A Heuristic Strategy for Learning in Partially Observable and Non-Markovian Domains

Matteo Leonetti [1] and **Subramanian Ramamoorthy** [2]

Abstract. Robotic applications are characterized by highly dynamic domains, where the agent has neither full control of the environment nor full observability. In those cases a Markovian model of the domain, able to capture all the aspects that the agent might need to predict, is generally not available or excessively complex. Moreover, robots pose relevant constraints on the amount of experience they can afford, moving the focus of learning their behavior from reaching optimality in the limit, to making the best use of the little information available. We consider the problem of finding the best deterministic policy in a Non-Markovian Decision Process, with a special attention to the sample complexity and the transitional behavior before such a policy is reached. We would like robotic agents to learn in real time while being deployed in the environment, and their behavior to be *acceptable* even while learning.

1 Introduction

Robotic applications are characterized by highly dynamic domains, where the agent has neither full control of the environment nor full observability. In those cases a Markovian model of the domain, able to capture all the aspects that the agent might need to predict, is generally not available or excessively complex. A very general framework to face such problems is the one of Partially Observable MDP (POMDP) [2].

While most of the methods to solve POMDPs attempt some state estimation, we follow the previous work in the literature about learning with hidden states [6, 5, 4] (which make the system Non-Markovian in general), and focus on a different aspect of the learning process. In Reinforcement Learning (RL) optimality is usually the main target, but robotic applications pose relevant constraints on the number of experiments that the agent can afford (in terms of time, or other resources). We, therefore, believe that the focus should be moved from proving optimality in the limit, to obtaining the best possible behavior with the little information available, and gathering this information carefully.

We consider the problem of finding the best deterministic policy in a Non-Markovian Decision Process, with a special attention to the sample complexity and the transitional behavior before such a policy is reached. We would like robotic agents to learn in real time while being deployed in the environment, and their behavior to be *acceptable* even while learning. To this aim, we propose an algorithm structured in two phases: the first one, as short as possible unless simulated, provides an exploratory behavior that gathers information on the effect of actions.The second phase starts exploiting the data collected during the first phase making small exploratory steps and traversing the policies that look more promising on the basis of the collected data.

As an example, consider the domain of robotic soccer, in which multiple agents interact in both a cooperative and a competitive way, making the environment extremely dynamic and unpredictable from a single-agent perspective. Moreover, in applications such as the one just mentioned, the number of actions available at any time is considerable, making the branching factor of the policies an issue. Nonetheless, the actions actually meaningful in most of the situations are few. A robot should ideally be able to realize quickly that, for instance, just staring at the ball is not going to take it in any farther, no matter what other actions he could do later, and independently from all the other many aspects of the world. The method we propose aims at identifying those "wrong" actions and avoiding them unless proved necessary.

We provide a preliminary evaluation on a common test-bed in the literature of NMDP that allows us to easily compare our algorithm with the best results obtained so far.

2 Problem formulation

We consider NMDPs with a finite set of states S and a finite set of actions A, with similar assumptions regarding observations as in POMDPs. A deterministic policy π on the NMDP maps each state $s \in S$ to an action $a \in A(s)$ among those available in s. In the following, we borrow the notation from Perkins [5] indicating with $\tau = \{s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T\}$ a trajectory in the NMDP. A policy determines a well defined probability measure, $\mu(\pi)$, over the set of all possible trajectories. The reward corresponding to each trajectory is a random variable defined as $R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$ where $\gamma \in [0, 1]$ is the discount factor. We define the value of a policy as the expected discounted reward:

$$V^\pi = E_{\tau \sim \mu(\pi)}\{R(\tau)\} = E^\pi\{R(\tau)\}$$

The tasks are episodic, which means that they terminate under any policy with probability one. We limit ourselves to the search for the best deterministic policy, although in NMDPs the optimal policy might be stochastic [6]. We also adopt Perkins's [5] definition of the action-value function that we report in the following. Given a trajectory τ, the portion of $R(\tau)$ preceding a state s is denoted as $R_{pre-s}(\tau)$. Similarly, the portion of $R(\tau)$ following a state s is denoted as $R_{post-s}(\tau)$. For any state s the value of a policy can be rewritten as

$$\begin{aligned} V^\pi &= E^\pi[R(\tau)] \\ &= E^\pi[R_{pre-s}(\tau)] + E^\pi[R_{post-s}(\tau)] \end{aligned}$$

[1] Department of Computer and System Sciences, Sapienza University of Rome
[2] School of Informatics, The University of Edinburgh

Let $\pi \leftarrow (s,a)$ represent the policy that is identical to π except for the state s that is mapped to the action a. We define the action-value function for a pair $\langle s,a\rangle$ as:

$$Q^{\pi}(s,a) = E^{\pi \leftarrow (s,a)}[R_{post-s}(\tau)] \tag{1}$$

We refer to the original paper for an explanation of the differences with the traditional definition. We only point out that if an action a is chosen for a state s, then every time s is encountered the agent will execute a, according to $\pi \leftarrow (s,a)$.

In the following, we present an algorithm for maximizing $Q^{\pi}_{s,a}$ making use of a particular initial exploration phase to collect heuristic information on the most promising policies to be subsequently exploited.

3 Parr and Russell's Grid World

Before proceeding with the description of the algorithm, we introduce the test domain: Parr and Russell's Grid World [3]. Grid World has been used as a test domain in several papers [3, 1, 5] and provides a simple and structured environment with a reasonable branching factor. It has 11 states (figure 1) in a 4 by 3 grid with one obstacle. The agent starts at the bottom left corner. There is a target state and

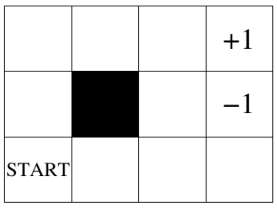

Figure 1. Grid World

a penalty state whose rewards are +1 and -1 respectively. Both are absorbing states, that is when the agent enters them the episode terminates. The actions available in every state are *move north*, *move south*, *move east*, and *move west* which succeed with probability 0.8. With probability 0.1 the agent moves in one of the directions orthogonal to the desired one. In all of the previous cases if the movement is prevented by an obstacle the agent stays put. In any state the agent can only observe the squares east and west of it, having a total of four possible observations. Those observations are going to form the state space of an NMDP whose controller we are going to learn.

4 The algorithm: ϵMaCs

The main idea of the algorithm lies on the intuition that often a few bad choices disrupt the value of all the policies that include them. For instance, consider the initial state in Grid World. Any of the 128 policies out of the 256 total ones that map the initial observation to either *move west* or *move south* have no chance to be optimal. Taking those policies as if they were as valuable as any other, in the search for the optimal policy, just wastes samples. We would rather like to realize that those actions are not promising and not consider them unless we have tried all the other possibilities.

The strategy would ideally consider all the policies from the most *promising* to the least ones, which we believe is beneficial in at least two ways: (1) the algorithm reaches the optimal policy earlier; (2) during the phase of evaluation of those *promising* but suboptimal policies, the behavior is as good as the current information allows.

The algorithm is constituted by two parts: the *exploratory* phase and the *assessing* phase.

Algorithm 1 ϵMaCs

```
$exp\_length \leftarrow$ number of episodes in the exploratory phase
$\epsilon \leftarrow$ probability of exploration in the assessing phase
$\alpha \leftarrow$ learning step parameter
initialize $Q(s,a)$ pessimistically
{Exploratory phase}
for $i = 1$ to $exp\_length$ do
  generate a trajectory $\tau$ according to a policy $\pi$ extracted uniformly at random
  for all $s \in S$, $a \in A$ $s.t. \langle s,a\rangle$ is in $\tau$ do
    $Q(s,a) = max(Q(s,a), R_{post-s}(\tau))$
  end for
end for
{Assessing phase}
for all other episodes do
  $v \leftarrow$ a value in $[0,1]$ uniformly at random
  if $v \geq \epsilon$ then
    $\pi' \leftarrow$ the policy that greedily maximizes Q
  else
    $\pi' \leftarrow$ a policy chosen uniformly at random
  end if
  generate a trajectory $\tau$ from $\pi'$
  if $v \geq \epsilon$ then
    for all $s \in S$, $a \in A$ $s.t. \langle s,a\rangle$ is in $\tau$ do
      $Q(s,a) = (1-\alpha)Q(s,a) + \alpha R_{post-s}(\tau)$
    end for
  else
    for all $s \in S$, $a \in A$ $s.t. \langle s,a\rangle$ is in $\tau$ do
      $q = (1-\alpha)Q(s,a) + \alpha R_{post-s}(\tau)$
      $Q(s,a) = max(Q(s,a), q)$
    end for
  end if
end for
```

4.1 Exploration: gathering information

The exploration initializes the Q-function to drive the execution in the subsequent phase. For a number of episodes exp_length the agent chooses a policy at random, and in each pair $\langle s,a\rangle$ stores the highest value that any policy, going through $\langle s,a\rangle$, has obtained until then. Consider the simple example of the NMDP in figure 2(a). This NMDP has three states and four actions with a total of four policies. Let the reward returned by each of those policies be normally distributed, with means and standard deviations represented in figure 2(b). Figure 2(c) and 2(d) show the value of the Q-function for each action during a particular run. The first 100 episodes belong to the exploratory phase, in which A1 and A2 obtain the highest reward, making the policy A1-A2 look particularly promising. An action is considered as *promising* as the highest value of the reward that choosing that action has ever given. In the case of A1-A2, its good result is due to the high variance, rather than the highest mean. This aspect is going to be addressed by the second phase of the algorithm.

Several other choices are possible both for the exploration (uniformly at random) and for the value stored (the maximum); for instance, making use of SoftMax we might give a higher priority to the

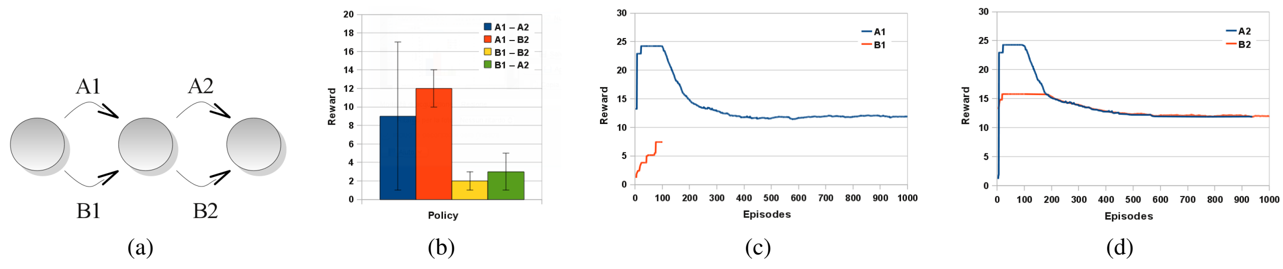

Figure 2. A simple example of an NMDP (a). The four policies return a reward normally distributed whose means and standard deviations are shown in (b). The evolution of the Q-function for the first state (actions A1 and B1) is represented in figure (c), while for the second state (actions A2 and B2) is represented in figure (d).

policies that returned a higher reward. In future work, we will consider relying on further statistics about the policies traversing a given choice point.

4.2 Assessment

We want to maximize the *expected* cumulative discounted reward, rather than the maximum obtainable one, therefore an evaluation of the *promising* policies is needed.

In the second phase the agent acts greedily according to the Q-function previously constructed. It starts from the policy that has given the highest reward and, taking subsequent samples from it, lowers its value until it reaches the expected value. The value of the current policy, in this process, could get lower than the maximum value obtained by some other policy, and stored in one of its actions. In this case, the two policies would be alternately chosen and would race each other down toward their respective expected values, stopping at the highest one. An example of this behavior is shown in figure 2(d). Starting from the episode 101, the policy A1-A2 is greedily executed. Since its mean is considerably lower than the value stored, the estimate keeps dropping until it reaches the value previously stored for B2. At this point, A1-A2 and A1-B2 are executed almost alternately, each one pushing down its estimate toward their respective mean. The learning rate parameter determines the speed with witch the estimates tend to the means. The higher the learning rate, the faster the estimate will reach the mean, but also, the more it oscillates confusing policies close to each other. At some point, A1-B2 reaches its mean and stabilizes, so that the value of A1 is the same for both the policies, but B2 is definitely higher than A2, making A1-B2 the learned policy which is also the optimal one. Notice how in this process B1 has never been executed. This is because the highest value it had given in the first phase has always been lower than the current estimates of the policies taken into account during the assessing phase.

If the maximum value obtained by the optimal policy has been stored in one of the state-action pairs, this algorithm quickly converges to the optimal policy. Unfortunately, this might not happen for two reasons: (1) the optimal policy might have, in all of its states, another policy (not the same for every state) that shares the same action in that state and differs elsewhere, but gives a higher maximum reward. In this case the maximum reward obtained by the optimal policy in the first phase would be hidden by those policies. (2) The optimal policy has never been sampled above the expected reward, and no optimistic estimate of it has had a chance to be stored. Ideally, in the first phase, every policy should be sampled above its expected value at least once. The number of episodes necessary to meet this condition would probably be impractically high for most domains. For this reason, and since the first point wouldn't be avoidable anyway, we add a step of exploration in the second phase too, that guarantees that each policy is continually sampled on the long term. When the agent takes an exploratory step (with probability ϵ) the Q-function is updated only in those state-action pairs that gave a value higher than the current one.

Clearly taking an exploratory step could disrupt the optimal policy if this had already been found. The racing among the two policies would have to happen again until the optimal policy is established once more. This cannot be prevented if we want to make sure that the optimal policy has always a chance to be sampled.

5 Experimental evaluation

We conducted a preliminary evaluation of our algorithm on Grid World, in order to show how the different parameters impact the behavior of the agent. Every 20 episodes for the short term experiments, and every 100 episodes for the long term ones, we pause the learning and evaluate the current controller for 20 episodes. The results are averaged over 200 runs. By "evaluating the controller" we mean that, during the evaluation, the behavior of the agent is the same as if it were learning, but the Q-function is left unchanged. Thus, if at a specific point the agent would choose a policy at random, the reward obtained will be the average of the reward returned following 20 policies picked uniformly at random. Notice that choosing a policy at random, in this context, is different from following the *random policy*. In the former case the same decision is always made in the same state, while in the latter case each time a state is hit a random choice is made.

We compare our results with two control strategies: Sarsa(λ) with ϵ-greedy exploration, and Sarsa(λ) with optimistic initialization. The latter strategy consists in initializing the Q-function at an optimistic value for each state-action pair, and exploiting the current estimate at any time. We borrowed a few parameters from the literature [1, 5] and spent some time optimizing others. When not differently specified the Q-function has been initialized at -4. The best behavior we could achieve for ϵ-greedy was with ϵ starting at 0.2 and linearly decaying to 0 in 80000 actions. For the optimistic initialization, the Q-function has been initialized at 1. In both cases $\alpha = 0.01$ and $\lambda = 0.9$

Figure 3 shows the cumulative rewards obtained by different controllers. ϵMaCs here has been evaluated without any exploration in its second phase. Sarsa(0.9) with optimistic initialization reliably converges to the optimal policy a lot faster than ϵ-greedy. It is this behavior that we want to improve, pruning some of the exploration by getting a more realistic initialization. With an initial phase of 100

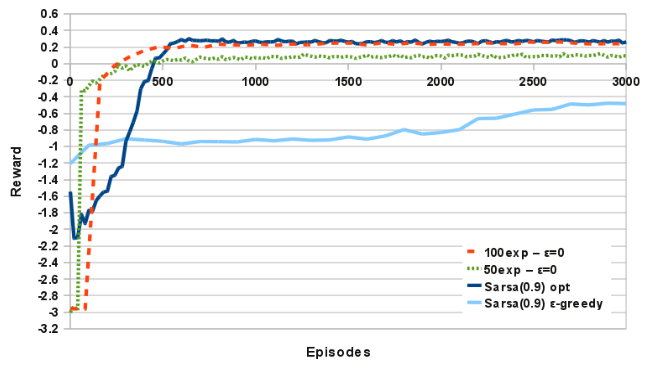

Figure 3. Cumulative reward on the short term for different controllers. ϵMaCs is evaluated without any exploration in the second phase.

episodes and $\alpha = 0.01$, ϵMaCs always converges to the optimal policy shortly after the initial exploration. We also evaluated the behavior of the agent with 50 episodes, in order to understand the consequences of little initial sampling. In case the agent could not afford a longer initial phase, we would like that it still quickly converged to a "good" policy, if not the optimal one. Indeed, after 50 episodes the average reward stabilizes at around 0.1, while the optimum is around 0.25. Considering that most of the policies give a reward of -4 and that the average reward obtained is non-decreasing, this can be probably considered a good result.

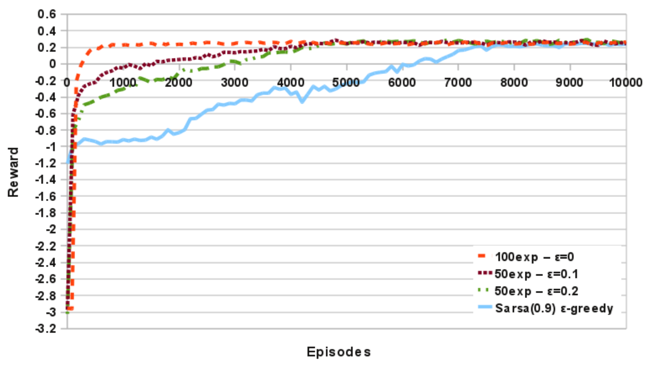

Figure 4. Cumulative reward on the long term allowing exploration in the second phase of ϵMaCs

In the second set of experiments we tried to establish whether, by allowing some exploration also on the second phase, it is eventually possible to reach the optimal policy even from a short initial phase. Clearly exploration is a double-edged sword: on the one hand it allows to discover the optimal policy, on the other hand it worsen the average behavior of the agent that leaves its current "good" policy. Figure 4 shows the results for two different settings, compared with Sarsa(λ) and ϵMaCs after 100 initial episodes as already described. In one setting we let ϵ start at 0.2 and reach 0.01 in 5000 episodes, remaining constant afterwords. In the other setting ϵ started at 0.1. The two results fall in between Sarsa and the optimum obtained with 100 initial episodes. Moreover, the increase in the performance is linear and follows perfectly the decay of ϵ. This probably means that the optimal policy is identified early and, from that point on, the exploration is the only responsible for the sub-optimal behavior. We have not performed an extensive evaluation over the possible values for the initial ϵ and its decaying rate, therefore we cannot state exactly how close the behavior can be pushed towards the optimal line above by varying these two values. It seems reasonable though, that the linear dependence allows for a faster convergence up to a point when the exploration becomes too short, and we fall into the initial case of figure 3 with no exploration at all.

6 Conclusion

We devised an algorithm to learn the best deterministic policy in an NMDP searching the policy space in a favorable order. The algorithm first attempts to collect information about the actions' values and then exploits it preventing the agent from behaving arbitrarily bad, possibly allowing its early deployment in the environment. Several future directions can be followed in making this simple algorithm more efficient in the exploration, directing the search even in the first phase rather than acting randomly. We believe that novel and more efficient techniques on NMDPs can help to the simplifications of robots' controllers and the scalability of RL in practical applications.

REFERENCES

[1] J. Loch and S. Singh, 'Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes', in *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 323–331. Citeseer, (1998).

[2] G.E. Monahan, 'A survey of partially observable Markov decision processes: Theory, models, and algorithms', *Management Science*, **28**(1), 1–16, (1982).

[3] R. Parr and S. Russell, 'Approximating optimal policies for partially observable stochastic domains', in *Ineternational Joint Conference on Artificial Intelligence*, volume 14, pp. 1088–1095. Citeseer, (1995).

[4] Mark D. Pendrith and Michael McGarity, 'An analysis of direct reinforcement learning in non-markovian domains.', in *ICML*, ed., Jude W. Shavlik, pp. 421–429. Morgan Kaufmann, (1998).

[5] T.J. Perkins, 'Reinforcement learning for POMDPs based on action values and stochastic optimization', in *Proceedings of the national conference on artificial intelligence*, pp. 199–204. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, (2002).

[6] S.P. Singh, T. Jaakkola, and M.I. Jordan, 'Learning without state-estimation in partially observable Markovian decision processes', in *Proceedings of the eleventh international conference on machine learning*, pp. 284–292. Citeseer, (1994).