change the minimum number of predicates in the initial state. An excuse φ is acceptable iff $\forall \varphi', C_\varphi \subseteq C_{\varphi'}$ and $s_0 \triangle s_\varphi \subseteq s_0 \triangle s_{\varphi'}$ (where $\triangle$ denotes the symmetric set difference).
- Good excuses are acceptable excuses with changes that cannot be explained by another acceptable excuse.
- A perfect excuse is a good excuse that obtained the minimal cost using a cost function.

Applying goal regression over all acceptable excuses would be highly suboptimal, so a set of good excuse candidates will be generated [25], which are then checked with the planner.

Candidates for good excuses can be obtained under certain assumptions using a causal graph and a domain transition graph [34], which are generated with the rule set Γ that defines the domain. A causal graph CG_Γ is a directed graph that represents the dependencies of predicates between each other. A domain transition graph G_p of a predicate p is a labeled directed graph that represents the possible ways that the groundings of the predicate can change and the conditions required for those changes.

Definition 2. A causal graph CG_Γ is a directed graph that represents the dependencies of predicates between each other. An arc (u, v) exists when $u \in \phi_r$ and $v \in \Omega_r$ for a rule $r \in \Gamma$, or when both are in the effects of the rule $u, v \in \Omega_r$.

Definition 3. A domain transition graph G_p of a predicate p is a labeled directed graph that represents the possible ways that the groundings of the predicate can change and the conditions required for those changes. An arc (u, v) exists when there is a rule $r \in \Gamma$ such that u and v are groundings of p, $u \in \phi_r$ and $v \in \Omega_r$. The label comprises the predicates $\phi_r \setminus \{u\}$.

To restrict the number of candidates, we only consider those that are relevant to achieving the goal. Using the causal graph and the domain transition graph, the candidates can be obtained with a fix point iteration [25] by adding the predicates that contribute to the goal and those that are potentially required to reach other predicates added previously to the candidate set. From the set of excuse candidates, we select those that are not reachable by G_p from any predicate in the current state, or those that are involved in a cyclic dependency in CG_Γ.

Finally, the planner is used to test which of the excuse candidates should be added to obtain the best results. The best are selected as the excuses that explain the failure.

Note that rules in stochastic domains may have several effects with different probabilities as well as a noisy effect. To generate the excuses in these conditions, noisy and low probability effects are ignored when generating the causal graph and domain transition graph.

5.2. Providing guidance to the teacher

The REX-D algorithm requests demonstrations from the teacher whenever it does not know how to complete the task. However, several actions may be required to complete the task, only one of which might be unknown to the system. Thus, if no guidance is provided, the teacher may demonstrate actions that the system already knows before demonstrating the action that is actually required. Our objective is to avoid these unnecessary demonstrations, thereby minimizing the number of interactions with the teacher.

The excuses presented in Section 5.1 are used to find problems in the model to guide the teacher so he demonstrates the action that the system needs. There are several possible reasons why the system may fail to plan, i.e., the wrong preconditions have been added, missing action effects, or dead ends.

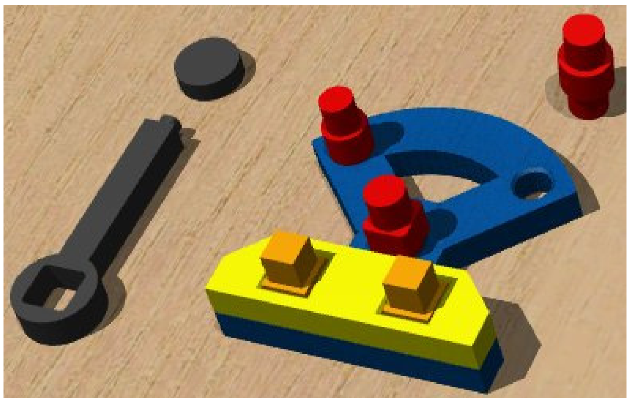

Figure 3: Intermediate state in the Cranfield benchmark. The pendulum, a rounded peg, and the front faceplate still have to be placed.

Proposition 4. Let Γ be the complete rule set where there is an incorrect rule $r_{incorrect} \in \Gamma$, which has p_w as a precondition but its effects are not dependent on it. Let φ be an excuse that adds p_w to the state and if φ does not prevent any other transition from other rules, φ is an acceptable excuse.

Proposition 5. Let Γ' be an incomplete rule set, which has one rule missing, $r_{missing} \notin \Gamma'$. If the effects $\Omega_{r_{missing}}$ are required to achieve the goal and no other rule can obtain the predicates, i.e., $\exists p \in \Omega_{r_{missing}} | \forall r' \in \Gamma', p \notin \Omega_{r'}$, then an excuse φ that includes the necessary changes to $\Omega_{r_{missing}}$ is an acceptable excuse.

We will suppose that excuses found point at the problems explained in the previous propositions. However it should be noted that excuses that create invalid alternative paths to the goal may also exist in some domains. The teacher will be warned about possible wrong preconditions or unknown needed effects:

- Warning about incorrect preconditions. When the changes in the excuse φ affect the preconditions of one or more rules ($\exists r \in \Gamma | \varphi \in \phi_r$), the teacher is warned that these rules require the excuse φ. If one of the preconditions is incorrect, the teacher can simply execute the action to prove to the system that it is not actually required.
- Warning about missing effects. The teacher is warned that the system does not know how to obtain the changed predicates in φ.

5.2.1. Guidance examples

Figure 3 shows an example of the Cranfield benchmark scenario (as explained in 3.1), where the robot has to mount the different parts to complete an assembly. In the current state, two possible actions have the same reward: placing a red peg, or placing the pendulum (gray). If the system already knows how to place a peg, the teacher will be warned that the planner does not know how to obtain a "pendulumPlaced" predicate.

It is recommended that the predicates used in the domain have descriptive names to make excuses guidance useful for non-technical teachers. Teaching the system can be relatively easy if the domain semantics are described appropriately in the predicates, as demonstrated by the following examples of robot messages.

- *I want to "pickUp(block)", but this requires "onTable(block)" [possible incorrect precondition].* If the teacher knows that the "onTable" predicate is not required, he only has to execute the pickUp action to demonstrate this.
- *I don't know how to remove a "flatTire()" [missing effect].* The action "changeTire" that removes "flatTire" has to be demonstrated.
- *I don't know how to remove a "broken(table)" [missing effect].* If the table is needed to complete the task and it cannot be repaired, this is actually a dead end. Thus, the teacher has to signal a dead end to REX-D.

Algorithm 2 GenerateRuleAlternatives

Input: Current rule set Γ, state s, experiences E, valid excuses $\varphi_1 ... \varphi_n$
Output: New rule set Γ'

```
 1: $RS_{candidates} = \phi$        ▷ List of rule set candidates
 2: for all $\varphi_i \in \varphi_1 ... \varphi_n$ do
 3:     $p_\varphi = s \triangle \varphi_i$        ▷ Predicates changed by the excuse
 4:     $\Gamma_\varphi$ = GetRulesRequiringExcuse($\Gamma, s, p_\varphi$)
 5:     for all $r_\varphi \in \Gamma_\varphi$ do
 6:         for all $p_i \in s$ do
 7:             $\Gamma_{new} = \Gamma$
 8:             Substitute precondition $p_\varphi$ for $p_i$ in rule $r_\varphi \in \Gamma_{new}$
 9:             Add $\Gamma_{new}$ to $RS_{candidates}$
10:         end for
11:     end for
12: end for
13: $\Gamma' = \Gamma$
14: for all $\Gamma_i \in RS_{candidates}$ do
15:     if Score($\Gamma_i$) $\simeq$ Score($\Gamma'$) then        ▷ Score() is computed using equation 1
16:         if $\exists$ plan($\Gamma_i, s$) then
17:             $\Gamma' = \Gamma_i$
18:         end if
19:     end if
20: end for
```

5.3. Rule alternatives

The ζ parameter (for considering known states and actions, as introduced in Section 4.1) has to be optimized for each different domain to obtain good results. However, we use a small threshold to learn rapidly and excuses are used to generate alternative models that compensate for this overconfidence. The system uses the Pasula learner (as explained in Section 2.4), which is a greedy heuristic learner for obtaining rule sets that explain previous experiences. Because the learner obtains rule sets that optimize equation 1, if these experiences are not sufficiently complete, many different rule sets could explain the experiences with similar scores.

Let E be a set of experiences and Γ a rule set that explains those experiences. If E does not cover the whole state space, different rule sets Γ' may explain the experiences as well as Γ (score(Γ) $\simeq$ score(Γ')). If no plan is found but the system obtains an excuse φ, it is assumed that Γ might not be the correct rule set. The changes in the excuses $p_\varphi \in s \triangle s_\varphi$ are used to find alternative models. The system will analyze rules with excuse changes in their preconditions $p_\varphi \in \phi_r$ and check for alternative equivalent rules that may obtain a plan for the current state. Note that the complexity grows exponentially when finding rule alternatives with excuses that change more than one predicate.

If no plan is found, the algorithm `GenerateRuleAlternatives` (Algorithm 2) tries to find an alternative rule set that explains past experiences and obtains a plan from the current state.

- First, for each excuse that explains the planner's failure, it finds the rules that are required to reach the goal and that can only be executed when the excuse has been applied. The `GetRulesRequiringExcuse` routine (Algorithm 3) finds these rules and it outputs the set Γ_φ of rules that allow a valid plan to be obtained by removing the predicates changed by the excuse from their preconditions.

- Γ_φ is used to create the set of candidates that can be valid rule alternatives. Each candidate is generated by making a copy of the current rules Γ_{new} and modifying one of the rules $r_\varphi \in \Gamma_{new}$, which is also in Γ_φ. The preconditions r_φ are changed by replacing a predicate that is changed by the excuse φ with a predicate that is present in the current state $p \in s$. The aim is to check whether the predicates changed by the excuse are actually in the preconditions, or if any other predicates in the current state should be used instead.

Algorithm 3 GetRulesRequiringExcuse

```
Input: Current rule set $\Gamma$, state $s$, excuse changes $p_\varphi$
Output: Rules $\Gamma_\varphi$ that obtain a plan if $p_\varphi$ is removed from preconditions
 1: $\Gamma_\varphi = \phi$
 2: $\Gamma_{prec_\phi} = \{r | r \in \Gamma, p_\varphi \in \phi_r\}$.        ▷ rules with $p_\varphi$ as a precondition
 3: for all $r_i \in \Gamma_{prec_\phi}$ do
 4:     $\Gamma'_i = \Gamma$
 5:     $r'_i = r_i$
 6:     Remove $p_\varphi$ from $\phi_{r'_i}$
 7:     Substitute $r_i$ for $r'_i$ in $\Gamma'_i$
 8:     if $\exists$ plan($\Gamma'_i, s$) then
 9:         Add $r_i$ to $\Gamma_\varphi$
10:     end if
11: end for
```

- After the rule set candidates are obtained, the score is calculated (using equation 1) for each. If the score is at least equal to the current rule set score, then the candidate can explain past experiences as well as the current rule set, and thus it is a good candidate. Note that although many candidates are usually generated, the score can be calculated very rapidly and most of the candidates are pruned during this step.
- Finally, the planner uses good candidates to plan from the current state and a candidate is selected as the output rule set if it yields a plan.

5.4. Subgoals

Help is requested from a teacher if the planner cannot obtain a sequence of actions to complete the task. However, the teacher may be asked to perform an action that has to be executed after executing other actions that the system already knows. In some scenarios, it is useful to get as close as possible to the goal to minimize the number of actions that the teacher has to execute. Thus, subgoals are used to approach the goal. Note that the goal has to be defined as a set of target predicates for this feature to work.

Definition 4. (Subgoal) Given a goal $G = \{p_1, ..., p_n\}$, a subgoal is a set of predicates that belongs to the goal, or the requirements of goal predicates $G' = \{p_1, ..., p_n \, | p_i \in G \vee p_i \in requirements(G)\}$. The requirements of a predicate are obtained by following the precondition-effect arcs in the causal graph defined by the rules, requirements(p) = $\{p' \vee requirements(p') | \exists \text{ arc}(p', p) \in CG_\Gamma\}$.

With a valid excuse φ, we can generate a subgoal G_φ to complete the task up to the point where φ is needed, as explained in the routine `SubgoalGeneration` (Algorithm 4). This algorithm starts by initializing the subgoal with the goal predicates and finding the subgoal predicates that have a dependency on the predicates changed by the excuse in the causal graph. These subgoal predicates are considered to be unreachable without the changes defined by the excuse, and thus they must be removed from the subgoal. Instead of only removing those predicates, they are substituted by their dependencies in the causal graph. This process is repeated until all of the subgoal predicates no longer depend on the excuse changes.

Figure 4 shows an example of the Cranfield benchmark domain, where the robot has to mount several parts to complete the assembly. If the robot does not know how to place the front faceplate, the teacher would have to assemble all the other parts before demonstrating how to assemble the faceplate. To reduce the number of demonstrations required, the decision maker can create a subgoal where all of the known parts (the peg and the pendulum) should be mounted before requesting help.

6. Experimental results

In this section, we present the results obtained using the REX-D approach. We compared the following configurations to assess the performance of the different algorithms explained in the present study.

Algorithm 4 SubgoalGeneration

Input: Goal G, excuse changes p_φ, causal graph CG_Γ
Output: Subgoal G'

```
1: $G' = G$
2: while $\exists p \in G' | p_\varphi \in requirements(p)$ do
3:     Remove $\{p | p \in G', p_\varphi \in requirements(p)\}$ from $G'$    ▷ Remove predicates that depend on $p_\varphi$
4:     Add $\{p' | p \in G', \exists\, \text{arc}(p', p) \in CG_\Gamma\}$ to $G'$    ▷ Add the requirements for the removed predicates
5: end while
```

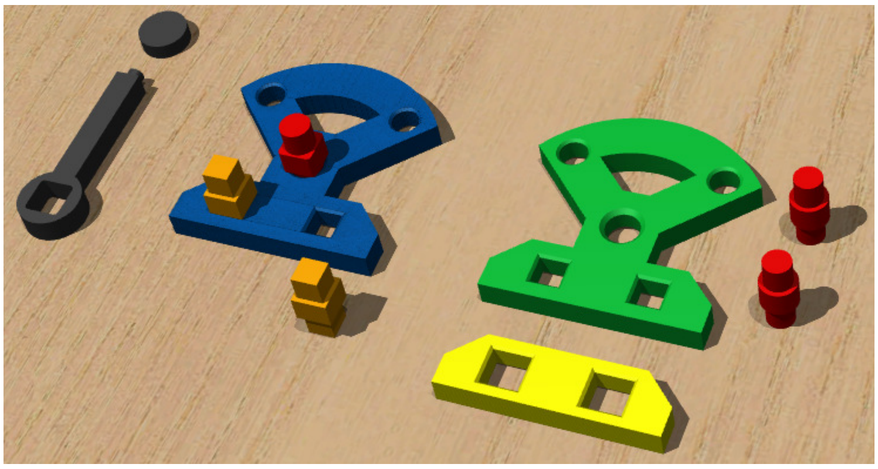
Figure 4: Intermediate state in the Cranfield benchmark scenario. Most of the assembly parts have not yet been placed in position.

- REX: The E^3 version of the REX algorithm from Lang et al. The results were obtained from [12].
- REX-D basic: The REX-D algorithm (Section 4.1).
- REX-D rules: The REX-D algorithm with rule alternatives (Section 5.3). Only excuses that changed one predicate were considered.
- REX-D rules+sub: The REX-D algorithm with rule alternatives and subgoals (Section 5.4).

Different scenarios were used to perform these comparisons. First, two problems from the international planning competition [35] were tested in order to make comparisons with other relational RL algorithms: *Exploding Blocksworld* and *Triangle Tireworld*. These are interesting problems with dead ends, which make extensive use of relational actions.

The robot setup for the REX-D algorithm was devised using the Cranfield benchmark (3.1). To demonstrate the performance of REX-D in this scenario, extensive experiments were conducted using this setup in a VR system. The following features of our approach were assessed.

- The capacity to cope with stochastic domains compared with deterministic domains.
- The generalization capabilities over similar tasks.
- The improvements obtained using rule alternatives in terms of reducing the number of demonstration requests needed (see Section 5.3).
- The improvements obtained by using subgoals in terms of reducing the number of demonstration requests (see Section 5.4).

To evaluate the relational algorithms we set the threshold ζ to consider known states and actions. Similar to other RL algorithms [12, 36], we set ζ heuristically because it cannot be derived from theory when using a heuristic learner. To obtain results that are comparable to those produced by REX, we used a value employed in previous studies ($\zeta = 2$), which obtains good results in many scenarios. In addition, the decision maker was limited to executing 100 actions per episode. After these actions were executed, the episode was marked as failed and the experiment continued with the following episode. All of the REX-D configurations used Gourmand [37] as the planner.

6.1. Comparing REX and REX-D

In this section, we compare the REX and REX-D algorithms. Although the results are not directly comparable due to the addition of demonstrations, these experiments illustrate the improvements obtained by using a teacher that demonstrates a few actions. The domains used for these experiments were taken from the international planning competition in 2008 [35].

6.1.1. Experiment 1: Exploding Blocksworld (IPPC)

This domain is an extension of the well-known Blocksworld domain that includes dead ends. The robot employs "pick up" and "put on" actions to position a set of blocks in a given layout. A block that is placed on the table, or another block, has a probability of exploding and destroying the object beneath. After a block is destroyed, it cannot be picked up and no other blocks can be placed on top of it. Destroying the table also implies that the system cannot place blocks on it anymore. To solve these problems, the planner has to take care to avoid destroying the blocks that are important for reaching the solution. The representation of this domain with NID rules comprised four actions represented by six rules, and eight different types of predicates.

The results obtained are shown in Fig. 5. The performance of the REX-D algorithm was clearly better because demonstrations allowed it to quickly learn the correct usage of the different actions. By contrast, REX had to execute an average of 120 extra exploration actions to learn how to apply the actions correctly. Given that each action may take several seconds to complete, the amount of time saved can be very important.

The main problem for algorithms without demonstrations is finding positive examples. As explained in Proposition 1, we can calculate the number of experiences that would be required to obtain a positive example with the *pick-up(X,Y)* action in the worst case. For simplicity, we assume that this action does not have deictic references. As $P_r = 16$, $\Omega(2^{16})$ different experiences would be required in the worst case. Nevertheless, having one positive example, such as *pick-up(box5, box3)*, allows its preconditions to be learned with just 16 different experiences, i.e., one to test each of the 16 predicates in the state that may be related to the action, as explained in Proposition 2. Although the actual number of experiences required to learn this domain is much lower (the rule only has four preconditions and the initial state also limits the number of incorrect experiences that can be performed), these theoretical numbers reflect the reduction in complexity obtained through demonstrations.

Using rule alternatives and subgoals reduced the number of demonstrations required. However, the alternative paths that they generated sometimes led the robot to a dead end, whereas demonstrations took the optimal path.

The number of teacher interactions varied between five and seven, which was sufficiently low considering the improvements in the overall number of actions required and the improvements in the success ratio. It should also be noted that the teacher was no longer required after the fifth episode.

6.1.2. Experiment 2: Triangle Tireworld (IPPC)

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.

The results obtained in this domain are shown in Fig. 6. The difficulty of this domain was the dead ends that the agent encountered when it had a flat tire and no spare tires were available. Both algorithms failed to learn how to change a tire if no flat tires were obtained in a location with a spare tire during the initial episodes. After they considered the actions as known, they always took the shortest dangerous path because they did not know that tires

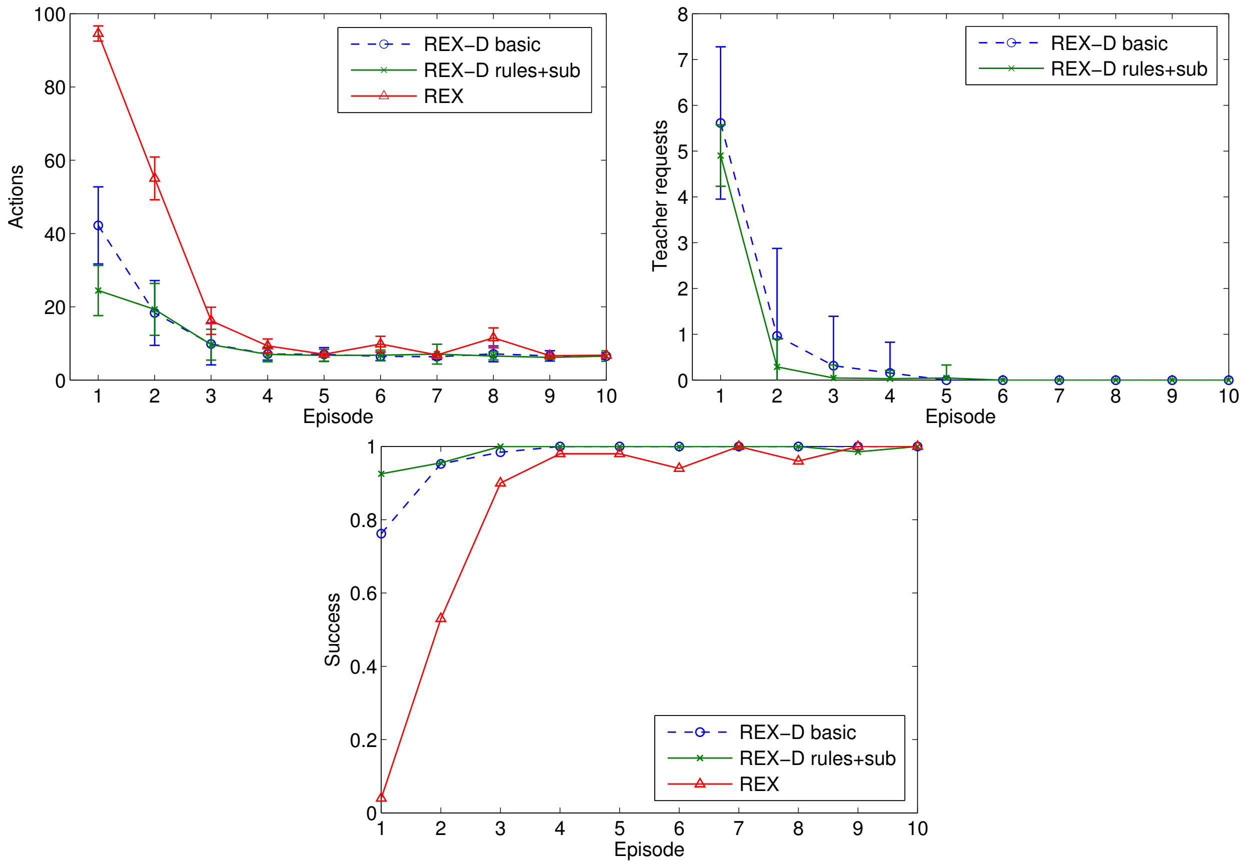

Figure 5: Experiment 1 (Exploding Blocksworld, problem instance 5): The robot started with no knowledge of the actions in the first episode. The results shown are the mean and standard deviations obtained over 50 runs. REX data were obtained from [12]. **Upper left:** the number of actions executed in each episode. **Upper right:** the number of teacher demonstrations requested. **Bottom:** the success ratio of the algorithms.

could be changed. The main problem for REX is that it moved through random paths initially and it could become confident about its model before learning how to fix a tire. REX-D exhibited better behavior because the initial demonstrations already moved the agent through roads with spare tires and a flat tire could be replaced if the car had a spare tire in its location. REX-D also required more actions per episode because it selected longer and safer paths.

Because it was easy to learn the preconditions in this domain and the difficulty was avoiding dead ends, rule alternatives and subgoals did not improve the results.

6.2. *Cranfield benchmark scenario*

The REX-D algorithm was devised in the framework of the EU Project IntellAct [1]. In this project, the robot has to learn how to complete the Cranfield benchmark assembly with the help of a teacher. The robot starts with no previous knowledge and it learns different actions based on teacher demonstrations and experience. The decision maker that uses the REX-D algorithm is in charge of high-level reasoning, where it decides when to request teacher demonstrations and which actions to execute. Although REX-D has been integrated successfully into a robot platform [6] as described in Section 3, all of the results presented in this section were executed by a simulated robot in the VR setup because it permitted large numbers of replicates to be generated to allow a meaningful statistical analysis of the performance of our algorithms.

The state that defines the scene sent to the decision maker included the following information.

- The parts of the assembly laid horizontally on the table.

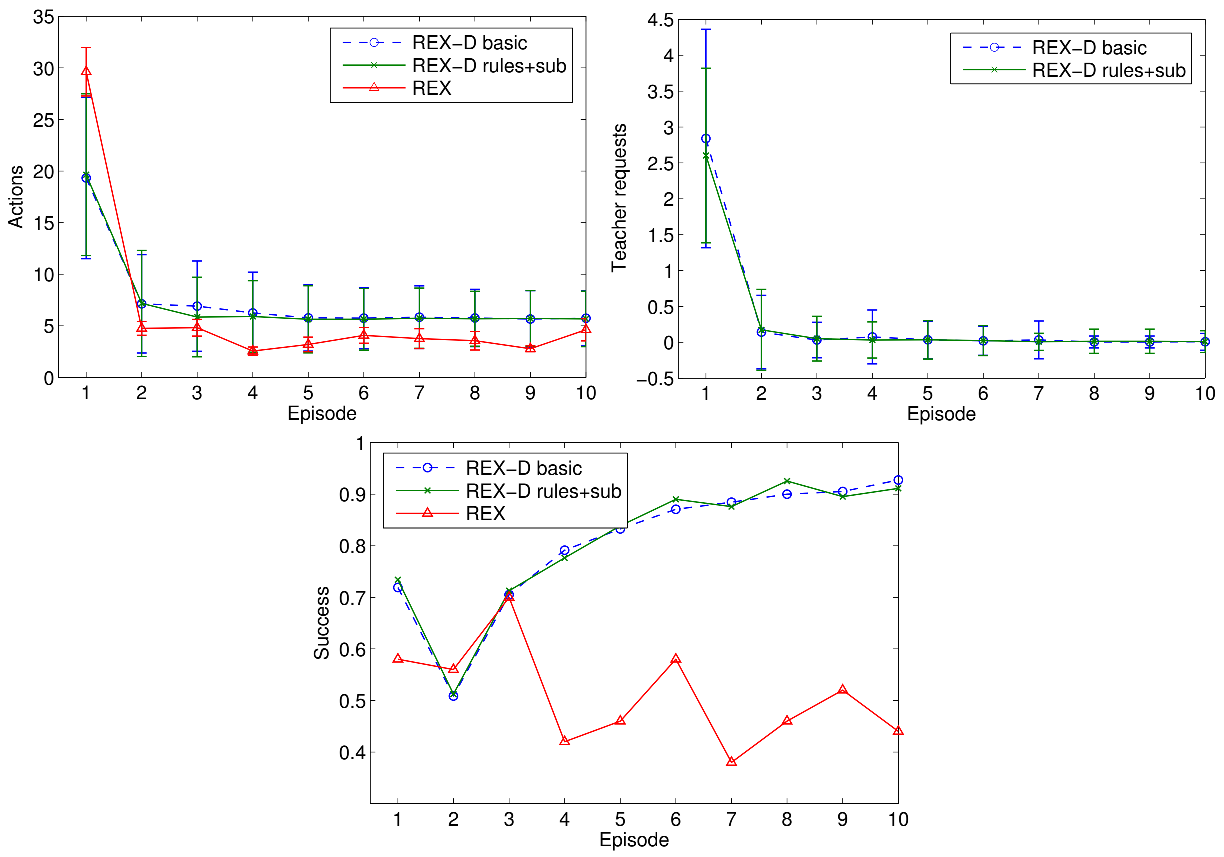

Figure 6: Experiment 2 (Triangle Tireworld, problem instance 1): The robot started with no knowledge of the actions in the first episode. The results shown are the means and standard deviations obtained from 100 runs. REX data was obtained from [12]. **Top left:** the number of actions executed in each episode. **Top right:** the number of teacher demonstrations requested. **Bottom:** the success ratio of the algorithms.

- The parts that could be grasped.
- The holes that were free.
- The parts that were already placed correctly.

In this section, we present several experiments based on the Cranfield benchmark scenario. These experiments were performed in the VR system since an excessive number of teacher requests were required to obtain statistically significant results, which would have been tedious for a human teacher. Therefore, an automated teacher that planned with the correct rule set was used to perform the experiments described in this section. This automated teacher also considered the guidance provided by the system when generating plans. Because it obtained optimal policies, the results were the same as those that would have been obtained with a human teacher.

6.2.1. Experiment 3: Standard Cranfield benchmark

We tested how well REX-D performs in stochastic domains compared with deterministic domains. These experiments were performed using the standard Cranfield benchmark, where we changed the success probability of the actions to 60% for the stochastic case. There were no dead ends and the only change was the success ratio of the actions; thus, the differences in the results simply indicated the changes obtained with noise when learning.

Figure 7 shows the results obtained. The REX-D algorithm produced good results in both the deterministic and stochastic domains. By contrast, REX had to explore intensively, especially in the stochastic case. Using REX-D with rule alternatives significantly reduced the number of demonstrations requested. In the deterministic case, only the initial demonstrations were required, while only a few requests were made in the probabilistic case.

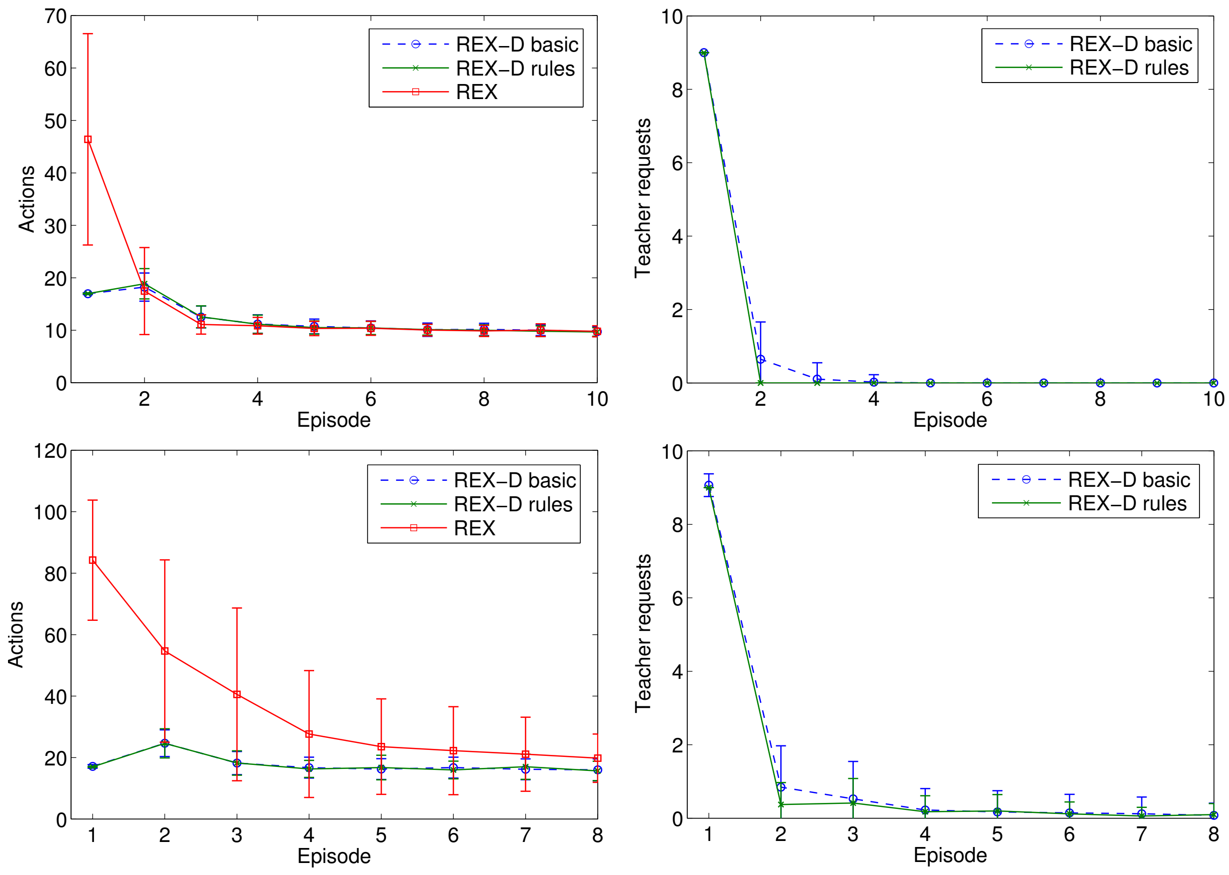

Figure 7: Experiment 3 (Standard Cranfield benchmark): The robot started with no knowledge of the actions in the first episode. The standard Cranfield benchmark assembly had to be completed in each episode. As there were no dead-ends and a teacher was available, the success ratio was 1. The results show the mean and standard deviations obtained over 100 runs. **Top:** deterministic Cranfield benchmark. **Bottom:** stochastic Cranfield benchmark. In this benchmark, the actions had a success ratio of 60%. **Left:** the number of actions executed in each episode is shown. **Right:** the number of teacher demonstration requests is shown.

6.2.2. Experiment 4: Generalizing tasks

In this experiment, the decision maker had to complete different tasks. The changes in the task required new actions to be learned to solve problems that had not been presented previously. Three different variants of the Cranfield benchmark were used.

- Episodes 1 - 4: The standard Cranfield benchmark was used.
- Episodes 5 - 8: The task started with a peg in a horizontal position. The action to place pegs failed if they were not in a vertical position, and thus the peg had to be repositioned with a new action before placing it.
- Episodes 9 - 12: Initially, the separator was placed without any pegs. To complete the scenario successfully, the robot needed to place the pegs but they could only be placed when the separator was not in place, so it had to be removed first, which required that a new action was demonstrated.

Figure 8 shows the results obtained with the REX-D algorithm. Each time the task was changed, after executing a few actions, the system realized that it needed a new action and requested a demonstration from the teacher. As more episodes were added, although some changes were made to the task, the results became closer to the optimum. By contrast, when REX-D restarted after a problem change, numerous demonstrations and exploration actions had to be executed to complete the task because everything had to be learned again.

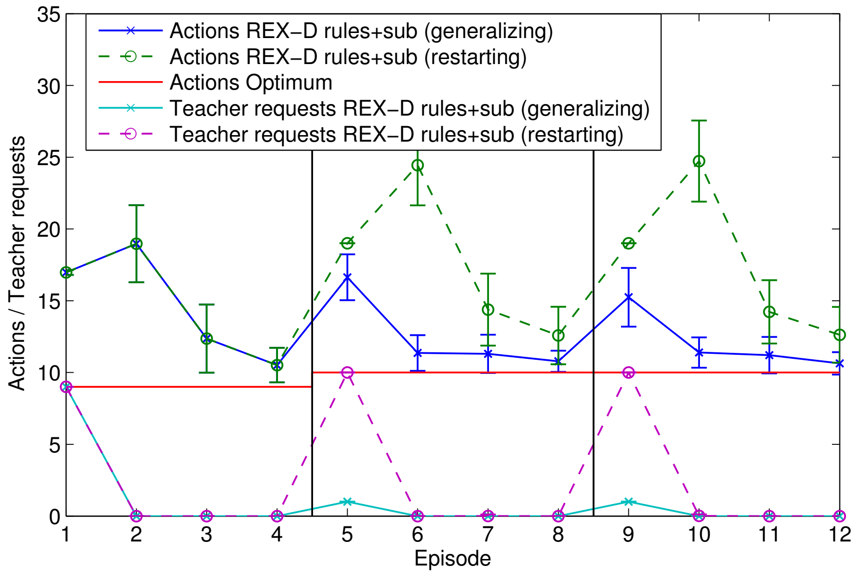

Figure 8: Experiment 4 (Generalizing tasks in the Cranfield benchmark): Three different Cranfield benchmark problems had to be completed, which are separated by vertical black lines. The optimal number of actions required to complete the task changed because the variants of the domain used demanded that an additional action was completed. Because there were no dead ends and a teacher was available in case of failure, the success ratio was 1. The results represent the means and standard deviations obtained from 50 runs. The number of actions performed and the number of demonstration requests are shown. **REX-D (generalizing)** started with no previous knowledge of the actions in the first episode, but it maintained its experience when the problem changed. **REX-D (restarting)** started with no previous action knowledge each time the problem changed.

6.2.3. Experiment 5: Goal changes

This experiment demonstrated the advantages of using subgoals. The decision maker started learning from scratch with the standard Cranfield benchmark assembly and a change was made in the scenario after four episodes, where the goal changed such that a new special front faceplate with no requirement for a pendulum had to be placed.

The REX-D algorithm without subgoals requested a demonstration from the teacher before placing any parts because it did not know how to place the new front faceplate, which was required to reach the goal. This new part required all the pegs and the separator to be placed in advance, so the teacher had to position them before teaching the new action. However, the use of subgoals allowed the robot to complete the known parts of the assembly, i.e., the pegs and the separator, before issuing a help request to the teacher to learn only the missing action.

Figure 9 shows the improvement obtained after adding subgoals. The REX-D algorithm with subgoals only requested one demonstration from the teacher, whereas the standard REX-D required that all the actions were executed.

7. Conclusions

Learning with real robots is very time consuming if actions take several seconds to complete. Therefore, learning algorithms should require as few executions as possible. We addressed this problem using relational RL with demonstrations. We combined the generalization capabilities of learning in relational domains, where the same type of objects have to be learned only once, with teacher demonstrations, thereby significantly reducing the number of exploration actions required, which is the longest part of the learning process.

In the proposed REX-D algorithm, we added demonstration requests to the relational RL algorithm REX. In addition to the faster learning, demonstrations allow the algorithm to learn actions as they are required by extending generalization to different tasks, and new actions can be added if necessary.

Another aspect of the proposed method is the use of action rules analysis to help the teacher, which minimizes the number of demonstrations required in a task. The improvements include providing guidance to the teacher so

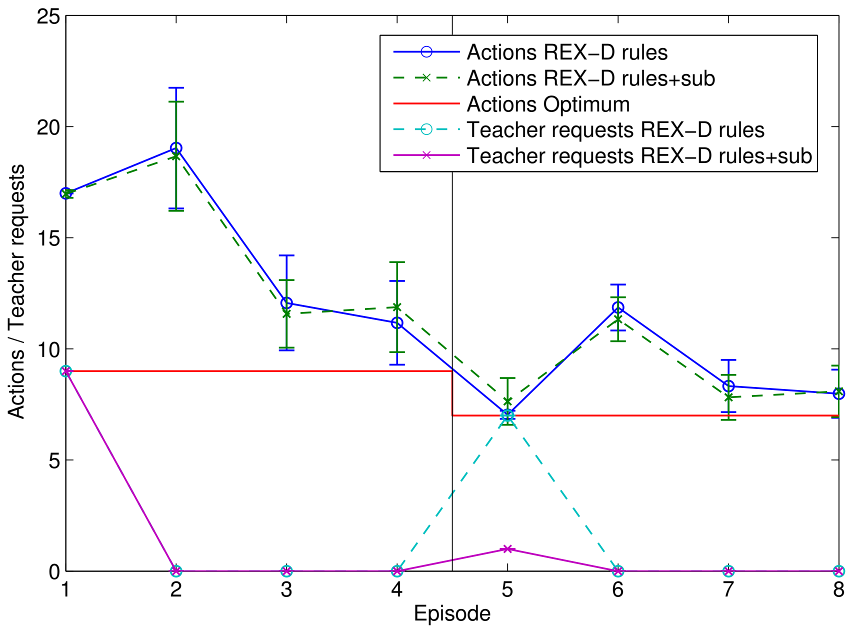

Figure 9: Experiment 5 (Goal changes): The standard Cranfield benchmark assembly is tackled during the first 4 episodes, and a special front faceplate that does not require a pendulum has to be placed during the last 4 episodes. The robot starts with no previous knowledge about the actions in the first episode. As there are no dead-ends and a teacher is available, the success ratio is 1. The results shown are the mean and standard deviations obtained over 50 runs. Both the amount of actions performed and the number of demonstration requests are shown.

only the unknown actions need to be demonstrated, checking for valid action rule alternatives before requesting new demonstrations, and using subgoals in tasks where demonstrating new actions may require the execution of previously known actions.

We performed several experiments to analyze the performance of the REX-D algorithm. Standard domains from the international planning competition were used to compare our approach with other relational RL methods, which demonstrated that significant improvements were obtained. Further experiments were also performed using the Cranfield benchmark encoded in a VR system, which provided the user with a similar experience to a real robot.

In conclusion, we proposed a relational RL algorithm with demonstration requests that improves the learning time while aiming to minimize the number of teacher interactions.

In future research, it would be useful to adapt REX-D to partially observable domains, where the planner and the learner would have to consider uncertainty. Approaches are available that learn partially observable domains instead of deterministic domains [26], but a new learning algorithm would have to be developed that covers both partially observable and stochastic actions. On the planning side, any partially observable MDP solver can be used.

8. Acknowledgements

We would like to thank Tobias Lang for providing the REX experimental results for IPPC problems and for answering various questions about the PRADA planner and REX, and to the Intellact consortium, specially Nils Wantia and Christian Schlette, for their support with the VR environment. We also thank the editors and reviewers for their helpful comments. This research was supported by EU Project IntellAct FP7-ICT2009-6-269959 and by the Spanish Ministry of Science and Innovation under project PAU+ DPI2011-27510. D. Martínez was also supported by the Spanish Ministry of Education, Culture, and Sport via a FPU doctoral grant (FPU12-04173).

References

[1] IntellAct Project, 2013. URL: `http://www.intellact.eu/`.

[2] T. Savarimuthu, D. Liljekrans, L.-P. Ellekilde, A. Ude, B. Nemec, N. Kruger, Analysis of human peg-in-hole executions in a robotic embodiment using uncertain grasps, in: Proc. of International Workshop on Robot Motion and Control, 2013, pp. 233–239.

[3] A. G. Buch, D. Kraft, J.-K. Kamarainen, H. G. Petersen, N. Kruger, Pose estimation using local structure-specific shape and appearance context, in: Proc. of International Conference on Robotics and Automation, 2013, pp. 2080–2087.

[4] J. Papon, T. Kulvicius, E. E. Aksoy, F. Wörgötter, Point cloud video object segmentation using a persistent supervoxel world-model, in: Proc. of International Conference on Intelligent Robots and Systems, 2013.

[5] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, F. Wörgötter, Learning the semantics of object–action relations by observation, International Journal of Robotics Research 30 (2011) 1229–1249.

[6] T. R. Savarimuthu, A. G. Buch, Y. Yang, S. Haller, J. Papon, D. Martínez, E. E. Aksoy, Manipulation monitoring and robot intervention in complex manipulation sequences, in: Proc. of RSS Workshop on Robotic Monitoring, 2014.

[7] J. Kober, J. Peters, Reinforcement learning in robotics: a survey (2012) 579–610.

[8] A. Cocora, K. Kersting, C. Plagemann, W. Burgard, L. De Raedt, Learning relational navigation policies, in: Proc. of International Conference on Intelligent Robots and Systems, 2006, pp. 2792–2797.

[9] D. Katz, Y. Pyuro, O. Brock, Learning to manipulate articulated objects in unstructured environments using a grounded relational representation, in: Proc. of Robotics: Science and Systems, 2008, pp. 254–261.

[10] C. Diuk, A. Cohen, M. L. Littman, An object-oriented representation for efficient reinforcement learning, in: Proc. of the International Conference on Machine learning, 2008, pp. 240–247.

[11] C. Rodrigues, P. Gérard, C. Rouveirol, Incremental learning of relational action models in noisy environments, in: Proc. of the International Conference on Inductive Logic Programming, Springer, 2011, pp. 206–213.

[12] T. Lang, M. Toussaint, K. Kersting, Exploration in relational domains for model-based reinforcement learning, Journal of Machine Learning Research 13 (2012) 3691–3734.

[13] L. Li, M. L. Littman, T. J. Walsh, A. L. Strehl, Knows what it knows: a framework for self-aware learning, Machine learning 82 (2011) 399–443.

[14] T. J. Walsh, I. Szita, C. Diuk, M. L. Littman, Exploring compact reinforcement-learning representations with linear regression, in: Proc. of the Conference on Uncertainty in Artificial Intelligence, 2009, pp. 591–598.

[15] T. J. Walsh, Efficient learning of relational models for sequential decision making, Ph.D. thesis, Rutgers, The State University of New Jersey, 2010.

[16] H. M. Pasula, L. S. Zettlemoyer, L. P. Kaelbling, Learning symbolic models of stochastic domains, Journal of Artificial Intelligence Research 29 (2007) 309–352.

[17] B. D. Argall, S. Chernova, M. Veloso, B. Browning, A survey of robot learning from demonstration, Robotics and Autonomous Systems 57 (2009) 469–483.

[18] Ç. Meriçli, M. Veloso, H. L. Akın, Multi-resolution corrective demonstration for efficient task execution and refinement, International Journal of Social Robotics 4 (2012) 423–435.

[19] T. J. Walsh, K. Subramanian, M. L. Littman, C. Diuk, Generalizing apprenticeship learning across hypothesis classes, in: Proc. of the International Conference on Machine Learning, 2010, pp. 1119–1126.

[20] P. Abbeel, A. Y. Ng, Exploration and apprenticeship learning in reinforcement learning, in: Proc. of the International Conference on Machine learning, 2005, pp. 1–8.

[21] W. B. Knox, P. Stone, Interactively shaping agents via human reinforcement: The tamer framework, in: Proc. of International Conference on Knowledge Capture, ACM, 2009, pp. 9–16.

[22] D. H. Grollman, O. C. Jenkins, Dogged learning for robots, in: Proc. of International Conference on Robotics and Automation, 2007, pp. 2483–2488.

[23] S. Chernova, M. Veloso, Interactive policy learning through confidence-based autonomy, Journal of Artificial Intelligence Research 34 (2009) 1–25.

[24] A. Agostini, C. Torras, F. Wörgötter, Integrating task planning and interactive learning for robots to work in human environments., in: Proc. of the International Joint Conference on Artificial Intelligence, 2011, pp. 2386–2391.

[25] M. Göbelbecker, T. Keller, P. Eyerich, M. Brenner, B. Nebel, Coming up with good excuses: What to do when no plan can be found., in: Proc. of the International Conference on Automated Planning and Scheduling, 2010, pp. 81–88.

[26] H. H. Zhuo, Q. Yang, D. H. Hu, L. Li, Learning complex action models with quantifiers and logical implications, Artificial Intelligence 174 (2010) 1540–1569.

[27] R. I. Brafman, M. Tennenholtz, R-max-a general polynomial time algorithm for near-optimal reinforcement learning, Journal of Machine Learning Research 3 (2003) 213–231.

[28] M. Kearns, S. Singh, Near-optimal reinforcement learning in polynomial time, Machine Learning 49 (2002) 209–232.

[29] J. Rossmann, C. Schlette, N. Wantia, Virtual reality in the loop - providing an interface for an intelligent rule learning and planning system, in: Proc. of ICRA workshop in Semantics, Identification and Control of Robot-Human-Environment Interaction, 2013, pp. 60–65.

[30] D. Martínez, G. Alenyà, P. Jiménez, C. Torras, J. Rossmann, N. Wantia, E. E. Aksoy, S. Haller, J. Piater, Active learning of manipulation sequences, in: Proc. of the International Conference on Robotics and Automation, 2014, pp. 5671–5678.

[31] N. Krüger, C. Geib, J. Piater, R. Petrick, M. Steedman, F. Wörgötter, A. Ude, T. Asfour, D. Kraft, D. Omrcen, et al., Object-action complexes: Grounded abstractions of sensory-motor processes, Robotics and Autonomous Systems 59 (2011) 740–757.

[32] G. Randelli, T. M. Bonanni, L. Iocchi, D. Nardi, Knowledge acquisition through human–robot multimodal interaction, Intelligent Service Robotics 6 (2013) 19–31.

[33] M. V. Menezes, L. N. de Barros, S. do Lago Pereira, Planning task validation, Proc. of the ICAPS workshop in Scheduling and Planning Applications (2012) 48–55.

[34] M. Helmert, The fast downward planning system, Journal of Artificial Intelligence Research 26 (2006) 191–246.

[35] IPPC, Sixth International Planning Competition, Uncertainty Part, 2008. URL: `http://ippc-2008.loria.fr/wiki/index.html`.

[36] L. Li, A unifying framework for computational reinforcement learning theory, Ph.D. thesis, Rutgers, The State University of New Jersey,

2009.
[37] A. Kolobov, Mausam, D. Weld, LRTDP vs. UCT for online probabilistic planning, in: Proc. of AAAI Conference on Artificial Intelligence, 2012, pp. 1786–1792.