

A method for the online construction of the set of states of a Markov Decision Process using Answer Set Programming

Leonardo A. Ferreira¹, Reinaldo A. C. Bianchi², Paulo E. Santos², Ramon Lopez de Mantaras³

¹Universidade Metodista de São Paulo, São Bernardo do Campo, Brazil.

²Centro Universitário FEI, São Bernardo do Campo, Brazil.

³IIIA-CSIC, Bellaterra, España.

leonardo.ferreira@metodista.br, {rbianchi,psantos}@fei.edu.br, mantaras@iiia.csic.es

Abstract

Non-stationary domains, that change in unpredictable ways, are a challenge for agents searching for optimal policies in sequential decision-making problems. This paper presents a combination of Markov Decision Processes (MDP) with Answer Set Programming (ASP), named *Online ASP for MDP* (oASP(MDP)), which is a method capable of constructing the set of domain states while the agent interacts with a changing environment. oASP(MDP) updates previously obtained policies, learnt by means of Reinforcement Learning (RL), using rules that represent the domain changes observed by the agent. These rules represent a set of domain constraints that are processed as ASP programs reducing the search space. Results show that oASP(MDP) is capable of finding solutions for problems in non-stationary domains without interfering with the action-value function approximation process.

1 Introduction

A key issue in Artificial Intelligence (AI) is to equip autonomous agents with the ability to operate in changing domains by adapting the agents' processes at a cost that is equivalent to the complexity of the domain changes. This ability is called *elaboration tolerance* [McCarthy, 1987; McCarthy, 1998]. Consider, for instance, an autonomous robot learning to navigate in an unknown environment. Unforeseen events may happen that could block passages (or open previously unavailable ones). The autonomous agent should be able to find new solutions in this changed domain using the knowledge previously acquired plus the knowledge acquired from the observed changes in the environment, without having to operate a complete code-rewriting, or start a new cycle of domain-exploration from scratch.

Reinforcement Learning (RL) is an AI framework in which an agent interacts with its environment in order to find a sequence of actions (a policy) to perform a given task [Sutton and Barto, 2015]. RL is capable of finding optimal solutions to Markov Decision Processes (MDP) without assuming total information about the problem's domain. However, in spite of having the optimal solution to a particu-

lar task, a RL agent may still perform poorly on a new task, even if the latter is similar to the former [Garnelo *et al.*, 2016]. Therefore, Reinforcement Learning alone does not provide elaboration-tolerant solutions. Non-monotonic reasoning can be used as a tool to increase the generality of domain representations [McCarthy, 1987] and may provide the appropriate element to build agents more adaptable to changing situations. In this work we consider Answer Set Programming (ASP) [Gelfond and Lifschitz, 1988; Lifschitz, 2002], which is a declarative non-monotonic logic programming language, to bridge the gap between RL and elaboration tolerant solutions. The present paper tackles this problem by introducing a novel algorithm: *Online ASP for MDP* (oASP(MDP)), that updates previously obtained policies, learned by means of Reinforcement Learning (RL), using rules that represent the domain changes as observed by the agent. These rules are constructed by the agent in an *online* fashion (i.e., as the agent perceives the changes) and they impose constraints on the domain states that are further processed by an ASP engine, reducing the search space. Tests performed in non-stationary non-deterministic grid worlds show that, not only oASP(MDP) is capable of finding the action-value function for an RL agent and, consequently, the optimal solution, but also that using ASP does not hinder the performance of a learning agent and can improve the overall agent's performance.

To model an oASP(MDP) learning agent (Section 3), we propose the combination of Markov Decision Processes and Reinforcement Learning (Section 2.1) with ASP (Section 2.2). Tests were performed in two different non-stationary non-deterministic grid worlds (Section 4), whose results show a considerable increase in the agent's performances when compared with a RL base algorithm, as presented in Sections 4.1 and 4.2.

2 Background

This section introduces Markov Decision Processes (MDP), Reinforcement Learning (RL) and Answer Set Programming (ASP) that are the foundations of the work reported in this paper.

2.1 MDP and Reinforcement Learning

In a sequential decision making problem, an agent is required to execute a series of actions in an environment in order to

find the solution of a given problem. Such sequence of actions, that forms a feasible solution, is known as a policy (π) which leads the agent from an initial state to a goal state [Bellman, 1957; Bellman and Dreyfus, 1971]. Given a set of feasible solutions, an optimal policy π^* can be found by using Bellman’s Principle of Optimality [Bellman and Dreyfus, 1971], which states that “an optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision”; π^* can be defined as the policy that maximises/minimises a desired reward/cost function.

A formalisation that can be used to describe sequential decision making problems is a Markov Decision Process (MDP) that is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where:

- \mathcal{S} is the set of states that can be observed in the domain;
- \mathcal{A} is the set of actions that the agent can execute;
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the transition function that provides the probability of, being in $s \in \mathcal{S}$ and executing $a \in \mathcal{A}$, reaching the future state $s' \in \mathcal{S}$;
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function that provides a real number when executing $a \in \mathcal{A}$ in the state $s \in \mathcal{S}$ and observing $s' \in \mathcal{S}$ as the future state.

One method that can be used to find an optimal policy for MDPs, which does not need *a priori* knowledge of the transition and reward functions, is the reinforcement learning model-free off-policy method known as Q-Learning [Watkins, 1989; Sutton and Barto, 2015].

Given an MDP \mathcal{M} , Q-Learning learns while an agent interacts with its environment by executing an action a_t in the current state s_t and observing both the future state s_{t+1} and the reward r_{t+1} . With these observations, Q-Learning updates an action-value function $Q(s, a)$ using

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot (r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)),$$

where α is the learning rate and γ is a discount factor. By using these reward values to approximate a $Q(s, a)$ function that maps a real value to pairs of states and actions, Q-Learning is capable of finding π^* which maximises the reward function. Since Q-Learning is a well-known and largely used RL method, we omit its detailed description here, which can be found in [Watkins, 1989; Sutton and Barto, 2015].

Although Q-Learning does not need information about \mathcal{T} and \mathcal{R} , it still needs to know the set \mathcal{S} of states before starting the interaction with the environment. For finding this set, this work uses Answer Set Programming.

2.2 Answer Set Programming

Answer Set Programming (ASP) is a declarative non-monotonic logic programming language that has been successfully used for NP-complete problems such as planning [Lifschitz, 2002; Zhang *et al.*, 2015; Yang *et al.*, 2014].

An ASP rule is represented as

$$A \leftarrow L_1, L_2, \dots, L_n \quad (1)$$

where A is an atom (the head of the rule) and the conjunction of literals L_1, L_2, \dots, L_n is the rule’s body.

An ASP program Π is a set of rules in the form of Formula 1. ASP is based on the stable model semantics of logic programs [Gelfond, 2008]. A stable model of Π is an interpretation that makes every rule in Π true, and is a minimal model of Π . ASP programs are executed by computing stable models, which is usually accomplished by inference engines called answer set solvers [Gelfond, 2008].

Two important aspects of ASP are its third truth value for *unknown*, along with *true* and *false*, and its two types of negation: strong (or classical) negation and weak negation, representing *negation as failure*. As it is defined over stable models semantics, ASP respects the rationality that *one shall not believe anything one is not forced to believe* [Gelfond and Lifschitz, 1988].

Although ASP does not allow explicit reasoning with or about probabilities, ASP’s choice rules are capable of generating distinct outcomes for the same input. I.e., given a current state s and an action a , it is possible to describe in an ASP logic program states s_1 , s_2 and s_3 as possible outcomes of executing a in s as “ $1\{s_1, s_2, s_3\}1 :- a, s.$ ”. Such choice rules can be read as “given that s and a are true, choose at least one and at maximum of one state from s_1 , s_2 and s_3 ”. Thus, the answer sets $[s, a, s_1]$, $[s, a, s_2]$ and $[s, a, s_3]$ represent the possible transitions that are the effects of executing action a on state s .

This work assumes that for each state $s \in \mathcal{S}$ there is an ASP logic program with choice rules describing the consequences of each action $a \in \mathcal{A}_s$ (where $\mathcal{A}_s \subseteq \mathcal{A}$ is the set of actions for the state s). ASP programs can also be used to represent domain constraints: the allowed or forbidden states or actions. In this context, to find a set \mathcal{S} of an MDP and its $Q(s, a)$ function is to find every answer set for every state that the agent is allowed to visit, i.e. every allowed transition for each state-action pair. In this paper ASP is used to find the set of states \mathcal{S} of an MDP and Q-Learning is used to approximate $Q(s, a)$ without assuming prior knowledge of \mathcal{T} and \mathcal{R} . The next section describes this idea in more details.

3 Online ASP for MDP: oASP(MDP)

Given sets \mathcal{S} and \mathcal{A} of an MDP, a RL method M can approximate an action-value function $Q(s, a)$. If \mathcal{S} is constructed state by state while the agent is interacting with the world, M is still able to approximate $Q(s, a)$, as it only uses the current and past states for that. By using choice rules in ASP, it is possible to describe a transition $t(s, a, s')$ in the form $1\{s'\}1 :- a$ for each action $a \in \mathcal{A}_s$ and each state $s \in \mathcal{S}$. By describing possible transitions for each action in each state as a logic program, an ASP engine can be used to provide a set of observed states \mathcal{S}_o , a set of actions \mathcal{A}_s for each state and, finally, an action-value function defined from the interaction with the environment, that can be used to further operate in this environment. This is the essence of the oASP(MDP) method, represented in Algorithm 1.

In order to illustrate oASP(MDP) (Algorithm 1), let’s consider the grid world in Figure 1, and an oASP(MDP) agent, initially located at the state “S” (blue cell in the grid), that is capable of executing any action in the following set: $\mathcal{A} = \{$

1 Algorithm: oASP(MDP)

Input: The set of actions \mathcal{A} , an action-value function approximation method M and a number of episodes n .

Output: The approximated $Q(s, a)$ function.

```

2 Initialize the set of observed states  $\mathcal{S}_o = \emptyset$ 
3 while number of episodes performed is less than  $n$  do
4   repeat
5     Observe the current state  $s$ 
6     if  $s \notin \mathcal{S}_o$  then
7       Add  $s$  to the set of states  $\mathcal{S}_o$ .
8       Choose and execute a random action  $a \in \mathcal{A}$ .
9       Observe the future state  $s'$ .
10      Update state  $s$  logic program with observed
        transition adding a choice rule.
11      Update  $Q(s, a)$ 's description by finding
        every answer set for each state  $s$  added to
         $\mathcal{S}_o$  in this episode.
12    else
13      Choose an action  $a \in \mathcal{A}$  as defined by  $M$ .
14      Execute the chosen action  $a$ .
15      Observe the future state  $s'$ .
16    end
17    Update  $Q(s, a)$ 's value as defined by  $M$ .
18    Update the current state  $s \leftarrow s'$ .
19  until the end of the episode
20 end

```

Algorithm 1: The oASP(MDP) Algorithm.

go up, go down, go left, go right. This grid world has walls (represented by the letter “W”), that are cells where the agent cannot occupy and through which it is unable to pass. If an agent moves toward a wall (or toward an external border of the grid) it stays at its original location. When the interaction with the environment starts, the agent has only information about the set of actions \mathcal{A} . The set of observed states \mathcal{S}_o is initially empty.

At the beginning of the agent’s interactions with the environment, the agent observes the initial state s_0 and verifies if it is in \mathcal{S}_o . Since $s_0 \notin \mathcal{S}_o$, the agent adds s_0 to \mathcal{S}_o (line 7 of Algorithm 1) and executes a random action, let this action be *go up*. As a consequence of this choice, the agent moves to a new state s_1 (the cell above S) and receives a reward r_0 . At this moment, the agent has information about the previous state, allowing it to write the choice rule “ $1\{s_1\}1 : -s_0, go\ up$ ” as an ASP logic program. In this first interaction, the only answer set that can be found for this choice rule is “ $[s_0, go\ up, s_1]$ ”. With this information the agent can initialize a $Q(s_0, go\ up)$ and update this value using the reward r_0 (line 17).

After this first interaction, the agent is in the state s_1 (the cell above S). Again, this is an unknown state ($s_1 \notin \mathcal{S}_o$), thus, as with the previous state, the agent adds s_1 to \mathcal{S}_o , chooses a random action, let it be *go up* again, and executes this action in the environment. By performing *go up* in this state, the agent hits a wall and stays in the same state. With

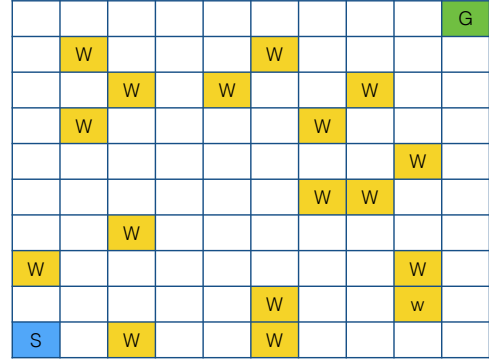


Figure 1: Example of a randomly generated grid world.

this observation, the agent writes the choice rule “ $1\{s_1\}1 : -s_1, go\ up$ ” and updates the value of $Q(s_1, go\ up)$ using the received reward r_1 .

Since the agent is in the same state as in the previous interaction, it knows the consequence of the action *go up* in this state, but has no information about any other actions for this state. At this moment, the agent selects an action using the action-selection function defined by the learning method M and executes it in the environment. For example, let it choose *go down*, returning to the blue cell (S). The state s_1 has now two choice rules: “ $1\{s_1\}1 : -s_1, go\ up$ ” and “ $1\{s_0\}1 : -s_1, go\ down$ ” which lead to the answer sets “ $[s_1, go\ up]$ ” and “ $[s_1, go\ down, s_0]$ ” respectively. Once again, the agent updates the $Q(s_1, go\ up)$ function using the method described in M with the reward r_2 received. After this transition, the agent finds itself once again in the initial state and continues the domain exploration just described. If, for example, the agent chooses to execute the action *go up* again, but due to the non-deterministic nature of the environment, the agent goes to the state on the right of the blue square, then a new state s_2 is observed and the choice rule for the previous state is updated to “ $1\{s_1, s_2\}1 : -s_0, go\ up$ ”. The answer sets that can be found considering this choice rule are “ $[s_0, go\ up, s_1]$ ” and “ $[s_0, go\ up, s_2]$ ”. With the reward r_3 received, the agent updates the value of $Q(s_0, go\ up)$.

The learning process of oASP(MDP) continues according to the chosen action-value function approximation method (from line 12 onwards). After a number of interactions with the environment, the oASP(MDP) agent has executed every possible action in every state that is possible to be visited and has the complete environment description. Note that this method excludes states of the MDP that are unreachable by the agent, which improves the efficiency of a RL agent in cases that the environment imposes state constraints (as we shall see in the next section).

The next section presents the tests applied to evaluate oASP(MDP) implemented with Q-Learning as the action-value function approximation method M .

4 Tests and Results

The oASP(MDP) algorithm was evaluated with tests performed in non-deterministic, non-stationary, grid-world domains. Two test sets were considered where, in each set, one

of the following domain variables was randomly changed: *the number and location of walls in the grid* (first test, Section 4.1), and *the transition probabilities* (second test, Section 4.2).

Four actions were allowed in the test domains considered in this work: *go up*, *go down*, *go left* and *go right*. Each action has a predefined probability of conducting the agent in the desired direction and also for moving the agent to an orthogonal (undesired) location. The transition probability for each action depends on the grid world and will be defined for each test, as described below. In all tests, the initial state was fixed at the lower-leftmost square (e.g., cell ‘S’ in Fig. 1) and the goal state fixed in the upper-rightmost square (e.g., cell ‘G’ in Fig. 1).

In the test domains, walls were distributed randomly in the grid as obstacles. For each grid, the ratio of walls per grid size is defined. The initial and goal states are the only cells that do not accept obstacles. Wall’s placement in the grid changed at the 1000th and 2000th episodes during each test trial. An example of a grid used in this work is shown in Figure 1.

Results show the data obtained from executing Q-Learning and oASP(MDP) (with Q-Learning as the action-value function approximation method) in the same environment configuration. The values used for the learning variables were: learning rate $\alpha = 0.2$, discount factor $\gamma = 0.9$, exploration/exploitation rate for the ϵ -greedy action selection method: $\epsilon = 0.1$ and the maximum number of steps before an episode is finished was 1000.

In each test, three variables were used to compare Q-Learning and oASP(MDP). First, the root-mean-square deviation (RMSD), that provides information related to the convergence of the methods by comparing values of the $Q(s, a)$ function in the current episode with respect to that obtained in the previous episode. Second, we considered the return (sum of the rewards) received in an episode. Third, the number of steps needed to go from the initial state to the goal state was evaluated. The results obtained were also compared with that of an agent using the optimal policy in a deterministic grid world (the best performance possible, shown as a red-dashed line in the results below).

For oASP(MDP), the number of state-action pairs known by the agent was also measured and compared with the size of Q-Learning’s fixed $Q(s, a)$ tabular implementation. This variable provides information of how far an oASP(MDP) agent is from knowing the complete environment along with how much the $Q(s, a)$ function could be reduced.

The test domains and related results are described in details in the next sections.

4.1 First test: changes in the wall-free-space ratio

In the first test, the size of the grid was fixed to 10×10 and the transition probabilities were assigned at 90% for moving on the desired direction and 5% for moving in each of the two directions that are orthogonal to the desired. In this test, changes in the environment occurred in the number and location of walls in the grid. Initially the domain starts with no walls (0%), then it changes to a world where 10% of the grid is occupied by walls placed at random locations and, finally,

the grid world changes to a situation where 25% of the grid is occupied by walls. Each change occurs after 1000 episodes.

The results obtained in the first test are represented in Figure 2. Figure 2a shows that the RMSD values of oASP(MDP) decrease faster than those of Q-Learning, thus converging to the optimal policy ahead of Q-Learning. It is worth observing that when a change occurs in the environment (at episodes 1000 and at 2000) there is no increase in oASP(MDP) RMSD values, contrasting with the significant increase in Q-Learning’s values. A similar behaviour is shown in Figure 2c, where there is no change in the number of steps of oASP(MDP) after a change occurs, at the same time that Q-learning number of steps increase considerably at that point.

The return values obtained in this test are shown in Figure 2b, where it can be observed that both oASP(MDP) and Q-learning reach the maximum value together during the initial episodes, but there is no reduction in the return values of oASP(MDP) when the environment changes, whereas Q-learning returns drop to the initial figures.

Figure 2d shows the number of state-action pairs that oASP(MDP) has found for the grid world. Values obtained after the 15th episode were omitted since they presented no variation. This figure shows that oASP(MDP) has explored every state of the grid world and performed every action allowed in each state, resulting in a complete description of the environment. Since oASP(MDP) has provided the complete description of the environment, the agent that uses oASP(MDP) optimizes the same action-value function as the agent that uses Q-Learning, thus the optimal policy found by both agents is the same. Due to the exploration of the environment performed in the beginning of the interaction, before the 10th episode the agent has executed every action in every possible state at least once and, as can be seen in line 7 of Algorithm 1, the agent then uses the underlying RL procedure to find the action-value function.

4.2 Second test: changes in the transition probabilities

In this test, the grid was fixed at a 10×10 size, with wall-free-space ratio fixed at 25%. Changes in the environment occurred with respect to the transition probabilities. Initially, the agent’s actions had 50% of probability for moving the agent in the desired direction and 25% for moving it in each of the two orthogonal directions. The first change set the probabilities at 75% (assigned to the desired action effect) and 12.5% (for the directions orthogonal to the desired). The final change assigned 90% for moving in the desired direction and 5% for moving in each of the orthogonal directions.

The RMSD values for oASP(MDP), in this case, decreased faster than those of Q-Learning, reaching zero before the first change occurred, while Q-Learning at that point had not yet converged, as shown in Figure 3a. Analogously to the first test, there is no change in RMSD values of oASP(MDP) when the environment changes, whereas Q-learning presents re-initializations. In the results on return and the number of steps, shown in Figures 3b and 3c respectively, the performance of oASP(MDP) improves faster than the Q-Learning performance when there is a change in the environment. This is explained by the fact that, after oASP(MDP) approxi-

mates the action-value function (in the periods between the changes), when a change occurs, the information about it, acquired by the agent, is used to find solutions in the new world situation. In this case, the current action-value function is simply updated. Q-Learning, on the other hand, is restarted at each time a change occurs, resulting in the application of an inefficient policy in the new environment.

The number of state-action pairs that oASP(MDP) was able to describe is shown in Figure 3d. Once more, values obtained after the 15th episode were omitted, as they present no variation after this point. Analogous to the results obtained in the first experiment, oASP(MDP) was capable of executing at least once every allowed action in every state possible to be visited. As before, by exploring the environment oASP(MDP) could efficiently find the set of allowed states, defining the complete $Q(s, a)$.

In summary, the tests performed in the domains considered show that the information previously obtained is beneficial to an agent that learns by interacting with a changing environment. The action-value function obtained by oASP(MDP) before a change occurs accelerates the approximation of this function in a new version of the environment, avoiding the various re-initializations observed in Q-learning alone (as shown in Figures 2 and 3). However, as the action-value function approximation method used in oASP(MDP) (in this work) was Q-Learning, the policies learnt by oASP(MDP) and Q-Learning alone were analogous. This can be observed when comparing the curves for oASP(MDP) and Q-Learning in Figures 2 and 3 after convergence, noticing also that they keep the same distance with respect to the best performance possible (red-dashed lines in the graphs).

Tests were performed in virtual machines in AWS EC2 with t2.micro configuration, which provides one virtual core of an Intel Xeon at 2.4GHz, 1GB of RAM and 8GB of SSD with standard Debian 8 (Jessie). oASP(MDP) was implemented in Python 3.4 using ZeroMQ for providing messages exchanges between agent and environment and Clingo [Gebser *et al.*, 2013] was used as the ASP Engine. The source code for the tests can be found in the following (anonymous) URL: <http://bit.ly/2k031kl>.

5 Related Work

Previous attempts at combining RL with ASP include [Zhang *et al.*, 2015], which proposes the use of ASP to find a predefined plan for a RL agent. This plan is described as a hierarchical MDP and RL is used to find the optimal policy for this MDP. However, changes in the environment, as used in the present work, were not considered in [Zhang *et al.*, 2015].

Analogous methods were proposed by [Khandelwal *et al.*, 2014; Yang *et al.*, 2014], in which an agent interacts with an environment and updates an action's cost function. While [Khandelwal *et al.*, 2014] uses the action language *BC*, [Yang *et al.*, 2014] uses ASP to find a description of the environment. Although both methods consider action costs, none of them uses Reinforcement Learning and they do not deal with changes in the action-value function description during the agent's interaction with the environment.

An approach to non-deterministic answer set programs

is P-Log [Baral *et al.*, 2009; Gelfond and Rushton, 2010]. While P-Log is capable of calculating transition probabilities from sampling, it is not capable of using this information to generate policies. Also P-Log does not consider action costs. Thus, although P-Log can be used to find the transition function, it cannot find the optimal solution, as proposed here.

Works related to non-stationary MDPs such as [Even-Dar *et al.*, 2009; Yu *et al.*, 2009], which deal only with changes in reward function, are more associated with RL alone than with a hybrid method such as oASP(MDP), since RL methods are already capable of handling changes in the reward and transition functions. The advantage of ASP is to find the set of states so that it is possible to search for an optimal solution regardless of the agent's transition and reward functions.

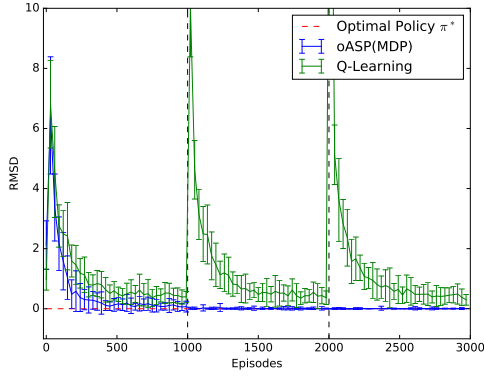
A proposal that closely resembles oASP(MDP) is [Garnelo *et al.*, 2016]. This method proposes the combination of deep learning to find a description to a set of states, which are then described as rules to a probabilistic logic program and, finally, a RL agent interacts with the environment using the results and learns the optimal policy.

6 Conclusion

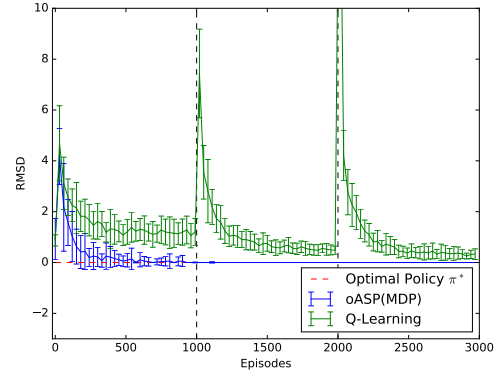
This paper presented the method oASP(MDP) for approximating action-value functions of Markov Decision Processes, in non-stationary domains, with unknown set of states and unknown transition and reward functions. This method is defined on a combination of Reinforcement Learning (RL) and Answer Set Programming (ASP). The main advantage of RL is that it does not need *a priori* knowledge of transition and reward functions, but it relies on having a complete knowledge to the set of domain states. In oASP(MDP), ASP is used to construct the set of states of an MDP to be used by a RL algorithm. ASP programs representing domain states and transitions are obtained as the agent interacts with the environment. This provides an efficient solution to finding optimal policies in changing environments.

Tests were performed in two non-stationary non-deterministic grid-world domains, where each domain had one property of the grid world changed over time. In the first domain, the ratio of obstacles and free space in the grid was changed, whereas in the second domain changes occurred in the transition probabilities. The changes happened in intervals of 1000 episodes in both domains. Results show that, when a change occurs, oASP(MDP) (with Q-learning as the action-value function) is capable of approximating the $Q(s, a)$ function faster than Q-learning alone. Therefore, the combination of ASP with RL was effective in the definition of a method that provides more general (or more elaboration tolerant) solutions to changing domains than RL methods alone.

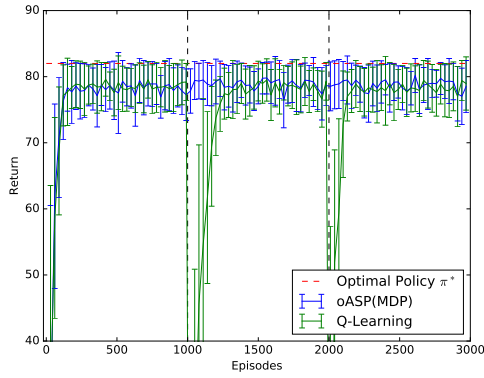
Future work will be directed toward the development of an interface to facilitate the use of oASP(MDP) with distinct domains, such as those provided by the DeepMind Lab [Beattie *et al.*, 2016]. Also, a comparison of oASP(MDP) with the framework proposed in [Garnelo *et al.*, 2016] is an interesting subject for future research.



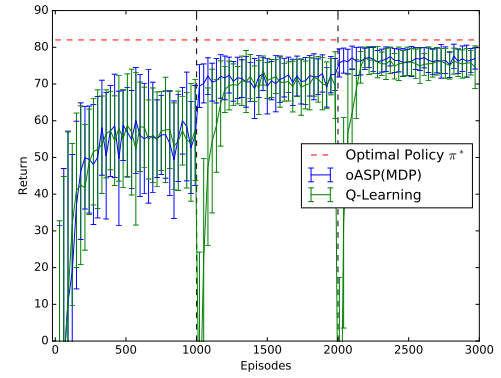
(a) RMSD.



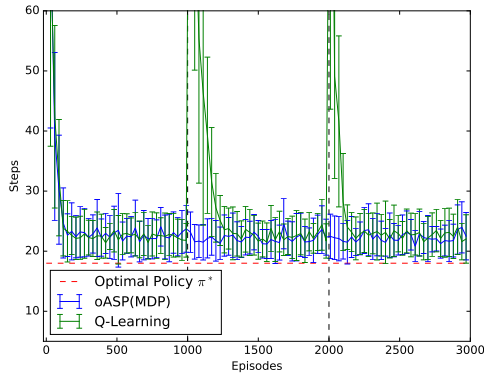
(a) RMSD.



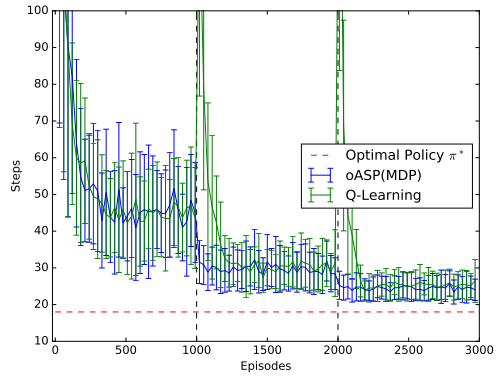
(b) Return.



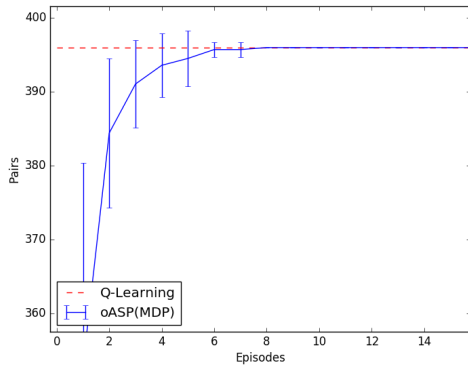
(b) Return.



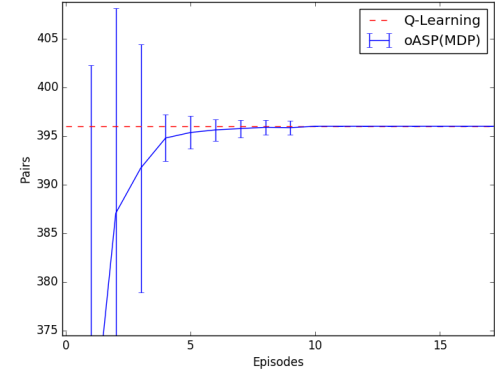
(c) Number of steps.



(c) Number of steps.



(d) Numbers of state-action pairs.



(d) Number of state-action pairs.

Figure 2: Results for the first test.

Figure 3: Results for the second test.

References

- [Baral *et al.*, 2009] Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming*, 9(1):57, 2009.
- [Beattie *et al.*, 2016] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, and Stig Petersen Shane Legg. Deepmind lab. *arXiv preprint arXiv:1612.03801v2 [cs.AI]*, December 2016.
- [Bellman and Dreyfus, 1971] Richard Ernest Bellman and Stuart E. Dreyfus. *Applied dynamic programming*. Princeton Univ. Press, 4 edition, 1971.
- [Bellman, 1957] Richard Bellman. A Markovian decision process. *Indiana University Mathematics Journal*, 6(4):679–684, 1957.
- [Even-Dar *et al.*, 2009] Eyal Even-Dar, Sham. M. Kakade, and Yishay Mansour. Online markov decision processes. *Mathematics of Operations Research*, 34(3):726–736, 2009.
- [Garnelo *et al.*, 2016] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518 [cs]*, September 2016.
- [Gebser *et al.*, 2013] Martin Gebser, Roland Kaminski, and Benjamin Kaufmann. *Answer set solving in practice*. Morgan & Claypool Publishers, 2013.
- [Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski, Bowen, and Kenneth, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988.
- [Gelfond and Rushton, 2010] Michael Gelfond and Nelson Rushton. Causal and probabilistic reasoning in P-log. *Heuristics, Probabilities and Causality. A tribute to Judea Pearl*, pages 337–359, 2010.
- [Gelfond, 2008] Michael Gelfond. *van Harmelen, Frank; Lifschitz, Vladimir; Porter, Bruce. Handbook of Knowledge Representation*, chapter Answer sets, page 285–316. Elsevier, 2008.
- [Khandelwal *et al.*, 2014] Piyush Khandelwal, Fangkai Yang, Matteo Leonetti, Vladimir Lifschitz, and Peter Stone. Planning in action language BC while learning action costs for mobile robots. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*, 2014.
- [Lifschitz, 2002] Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1):39–54, 2002.
- [McCarthy, 1987] John McCarthy. Generality in artificial intelligence. *Communications of the ACM*, 30(12):1030–1035, 1987.
- [McCarthy, 1998] John McCarthy. Elaboration tolerance. In *Proc. of the Fourth Symposium on Logical Formalizations of Commonsense Reasoning (Common Sense 98)*, volume 98, London, UK, 1998.
- [Sutton and Barto, 2015] Richard S Sutton and Andrew G Barto. *Reinforcement learning an introduction – Second edition, in progress (Draft)*. MIT Press, 2015.
- [Watkins, 1989] Christopher J. C. H. Watkins. *Learning from deSuttonlayed rewards*. PhD thesis, University of Cambridge England, 1989.
- [Yang *et al.*, 2014] Fangkai Yang, Piyush Khandelwal, Matteo Leonetti, and Peter Stone. Planning in answer set programming while learning action costs for mobile robots. In *AAAI Spring 2014 Symposium on Knowledge Representation and Reasoning in Robotics (AAAI-SSS)*, 2014.
- [Yu *et al.*, 2009] Jia Yuan Yu, Shie Mannor, and Nahum Shimkin. Markov decision processes with arbitrary reward processes. *Mathematics of Operations Research*, 34(3):737–757, 2009.
- [Zhang *et al.*, 2015] Shiqi Zhang, Mohan Sridharan, and Jeremy L. Wyatt. Mixed logical inference and probabilistic planning for robots in unreliable worlds. *IEEE Transactions on Robotics*, 31(3):699–713, 2015.