V-MIN: Efficient Reinforcement Learning through Demonstrations and Relaxed Reward Demands

David Martínez and **Guillem Alenyà** and **Carme Torras**
Institut de Robòtica i Informàtica Industrial (CSIC-UPC)
C/ Llorens i Artigas 4-6. 08028 Barcelona, Spain
{dmartinez,galenya,torras}@iri.upc.edu

Abstract

Reinforcement learning (RL) is a common paradigm for learning tasks in robotics. However, a lot of exploration is usually required, making RL too slow for high-level tasks. We present V-MIN, an algorithm that integrates teacher demonstrations with RL to learn complex tasks faster. The algorithm combines active demonstration requests and autonomous exploration to find policies yielding rewards higher than a given threshold V_{min}. This threshold sets the degree of quality with which the robot is expected to complete the task, thus allowing the user to either opt for very good policies that require many learning experiences, or to be more permissive with sub-optimal policies that are easier to learn. The threshold can also be increased online to force the system to improve its policies until the desired behavior is obtained. Furthermore, the algorithm generalizes previously learned knowledge, adapting well to changes. The performance of V-MIN has been validated through experimentation, including domains from the international planning competition. Our approach achieves the desired behavior where previous algorithms failed.

1 Introduction

A very important challenge in robotics is the ability to perform new unknown tasks and adapt to changes. Learning has become a crucial procedure to get such versatility. However, learning high-level tasks with robots can be very time consuming, and complex tasks may prove intractable for most learning methods. We present V-MIN, a novel algorithm that learns new tasks from scratch, adapts quickly to unexpected changes, and can be tuned online to require the best policies or be more permissive and accept sub-optimal policies.

V-MIN is based on the RL algorithms R-MAX (Brafman and Tennenholtz 2003) and REX-D (Martínez et al. 2014). R-MAX obtains near-optimal policies with a polynomial number of samples in Markov Decision Processes (MDP). It follows the principle of "optimism under uncertainty" to encourage active exploration until good policies are learned. REX-D, which is based on E^3 (Kearns and Singh 2002), issues demonstration requests whenever it gets to a known state for which it cannot obtain a plan to reach the goal.

V-MIN introduces a new parameter V_{min} in R-MAX that defines the minimum expected value to be achieved. When this V_{min} value cannot be reached, active demonstrations are requested as in REX-D, until a policy that guarantees such value is obtained. For example, if the reward is based on the time spent, V_{min} can be the time limit to complete the task. The claims for this simple but powerful algorithm are:

- V-MIN is a more general approach in that it is able to tackle domains where policies may yield different values depending on the quality of the plans, while REX-D focuses on domains with just one possible goal value. V-MIN allows to choose the quality of the desired policies, spending just the required amount of time to learn policies that satisfy such quality requirements. Therefore, worse but easier policies can be learned when there are tight time constraints, while better policies can be learned when speed is not a very important issue.
- The V_{min} value can be changed online to ease the use of V-MIN, not requiring to set a proper V_{min} in advance. The teacher can slowly increment it until the desired behavior is obtained. Thus in-depth knowledge of the system is not required to use it, and the adaptation to new tasks with different value requirements becomes easier.
- It can adapt to changes and new tasks, reusing all its previously learned knowledge. Moreover, teacher demonstrations permit learning new actions as required.

The paper is organized as follows. First the related work and background needed are introduced. Afterwards, in Sec. 4 the V-MIN algorithm is presented. Finally, Sec. 5 shows a set of experiments measuring the system performance.

2 Related Work

We are focusing on robotics tasks in which actions take long to execute and the decision maker has long periods of time to process little input data. Model-based RL has been applied successfully to such high-level tasks (Kober and Peters 2012) A more compact representation of the model is obtained with relational RL approaches (Diuk, Cohen, and Littman 2008; Rodrigues, Gérard, and Rouveirol 2011), leading to improved results in domains containing several objects of the same type. Lang, Toussaint, and Kersting (2010) proposed the REX algorithm, that introduces

relational generalizations in the exploration-exploitation dilemma, minimizing the amount of exploration required.

Robotic tasks also face the problem of uncertainty in the actions. Within the KWIK framework (Li et al. 2011) a method to learn the action effects and probabilities has been proposed (Walsh 2010). However, as the problem of learning action effects is NP (Walsh 2010), a large number of samples is needed. More efficient heuristic algorithms exist, such as TEXPLORE (Hester and Stone 2013) and the learner proposed by Pasula, Zettlemoyer, and Kaelbling (2007), that obtain suboptimal but good solutions. We will use Pasula et al.'s learner, since it yields more compact and tractable models with relational rules, deictic references and noisy effects.

The integration of teacher demonstrations has already been tackled within RL-like algorithms to improve the learning process. Meriçli, Veloso, and Akın (2012) presented an algorithm where the teacher issues corrective demonstrations if the robot doesn't perform as desired. In (Walsh et al. 2010), the teacher reviews the planned actions, and whenever they are not optimal, a demonstration is shown. In these approaches the teacher has to intervene to improve the robot behavior. In comparison, our algorithm actively requests demonstrations when needed, releasing the teacher from having to monitor the system continuously.

Active demonstration requests have been included in algorithms with confidence thresholds (Grollman and Jenkins 2007; Chernova and Veloso 2009), which request demonstrations for a specific part of the state space whenever the system is not sure about the expected behavior. Our approach combines active demonstration requests with autonomous exploration. As teacher time is considered to be very valuable, demonstration requests should be limited and replaced with exploration whenever possible.

3 Background on Relational RL

This section we present the background needed for V-MIN.

Markov Decision Processes are used to formulate fully-observable problems with uncertainty. A finite MDP is a five-tuple $\langle S, A, T, R, \gamma \rangle$ where S is a set of states, A is the set of actions that an agent can perform, $T : S \times A \times S \rightarrow [0, 1]$ is the transition function, $R : S \times A \rightarrow \mathbb{R}$ is the reward function and $\gamma \in [0, 1)$ is the discount factor. The goal is to find a policy $\pi : S \rightarrow A$ which chooses the best action for each state 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.

We use a **relational representation** for states and rules, which is based on predicates and actions that can take objects as arguments to define their grounded counterparts. For example, an action *open(X)* with objects *box1* and *box2* could be grounded to *open(box1)* and *open(box2)*.

Noisy Deictic Rules (NDR) (Pasula, Zettlemoyer, and Kaelbling 2007) model the actions. The transition model T is represented with a set of NDR rules Γ which are defined:

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

where a_r is the action, $\phi_r(\chi)$ are the preconditions that define the context in which the rule is applicable, $\Omega_{r,i}$ are the effects defining the predicates that are changed in the state with probability $p_{r,i}$ when the rule is applied, $\Omega_{r,0}$ is the noisy effect that represents all other, unmodeled, rare and complex effects, and χ is the set of variables of the rule. A rule covers a state-action (s, a) pair when the rule represents the action a and its preconditions are satisfied in the state s. We represent a rule covering (s, a) with $r_{s,a}$. Moreover, when the reward is used as a function of a rule $R(r)$, it refers to the state-action pairs that the rule covers. A NDR rule represents one action, while each action may be represented by several rules. Each state-action pair (s, a) is covered by just one rule r as all the rules defining one action have disjoint preconditions $\phi_{r_{a,i}} \wedge \phi_{r_{a,j}} = \emptyset \, \forall i, j \mid i \neq j$.

A RL algorithm maximizes cumulative rewards, having to balance between exploration (exploring the model to obtain better policies) and exploitation (choose actions to maximize the value function according to the current policy). Lang, Toussaint, and Kersting (2012) proposed a context-based density count function to handle the exploration-exploitation dilemma. It takes advantage of the relational representation and reduces the number of samples before considering states as known by grouping them in contexts:

$$k(s, a) = \sum_{r \in \Gamma} |E(r)| I(r = r_{s,a}), \qquad (2)$$

where $|E(r)|$ is the number of experiences that cover the rule r with any grounding, and $I()$ is a function that takes the value 1 if the argument evaluates to true and 0 otherwise. Contexts visited more than a fixed threshold $k(s, a) \geq \zeta$ are considered to be known.

Teacher demonstrations are actively requested by the robot, for which standard human-robot interaction capabilities such as speech and visual information can be used. We consider that the teacher knows the optimal policy π^*, and that he tells the robot the name of the action, its parameters and how to perform it. Learning from Demonstration (Argall et al. 2009) can be used to learn new actions.

4 V-MIN Algorithm

In this section we present the novel V-MIN algorithm, that extends R-MAX (Brafman and Tennenholtz 2003). R-MAX plans in a model where unknown state-action pairs yield the maximum reward R_{max} to encourage active exploration. The REX algorithm (Lang, Toussaint, and Kersting 2012) was devised to apply the context-based density function in equation 2 to R-MAX, improving the generalization over different states. Although these 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. Therefore, state-action pairs needed to attain the best policy may not be visited, and thus their contexts (i.e. rules) not learned.

Similar to REX-D (Martínez et al. 2014), V-MIN uses teacher demonstrations to learn new actions and contexts, and improve its performance. Nevertheless, the integration

Algorithm 1 V-MIN

Input: Reward function R, confidence threshold ζ
Updates: Set of experiences E

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

Algorithm 2 Create T_{vmin}

Input: Model T, reward function R, confidence threshold ζ
Output: V-MIN model T_{vmin}

$T_{vmin} = T$
for all $r \in T_{vmin}$ **do**
  if $known(r, \zeta)$ **then**
    $R(r) = R_{max}$
  end if
  $\phi_r = \phi_r \wedge -$*finished*()
  for all $\Omega_{r,i} \in r$ **do**
    $\Omega_{r,i} = \Omega_{r,i} \wedge$ *started*()
  end for
end for
Add rule r_{vmin} =“TeacherRequest()” to T_{vmin}
$R(r_{vmin}) = V_{min}$

of demonstration requests is different: V-MIN uses the concept of V_{min}, which is the minimum expected value $V^{\pi}(s)$. If the planner cannot obtain a $V^{\pi}(s) \geq V_{min}$, 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 exploitation and exploration 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). Note that in the special case where a value $V(s) \geq V_{min}$ cannot be obtained anymore, which is a dead-end, the teacher signals that the episode has failed instead of demonstrating an action.

To include V_{min} in the model, an action “TeacherRequest()” is defined which leads to an absorbing state with a V_{min} value. The “TeacherRequest()” action asks for a demonstration from the teacher.

The following features are the most important in V-MIN:

- A high V_{min} forces the system to learn good policies at the cost of a higher number of demonstrations and exploration actions, 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 better policies.
- V-MIN can adapt well to new tasks. Context generalizations and teacher demonstrations are specially useful, as they permit transferring a lot of knowledge and learn new actions and contexts as required. Using Taylor and Stone (2009) transfer learning categorization, the new task can change the start state, the goal state, the reward function, and the number of objects. Moreover the action set can be increased but not decreased.

Algorithm

We are using Pasula, Zettlemoyer, and Kaelbling (2007)’s learner and the Gourmand planner (Kolobov, Mausam, and Weld 2012). They take into account the uncertainty in the actions but not in the observations. To tackle partially observable domains we would just have to integrate a learner and a planner that could handle them, without needing to perform any further changes in the V-MIN algorithm.

Algorithm 1 shows a pseudocode for V-MIN. This algorithm maintains a model T that represents the robot actions with a set of relational symbolic rules as described in Section 3. These rules are updated every iteration to adapt to newer experiences using Pasula, Zettlemoyer, and Kaelbling (2007)’s learner. The T_{vmin} model is then created to plan the action to execute, implicitly selecting exploration actions until all the relevant contexts are considered known, and requesting teacher demonstrations whenever a value $V^{\pi}(s) > V_{min}$ cannot be obtained.

Algorithm 2 shows how to create T_{vmin} as an extension of T that includes implicit exploration and teacher demonstration requests. First, to check whether a rule is known, equation 2 is used to count if the rule covers at least ζ experiences (note that equation 2 includes relational generalizations). As in the R-MAX algorithm, every unknown rule is set the maximum reward R_{max} to explore implicitly following the principle of “optimism under uncertainty”. Moreover, to include teacher demonstration requests, the “*-finished()*” predicate is added to the preconditions of all rules, the “*started()*” predicate to all rule effects, and finally, the special rule “TeacherRequest()” is added with reward V_{min}.

V-MIN requires that plans containing the special action “TeacherRequest()” must have a value of V_{min}. If “TeacherRequest()” were combined with other actions in the same plan, their combined value would be $V_{min} + V(s')$, and thus we are limiting “TeacherRequest()” to be the only action in a plan when selected. This behavior is obtained through the use of the two new predicates: *started* and *finished*. In sum, the following changes are made to T_{vmin}:

- Create a rule for the “TeacherRequest()” action with $\phi_{teacher} = -$*started*() and $\Omega_{teacher,1} =$ *finished*().
- Change all preconditions $\phi_r = \phi_r \wedge -$*finished*() $\forall r \in \Gamma$.
- Change all rule effects $\Omega_{r,i} = \Omega_{r,i} \wedge$ *started*() $\forall i, r \in \Gamma$.
- Change the state $s = s \wedge -$*started*() $\wedge -$*finished*().

“TeacherRequest()” requires the precondition -started(), and all other rules will have the started() predicate as effect, so “TeacherRequest()” can only be selected as the first action. Moreover, all rules require the -finished() precondition, and “TeacherRequest()” has finished() as effect, so

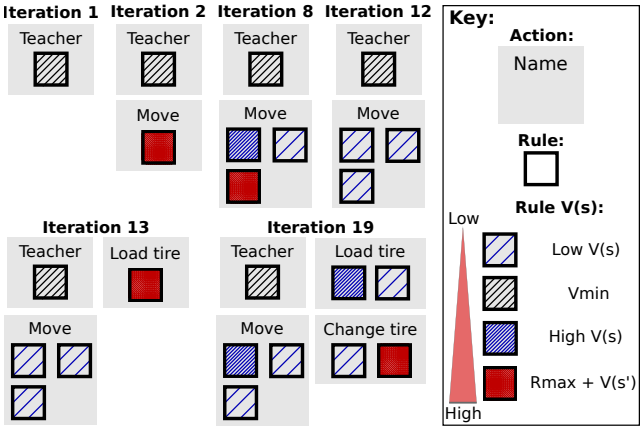


Figure 1: Example of the model generated by V-MIN in the Triangle Tireworld domain (Sec. 5). **Iteration 1:** the only action is requesting a demonstration. **Iteration 2:** a rule to model the new demonstrated action "Move" is available. It has a R_{max} reward as it hasn't been explored yet. **Iteration 8:** After several exploration actions, "Move" is modelled with 3 rules, 2 of which are already considered to be known. **Iteration 12-13:** a high value plan can't be found, so a new action is demonstrated. **Iteration 19:** new actions have been demonstrated and explored. Two rules that lead to high $V(s)$ are available. Note that although rules with R_{max} reward exist, they may not be reachable from the current state. For example, one "Change tire" rule requires a flat tire.

it can only be selected as the last action. Considering that "TeacherRequest()" can only be the first and last action, whenever selected it will be the only action in the plan.

Finally, Fig. 1 shows an example of how T_{vmin} evolves as actions get demonstrated and executed.

Rewards

The V-MIN algorithm requires the reward function to be known. Global rewards have to be defined explicitly in terms of predicates and the number of actions executed. Actions can also provide rewards which have to be either specified by the teacher during the demonstration, or perceived by the system, as V-MIN doesn't estimate rewards.

The value of V_{min} can be defined as a constant or as a function, and has to be updated as the system completes the task. When a discount factor γ exists, this discount has to be applied to the V_{min} value as the goal is approached.

In real-world applications, the teacher selects the V_{min} value. Although rewards can be defined with a measure that can be understood by the teacher (e.g. the time spent or the number of objects to be manipulated), selecting appropriate values for V_{min} can be complicated. As V_{min} can be modified online, the teacher can increase it until the system behaves as desired. This allows a teacher to use V-MIN without requiring in-depth knowledge about its reward function.

Performance Analysis

R-MAX is PAC-MDP (Strehl, Li, and Littman 2009), which means that it learns near-optimal behavior in MDPs with a polynomial number of samples $\widetilde{O}(|S|^2|A|/(\epsilon^3(1-\gamma)^6))$. The generalization to relational domains has also been proven to be PAC-MDP under the assumption that a KWIK learner and a near-optimal planner are used (Lang, Toussaint, and Kersting 2012). As the required set of actions is demonstrated during the learning process, and the exploration method is the same, the upper bound for the sample complexity of R-MAX holds also for V-MIN to get a policy π_{vmin} such that $V^{\pi_{vmin}}(s) \geq V_{min}$.

Nevertheless, the V-MIN algorithm performs much better in average situations. The use of generalizations and teacher demonstrations provides various structural assumptions that lead to an improved performance. The $|S|^2|A|$ product represents the $|S||A|$ state-action pairs that can be executed, which can lead to $|S|$ different states. Applying V-MIN to standard domains reduces this $|S|^2|A|$ number greatly:

- Standard domains usually have a small number of effects for each state-action pair, and furthermore, NDR rules permit modelling very rare effects as noise, reducing a $|S|$ factor to a small constant K_Ω that is the maximum number of effects that a rule may have in the domain.
- Learning the $|S||A|$ state-action pairs can be reduced to a complexity proportional to $\log_2(|S|)|\Gamma|$. Applying equation 2, the exploration is carried out over $|\Gamma|$ state-action contexts (where Γ are the rules needed to model the system). However, rule contexts are not provided, and finding them is proportional to $|S||A|$ as the system doesn't know which state-action pairs form them (Walsh 2010). Using the V-MIN algorithm, the teacher demonstrations provide positive examples of the relevant contexts. As the preconditions of a rule (which define the contexts) are a conjunction of predicates, only predicates in the state s_d where the demonstration was executed may be in the rule preconditions. Thus, testing every one of these predicates to see if they are really a part of the preconditions results in requiring $|s_d| = \log_2(|S|)$ samples for each context.

In conclusion, the expected sample complexity of V-MIN is proportional to $K_\Omega \log_2(|S|)|\Gamma|/(\epsilon^3(1-\gamma)^6)$. Moreover, V-MIN can afford larger values of ϵ and γ, compensating the errors with a slightly increased number of demonstrations.

5 Experimental Results

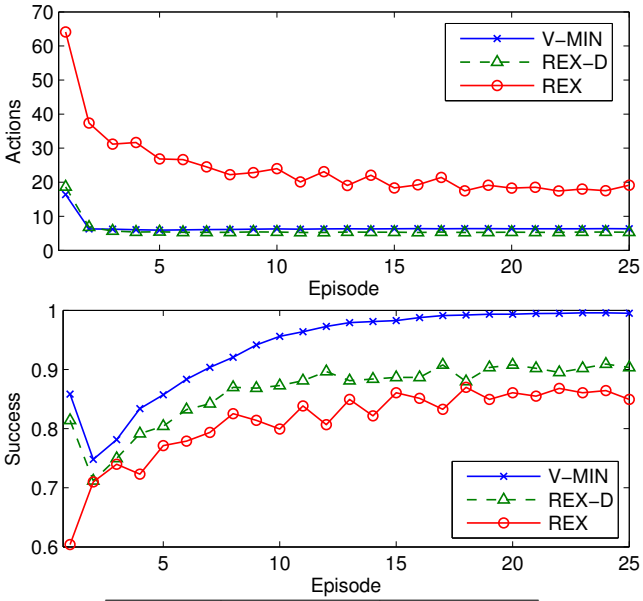
This section analyzes V-MIN performance:

1. Comparing V-MIN with REX-D (Martínez et al. 2014) and REX (Lang, Toussaint, and Kersting 2012).
2. Using different values of V_{min} and the increasing V_{min}.
3. Adaptation to changes in the tasks.

Episodes were limited to 100 actions before considering them as a failure. The exploration threshold was set to $\zeta = 3$, which yielded good results. V-MIN and REX-D start with no previous knowledge, while REX has the list of actions available as it cannot learn them from demonstrations.

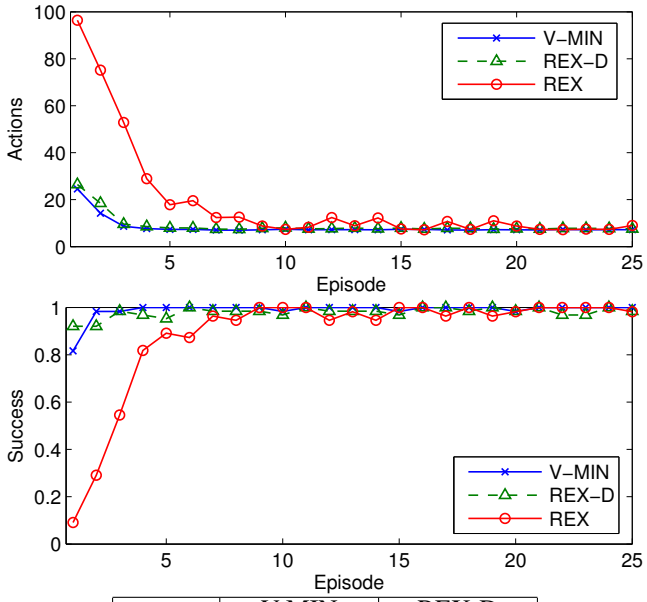
Experiment 1: Performance Comparison

In this experiment we compared V-MIN with REX-D and the R-MAX variant of REX in two problems of the International Probabilistic Planning Competition (2008). The V_{min}



	V-MIN	REX-D
Teacher	3.87 ± 1.53	2.65 ± 1.19

Figure 2: Experiment 1 (Triangle Tireworld problem 1): Means over 250 runs. **Top:** the number of actions executed in each episode. **Middle:** the success ratio. **Bottom:** the total number of teacher demonstrations requested.



	V-MIN	REX-D
Teacher	7.08 ± 2.65	5.67 ± 1.32

Figure 3: Experiment 1 (Exploding Blocksworld problem 5): Means over 100 runs. Same layout as in Fig. 2.

value was initialized with values slightly below the optimum.

Triangle Tireworld In this domain a car has to move to its destination, but it has a probability of 0.35 of getting a flat tire while it moves. The actions available are: "Move" to go to an adjacent position, "Load Tire" to load a spare tire into the car if there are any in the current location, and "Change Tire" to change a flat tire with a spare tire. The main difficulty are the dead-ends when the agent gets a flat tire and has no spare tires available. A safe longer path exists with spare tires while the shortest path doesn't have any.

Figure 2 shows that V-MIN always learns the safe policy, as it is the only one with $V(s) > V_{min}$. However, the other approaches fail to learn the safe path when no flat tires are obtained during the first episodes: once they consider the short path as known, they never try to find better paths. In contrast V-MIN looks for alternatives with higher values after discovering the dead-ends in the short path. Therefore, V-MIN requests a few more demonstrations but finds the best path in terms of safety. Note that the large number of action executed by REX is due to its inability to detect dead-ends, so it keeps trying up to the 100 action limit.

This problem has 3 actions and 5 predicates with 43 and 50 respective groundings. Based on the performance analysis in Sec. 4, using a discount factor $\gamma = 0.9$, an accuracy parameter $\epsilon = 0.1$ and an exploration threshold $\zeta = |S|/(\epsilon^2(1-\gamma)^4) = 3$, the upper bound for the R-MAX sample complexity is proportional to $|S||A|\zeta/(\epsilon(1-\gamma)^2) = 50 \cdot 43 \cdot 3/(0.1(1-0.9)^2) = 2.2 \cdot 10^6$. Meanwhile, the estimated V-MIN sample complexity is $K_\Omega \log_2(|S|)|\Gamma|\zeta/(|S| \cdot \epsilon(1-\gamma)^2) = 2\log_2(50)6 \cdot 3/(50 \cdot 0.1(1-0.9)^2) = 4 \cdot 10^3$, which is much lower. Note that V-MIN can use such a small ζ as it compensates the lack of exploration with a few extra demonstrations. Experimental results were better than this worst case, requiring just 14 exploration actions and 3.87 demonstrations.

Exploding Blocksworld This domain is an extension of the well-known Blocksworld in that it includes dead-ends. When a block is moved, it has a probability of exploding and destroy the object beneath, making it unusable.

The results obtained are displayed in Fig. 3. This is a good example of a domain with a simple goal. Once the needed actions are demonstrated, exploration is enough to learn policies that yield the highest value. As REX-D already performs well, V-MIN obtains similar results. V-MIN requests a slightly larger number of demonstrations, and in exchange has a slightly higher success ratio and explores less. REX also converges to the optimal policy, but requires an average of 250 extra exploration actions compared to the few demonstrations requested by the other two algorithms.

Experiment 2: V-MIN Adaptation Capabilities

This experiment was performed in the *Table Clearing* domain. It consists in taking plates, cups and forks from a table to the kitchen. Moving to the kitchen is a costly action, and therefore it is recommended to stack the objects before taking them to the kitchen, minimizing the time spent.

The actions *putPlateOn*, *putCupOn* and *putForkOn* place the corresponding object on another one. If a plate is placed on a cup or fork, or a cup is placed on a fork there is a prob-

ability that it may fall and break. There are another 3 inverse actions to place each of the objects back on the table, and one action *moveStacks* to take the stacks to the kitchen.

The reward is based on the time spent, but there is also a high penalty for each broken object. Each action reduces the reward by 0.05, while the *moveStacks* action reduces it by 1 for every stack, by 5 for every broken object, and increases it by 5 when it finishes clearing the table. V-MIN requires the reward function to be known, and therefore the reward yielded by *moveStacks* is known since the beginning, while all other actions are just demonstrated.

Increasing V-MIN In this experiment the table has 3 plates and a cup on it, and there is one fork placed on one of the plates. The performance of V-MIN is evaluated for different values of V_{min}, and the increasing V_{min}, that starts at $V_{min} = 1.2$ and increases by one at the episodes 21 and 41.

Figure 4 shows the results. REX-D is goal-driven and thus it just takes the stacks to the kitchen without trying to improve its policy. In contrast, V-MIN learns until it surpasses the V_{min} threshold. Higher V_{min} values increase the problem complexity, requiring more exploration and demonstrations, as extra actions are needed to get such values. Note that the learning is concentrated during the first episodes.

Finally, the increasing V_{min} learns incrementally at the pace requested by the teacher. V-MIN maintains its previous knowledge, and thus the exploration and demonstrations required to incrementally get to a certain V_{min} is similar to learning that V_{min} directly. When no new dead-ends have to be learned, as happens when changing from $V_{min} = 2.2$ to 3.2, the value also smoothly increases without significant degradations due to exploration.

Task Changes This experiment shows the V-MIN capabilities to adapt to changes in the scenario. Knowledge is maintained in all configurations but the "$V_{MIN} = 3.2$ reset" one, which starts each problem with no previous knowledge. Three different problems are solved:

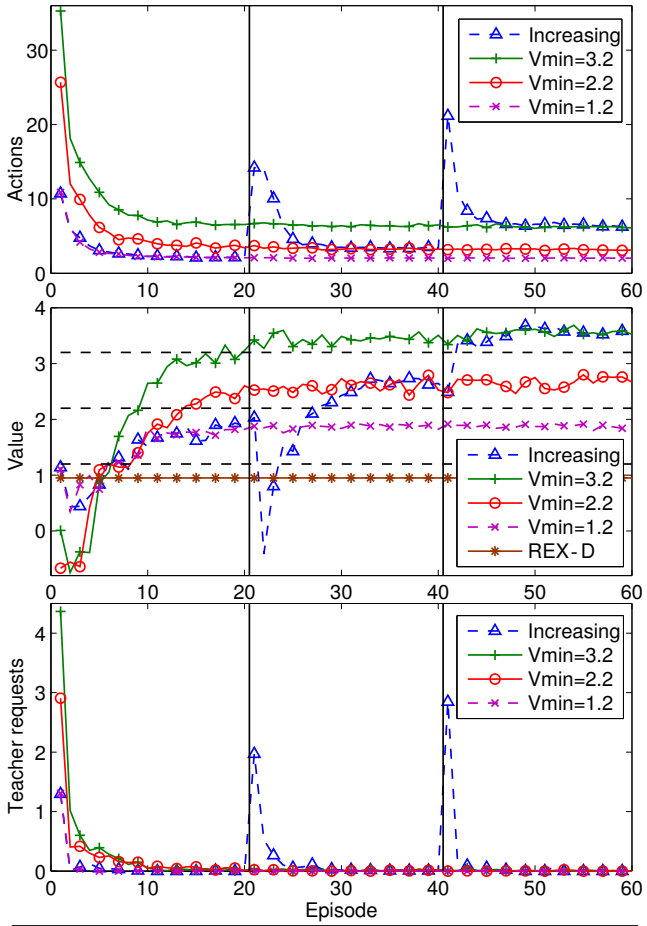
1. The table has 3 plates and a cup on it.
2. Same as 1, but the cup is placed on a plate instead.
3. Same as 1, but there is a new fork on one of the plates.

Figure 5 shows the results. For the first two problems, the set of actions needed to get a value of $V^{\pi}(s) \geq 1.2$ is already enough to find policies that get $V^{\pi}(s) \geq 2.2$, and thus both yield the same results. The adaptation to problem 2 is very smooth, as the only change is that a cup starts placed on a plate, and just in the case of $V_{min} = 3.2$ a new action "removeCup" is required if it wasn't already learned. Note that "$V_{MIN} = 3.2$ reset" has to learn everything again with a slightly more complicated initial state, resulting in a larger number of actions and demonstrations.

In contrast, problem 3 adds a new type of object, requiring to learn new actions and refine previous rules. V-MIN adapts rather well in just a few episodes, meanwhile "$V_{MIN} = 3.2$ reset" needs a much larger number of actions.

6 Conclusions and Future Work

We have proposed the novel V-MIN algorithm that learns policies satisfying the teacher quality requirements. The

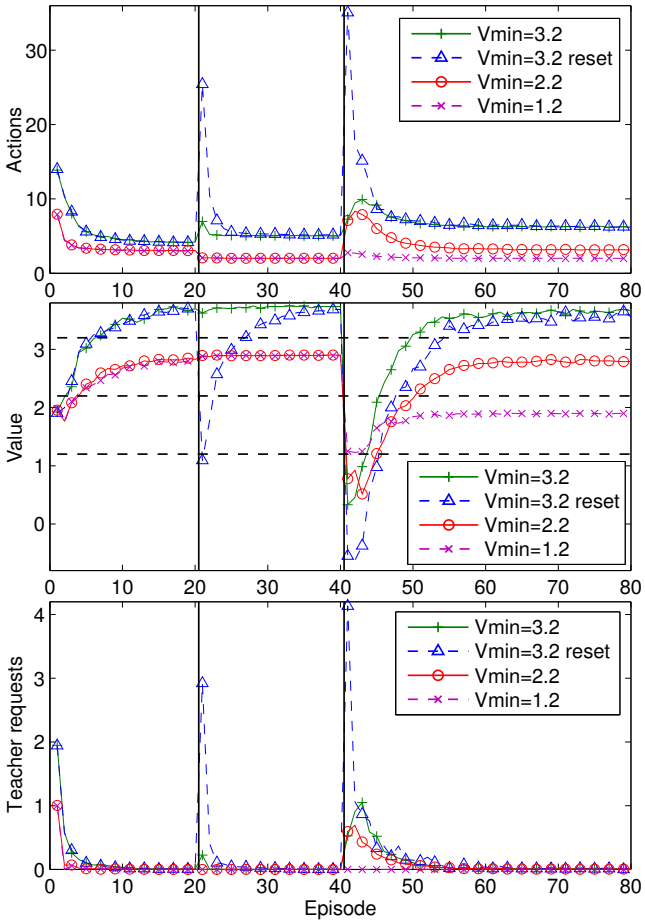


$V_{MIN} = 1.2$	$V_{MIN} = 2.2$	$V_{MIN} = 3.2$	Increasing
1.39 ± 0.62	5.34 ± 2.21	8.09 ± 2.50	8.18 ± 2.01

Figure 4: Experiment 2 (Table Clearing, standard): Means over 100 runs. The vertical black lines indicate that V_{min} was increased by 1. **Top:** actions executed per episode. **Upper middle:** value per episode. **Lower middle:** teacher demonstrations per episode. **Bottom:** total number of demonstrations requested.

V_{min} parameter gives the option of either learning good policies at the cost of a large number of demonstrations and exploration actions, or being more permissive with worse policies that are easier to learn. Even if the right V_{min} is not selected in the beginning, it can be tuned online until the teacher expectations are satisfied. Moreover, V-MIN is a general algorithm that can be applied to a wide set of scenarios, adapts very well to changes, and learns new actions as required. Experimental results show that V-MIN improves previous approaches in different scenarios and configurations. It can obtain better policies, while keeping the number of experiences and demonstrations low.

As future work, the performance of V-MIN in partially-observable domains can be analyzed. Furthermore, V-MIN can be extended to learn the reward function, although it would increase the amount of experiences required.



$V_{MIN} = 1.2$	$V_{MIN} = 2.2$	$V_{MIN} = 3.2$	Reset
1.13 ± 0.39	4.46 ± 1.34	8.68 ± 2.19	16.6 ± 5.73

Figure 5: Experiment 2 (Table Clearing, task changes): Means over 500 runs. The vertical black lines indicate the problem changes. Same layout as in Fig. 4.

Acknowledgements

This work is supported by CSIC project MANIPlus 201350E102 and by the Spanish Ministry of Science and Innovation under project PAU+ DPI2011-27510. D. Martínez is also supported by the Spanish Ministry of Education, Culture and Sport via a FPU doctoral grant (FPU12-04173).

References

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5):469–483.

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

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

Diuk, C.; Cohen, A.; and Littman, M. L. 2008. An object-oriented representation for efficient reinforcement learning. In *Proc. of the International Conference on Machine learning*, 240–247.

Grollman, D. H., and Jenkins, O. C. 2007. Dogged learning for robots. In *Proc. of the International Conference on Robotics and Automation*, 2483–2488.

Hester, T., and Stone, P. 2013. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine learning* 90(3):385–429.

IPPC. 2008. Sixth International Planning Competition, Uncertainty Part.

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

Kober, J., and Peters, J. 2012. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research* 579–610.

Kolobov, A.; Mausam; and Weld, D. S. 2012. Lrtdp versus uct for online probabilistic planning. In *Proc. of the AAAI Conference on Artificial Intelligence*.

Lang, T.; Toussaint, M.; and Kersting, K. 2010. Exploration in relational worlds. In *Proc. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 178–194.

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

Li, L.; Littman, M. L.; Walsh, T. J.; and Strehl, A. L. 2011. Knows what it knows: a framework for self-aware learning. *Machine learning* 82(3):399–443.

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

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

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research* 29(1):309–352.

Rodrigues, C.; Gérard, P.; and Rouveirol, C. 2011. Incremental learning of relational action models in noisy environments. In *Proc. of the International Conference on Inductive Logic Programming*, 206–213.

Strehl, A. L.; Li, L.; and Littman, M. L. 2009. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research* 10:2413–2444.

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

Walsh, T. J.; Subramanian, K.; Littman, M. L.; and Diuk, C. 2010. Generalizing apprenticeship learning across hypothesis classes. In *Proc. of the International Conference on Machine Learning*, 1119–1126.

Walsh, T. J. 2010. *Efficient learning of relational models for sequential decision making*. Ph.D. Dissertation, Rutgers, The State University of New Jersey.