Action Value Based Reinforcement Learning for POMDPs

Theodore J. Perkins

University of Massachusetts Amherst
Department of Computer Science
Technical Report UM-CS-2001-20

Action Value Based Reinforcement Learning for POMDPs

Theodore J. Perkins
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
perkins@cs.umass.edu

Abstract

We present a new, model-free reinforcement learning algorithm for learning deterministic tabular policies for controlling partially observable Markov decision processes. The class of policies we consider includes a number of special cases which have been studied by reinforcement learning researchers before, including: reactive (i.e., memoryless) policies, policies based on a finite history window or selected parts of the history, and policies incorporating finite-state memories. We propose a new definition of action-value for this context, and derive a Monte Carlo learning algorithm based on that definition. We show that locally optimal policies, and only locally optimal policies, are stable under this learning algorithm. We conjecture that the algorithm always converges to a locally optimal policy. This contrasts with many existing action-value based learning algorithms, such as Q-learning and Sarsa, which may converge to policies that are not locally optimal or may fail to converge at all when applied to partially observable problems. Experiments examine the convergence properties of the new algorithm.

1 Introduction

Partially observable Markov decision processes (POMDPs) are receiving increased attention from the reinforcement learning (RL) community, in part because POMDPs can model many RL problems more realistically than can Markov decision processes, and in part because the complexity of solving POMDPs exactly makes them an attractive target for approximate methods. However, POMDPs violate one of the fundamental assumptions of many RL methods—the Markov assumption. In general, the future behavior of a POMDP depends not only on the agent's current observation and action, but potentially on all of the previous observations, actions, and rewards as well. Value function-based RL methods in particular, e.g., Q-Learning, Sarsa(λ), TD(λ)-control, are known to perform unreliably in POMDP settings [3, 6, 11]. Pendrith and McGarity [11] analyzed the stability of (locally) optimal reactive policies under various of these algorithms. They showed, by example, that for some problems these algorithms can converge to policies that are not locally optimal. However, they also proved an important positive result: when rewards are undiscounted, (locally) optimal policies are stable under first-visit Monte Carlo action-value learning.

We extend their results to a more general class of tabular policies, which includes history-dependent policies and policies with finite-state internal memories. Further, we show that with an appropriate definition of action value, a new Monte Carlo learning algorithm can be devised which extends their positive stability result to the case of arbitrarily discounted returns. The key observation is that for a policy to be stable, it must be greedy with respect to its own action values. Under our new action

value definition, only locally optimal policies have this property, and hence are stable under our learning algorithm. In contrast, in the Appendices we construct POMDPs on which Q-learning and Sarsa(λ) cannot converge to any policy, because no policy is greedy with respect to its own action values, as learned by these algorithms.

2 Partially Observable Markov Decision Processes

A POMDP models the interaction of an agent with its environment. We study POMDPs in which the agent-environment interaction is divided into *episodes*. At the start of each episode, the environment's state is selected randomly according to a fixed distribution S_0 over the environment's state set S. An episode ends when the environment enters a terminal state. At discrete time steps, $t = 0, 1, 2, \ldots$ the agent receives an observation $o_t \in O$ distributed according to $P_O(s_t)$. The agent then chooses an action $a_t \in A(o_t)$. The agent's immediate reward, r_t, and the next state of the environment, s_{t+1}, are then determined according to a joint distribution $P_{r,s}(s_t, a_t)$. We will assume that the probabilities and expectations we use below are well-defined and finite. Subject to that constraint, we allow S and O to be arbitrary sets and the distributions S_0, $P_O(\cdot)$, and $P_{r,s}(\cdot,\cdot)$ to be appropriately defined probability measures. We also assume a global upper bound A_{max} on the number of actions available to the agent in any state.

Each episode generates a trajectory: a finite sequence of observations, actions, and rewards, $\{o_0, a_0, r_0, o_1, a_1, r_1, \ldots, o_m\}$, where the environment enters a terminal state at step m. The discounted return received during the episode is $R = \sum_{t=0}^{m-1} \gamma^t r_t$, where $\gamma \in [0,1]$ is a discount rate. A priori, the agent knows only O, A, and γ, and must learn to control the environment by interacting with it. The goal of the agent is to learn how to choose actions such that the expected discounted return over all trajectories is maximized.

3 Deterministic Tabular Policies

In this section we introduce the class of policies we study and describe how it relates to existing classes studied in the RL literature. A *history* is a beginning subsequence of a trajectory of the form: $\{o_0, a_0, r_0, o_1, a_1, r_1, \ldots, o_n\}$, where o_n may or may not correspond to the end of the episode. In general, acting optimally in a POMDP requires conditioning the choice of action on the entire history since the beginning of the episode [1], but this is often infeasible. In this paper we are concerned with learning deterministic tabular policies. We define such a policy to be a pair (f, g), where $f : H \to \{1, \ldots, k_f\}$ partitions the set of all histories, H, and $g : \{1, \ldots, k_f\} \to \{1, \ldots, A_{max}\}$ assigns an action to each partition. Let $A(h)$ denote the set of actions available to the agent after it has experienced history h. For a policy to be valid, we require that $g(f(h)) \in A(h)$ for all histories, and that $A(h_1) = A(h_2)$ for any h_1 and h_2 mapping to the same partition. We assume that the partitioning function, f, is fixed and that the role of learning is to optimize the partition-to-action mapping, g. Before defining the value of a policy and notions of optimality, we describe some classes of policies that fit this framework.

A common simplification made when searching for a good policy for a POMDP is to allow action choice to depend only on the k most recent observations, for some fixed k [3, 4, 6]. If O is finite, such policies are tabular. More generally, McCallum [7] introduced various techniques for determining which parts of the history are most relevant for decision making. These algorithms result in trees that condition action choice only on selected portions of history. In our framework, the tree structure corresponds to f and the assignment of actions to leaves corresponds to g. [1] For some problems, the agent must "remember" something that happened a long time in the past in order to perform well in the present. One way to achieve this is to augment an agent with an internal finite state memory, and let action choice be conditioned on the state of the memory as well as the current observation [4, 8]. If, for example, O is finite, then an N-state internal memory can be implemented as a deterministic

[1] We note that our work and that of McCallum are quite complementary, as his focus was on learning f and ours is on learning g.

tabular policy by augmenting the agent's observation space to be $O \times \{1,\ldots,N\}$ and asking the agent to learn a "reactive" policy: $g : O \times \{1,\ldots,N\} \to \{1,\ldots,A_{max}\} \times \{1,\ldots,N\}$. The first component of the action is applied to the POMDP and the second component determines the state of the memory for the next time step. Deterministic tabular policies allow for many other possibilities as well, such as: combinations of history dependence and internal memory; aggregation, which is often necessary when O is large or infinite; conditioning on rewards received; and even conditioning on factors that are not computable by finite machines, such as (to take an unlikely example) whether on not the observation sequence in the history is a palindrome.

4 Policy Value, Action Value, and Optimality

When searching for a good policy from some restricted class, it is not always possible to simultaneously maximize the expected discounted return from every history [1, 3]. Thus there may be no single policy that strictly dominates all others in that sense. A common recourse is to define a single numerical criterion to optimize. Following Bertsekas [1], we define the value of a deterministic tabular policy to be the discounted return that the agent can expect during one episode, when following that policy:

$$V^{\pi} = E\{\sum_{t=0}^{m-1} \gamma^t r_t \mid \pi\},$$

where m is a random variable denoting the time at which an episode ends and the r_t are random variables denoting the reward on the t^{th} time step. The conditional expectation notation has been used to denote the dependence on the policy π. The expectation also depends on S_0, P_O, and $P_{r,s}$, but since these are fixed we suppress them in the notation.

Littman [5] showed that finding a globally optimal reactive policy for a POMDP, even given complete knowledge of the POMDP, is NP-hard. Since reactive policies are a special case of tabular policies, finding globally optimal tabular policies is intractable. A lesser goal for an RL agent is to find a locally optimal policy, which we define to be a policy $\pi = (f,g)$ for which $V^{\pi} \geq V^{\pi'}$ for all $\pi' = (f,g')$ where g' differs from g on at most one partition.

To define action values, we begin by defining $V^{\pi}(\rho)$—the value of a partition ρ with respect to a given policy π. We say a partition ρ *occurs* on the n^{th} step of a trajectory if the history up to the n^{th} step is mapped to ρ by f. Let P^{π}_{ρ} be the probability that ρ occurs on a trajectory generated by following π. If $P^{\pi}_{\rho} = 0$, we will say $V^{\pi}(\rho)$ is undefined. If $P^{\pi}_{\rho} > 0$, then let occ-ρ denote the event that ρ occurs on a trajectory and let t_{ρ} be a random variable denoting the first time at which ρ occurs. We define partition value as:

$$V^{\pi}(\rho) = E\{\sum_{t=t_{\rho}}^{m-1} \gamma^t r_t \mid \pi, \text{occ-}\rho\}.$$

Note that although the summation only begins at t_{ρ}, this expectation is over the set of complete trajectories (from initial observation to termination) that contain ρ. If multiplied by P^{π}_{ρ}, $V^{\pi}(\rho)$ can be viewed as the portion of V^{π} following the first occurrence of ρ in a trajectory. If we define $R = \sum_{t=0}^{m-1} \gamma^t r_t$, $R_{pre-\rho} = \sum_{t=0}^{t_{\rho}-1} \gamma^t r_t$, and $R_{post-\rho} = \sum_{t=t_{\rho}}^{m-1} \gamma^t r_t$, then this follows by rewriting the policy value as: $V^{\pi} = E\{R \mid \pi\} = (1 - P^{\pi}_{\rho})E\{R \mid \pi, \neg\text{occ-}\rho\} + P^{\pi}_{\rho}E\{R \mid \pi, \text{occ-}\rho\} = (1 - P^{\pi}_{\rho})E\{R \mid \pi, \neg\text{occ-}\rho\} + P^{\pi}_{\rho}E\{R_{pre-\rho} \mid \pi, \text{occ-}\rho\} + P^{\pi}_{\rho}E\{R_{post-\rho} \mid \pi, \text{occ-}\rho\}$. The last term, the portion of V^{π} following the first occurrence of ρ, is just $P^{\pi}_{\rho}V^{\pi}(\rho)$.

We now define the value of a partition-action pair (ρ, a) as what the value of partition ρ would be if $g(\rho)$ were a. That is, letting $\pi \leftarrow (\rho,a) = (f,g')$ denote the policy that matches π except (possibly) that $g'(\rho) = a$, then partition action values for π are defined as:

$$Q^{\pi}(\rho,a) = V^{\pi \leftarrow (\rho,a)}(\rho) = E\{\sum_{t=t_{\rho}}^{m-1} \gamma^t r_t \mid \pi \leftarrow (\rho,a), \text{occ-}\rho\}.$$

Algorithm 1 Action Value Based Monte-Carlo Learning

Given initial policy π and partition-action values Q, initialize $\Gamma = \emptyset$ and repeat forever:

1. For all ρ, if $g(\rho) \notin \arg\max_a Q(\rho,a)$ then: *// Greedifying policy*
   - Set $g(\rho)$ to some element of $\arg\max_a Q(\rho,a)$.
   - Set $\Gamma = \emptyset$. *// Resetting Γ if policy changes*
2. Choose whether to generate an on-policy or off-policy trajectory. If on-policy:
   - Generate a trajectory τ by following π for one episode.
   - For each ρ occurring in τ, add ρ to Γ and Update$(Q,\tau,\rho,g(\rho))$.

   Else, if off-policy:
   - Choose $\rho \in \Gamma$ and an action $a \neq g(\rho)$ valid for ρ.
   - Generate a trajectory τ by following $\pi \leftarrow (\rho,a)$ for one episode.
   - If ρ occurs in τ, Update(Q,τ,ρ,a).
   - Add any ρ' occurring before ρ to Γ.

Update(Q,τ,ρ,a)

- Let t_ρ be the time ρ first occurs in τ. Update Q as $Q(\rho,a) \leftarrow (1-\alpha)Q(\rho,a) + \alpha \sum_{t=t_\rho}^{m-1} \gamma^t r_t$, where α is a learning rate parameter, which may vary.

For Markov decision processes, state-action values $Q^\pi(s,a)$ are defined as the expected discounted return if the agent starts in state s, takes action a, and follows policy π afterwards. Our definition differs in three ways: 1) the notion of starting in state s is replaced by the first occurrence of partition ρ; 2) the agent takes the action a not just on the first occurrence of ρ in an episode, but on all occurrences; 3) rewards are discounted depending on the time from the start of the episode, not from the time of first occurrence of ρ. E.g., if a partition always occurs on time step 2 and is followed by rewards r_2, r_3, and r_4, then the value of ρ is $\gamma^2 r_2 + \gamma^3 r_3 + \gamma^4 r_4$ and not $r_2 + \gamma r_3 + \gamma^2 r_4$. This definition of action value relates the values of individual actions to the global measure of value, V^π, and is crucial for the theoretical results derived below.

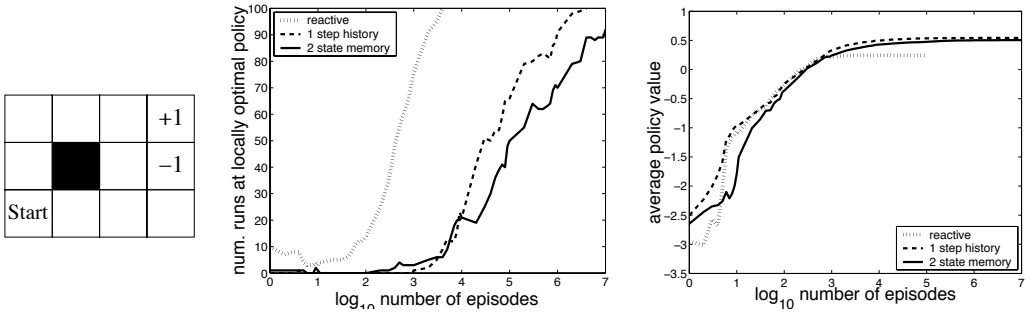
5 Learning Algorithm and Analysis

Algorithm 1 (see above) is a Monte-Carlo algorithm for learning partition-to-action mappings. It maintains a table of partition-action values, Q, which is updated based on trajectories from the POMDP, and a current policy, π, which is updated to be greedy with respect to Q. A naive approach to updating Q would be to repeatedly choose a partition-action pair (ρ,a) at random and follow $\pi \leftarrow (\rho,a)$ for one episode. This which generates one sample of $Q^\pi(\rho,a)$ *if* ρ occurs in the trajectory. Algorithm 1 uses two methods to make this process more efficient. First, it distinguishes between "on-policy" trajectories, generated by following π, and "off-policy" trajectories, generated by following $\pi \leftarrow (\rho,a)$ for some $a \neq g(\rho)$. On-policy trajectories are used to update $Q^\pi(\rho,g(\rho))$ for any ρ occurring in the trajectory. Off-policy trajectories are only used to update $Q(\rho,a)$. Second, the algorithm maintains a list, Γ, of partitions that have been observed to occur under the current policy. The algorithm only attempts to learn $Q(\rho,a)$ for partitions that have occurred before.

The primary theoretical motivation for introducing Algorithm 1 is summarized by the following two theorems. We use the convention that if action values are undefined for partition ρ under policy π, because ρ occurs with probability 0, then $\arg\max_a Q^\pi(\rho,a)$ is just the set of all actions available from ρ.

Theorem 1 *A deterministic tabular policy $\pi = (f,g)$ is locally optimal iff it is greedy with respect to its action values, i.e., iff $g(\rho) \in \arg\max_a Q^\pi(\rho,a)$ for all ρ.*

Proof: Let ρ and a be arbitrary, and let $\pi' = \pi \leftarrow (\rho,a)$. If $P_\rho^\pi = 0$, then clearly $V^\pi = V^{\pi'}$ and



(a) (b) (c)

Figure 1: (a) The Parr and Russell grid world POMDP. (b,c) Learning results.

$g(\rho) \in \arg\max_a Q^\pi(\rho, a)$. If $P_\rho^\pi > 0$ then:

$$\begin{aligned} & V^\pi \geq V^{\pi'} \\ \iff \quad & (1-P_\rho^\pi)E\{R \mid \pi, \neg\text{occ-}\rho\} + P_\rho^\pi E\{R_{pre-\rho} \mid \pi, \text{occ-}\rho\} + P_\rho^\pi E\{R_{post-\rho} \mid \pi, \text{occ-}\rho\} \\ & \geq (1-P_\rho^{\pi'})E\{R \mid \pi', \neg\text{occ-}\rho\} + P_\rho^{\pi'} E\{R_{pre-\rho} \mid \pi', \text{occ-}\rho\} + P_\rho^{\pi'} E\{R_{post-\rho} \mid \pi', \text{occ-}\rho\} \end{aligned}$$

Observe that P_ρ^π, $E\{R \mid \pi, \neg\text{occ-}\rho\}$, and $E\{R_{pre-\rho} \mid \pi, \text{occ-}\rho\}$ cannot depend on the action taken in partition ρ, hence these terms are the same for π and π'. Thus:

$$\begin{aligned} & (1-P_\rho^\pi)E\{R \mid \pi, \neg\text{occ-}\rho\} + P_\rho^\pi E\{R_{pre-\rho} \mid \pi, \text{occ-}\rho\} + P_\rho^\pi E\{R_{post-\rho} \mid \pi, \text{occ-}\rho\} \\ & \geq (1-P_\rho^\pi)E\{R \mid \pi, \neg\text{occ-}\rho\} + P_\rho^\pi E\{R_{pre-\rho} \mid \pi, \text{occ-}\rho\} + P_\rho^\pi E\{R_{post-\rho} \mid \pi', \text{occ-}\rho\} \\ \iff \quad & E\{R_{post-\rho} \mid \pi, \text{occ-}\rho\} \geq E\{R_{post-\rho} \mid \pi', \text{occ-}\rho\} \\ \iff \quad & Q^\pi(\rho, g(\rho)) \geq Q^\pi(\rho, a). \end{aligned}$$

This completes the proof.

Theorem 2 *If Algorithm 1 converges to a policy π (i.e., if π is the current greedy policy at some time and remains the greedy policy forever after) and if standard stochastic approximation conditions hold (as in, e.g., [1]) then π is locally optimal with probability 1.*

Proof sketch: Once the algorithm settles on a policy π, it is simply learning the partition-action values for that policy. Under standard stochastic approximation assumptions, its estimates will converge with probability 1 to the policy's true partition-action values. Since π is greedy with respect to these partition-action values, then by the previous theorem, π is locally optimal.

In Pendrith's and McGarity's terminology [11], the set of locally optimal policies corresponds to the set of "learning equilibria." Many action-value based RL algorithms, such as Q-learning, Sarsa(λ), and TD(λ) do not satisfy this property an may fail to converge or converge to policies that are not locally optimal (see [11] and the Appendices). In this sense, our new algorithm is more sound for learning in POMDPs.

6 Experiments

In this section, we briefly report on experiments testing the convergence properties of the algorithm in Figure 1. This task, modified slightly from Parr and Russell [9], is a small grid-world navigation task. Figure 1a depicts the agent's environment. Each of the 11 open squares represents a possible location of the agent. Each episode starts with the agent in the "Start" square and ends when the agent enters the $+1$ or the -1 square, which give terminal rewards of $+1$ and -1 respectively. All other transitions receive a reward of -0.04. In each state, the agent may take one of four actions: Up, Right, Down, or Left, which move the agent in the named direction with probability 0.8, and in

one of the perpendicular directions, each with probability 0.1. If one of these transitions would take the agent outside the maze or into the black square, the agent stays in place instead. The problem is partially observable because the agent only observes whether or not there is an open square to the left and whether or not there is on open square to the right.

We used Algorithm 1 to learn three different types of policies: reactive policies, policies based on one step of history (i.e., actions are conditioned on the current and previous observations), and policies based on the immediate observation but with a 2-state internal memory. We ran 100 independent runs under each condition, using a learning rate $\alpha = \frac{1}{n}$ for the n^{th} update of an action value. For step 2 of the algorithm, an episode was chosen to be on-policy with probability $\frac{1}{|\Gamma|+1}$, and otherwise the policy $\pi \leftarrow (\rho, a)$ was followed, where ρ was randomly chosen from Γ and a was a randomly-chosen off-policy action. The reactive policy runs lasted 10^5 episodes, and the other runs were given 10^7 episodes. Episodes were terminated if they did not reach a terminal state by time step 1000. During each run, each time the policy changed we computed the value of the policy using an off-line dynamic programming policy evaluation procedure. Local optimality was checked by evaluating all neighboring policies as well.

Figure 1b shows how many of the 100 runs were at a locally optimal policy as a function of the number of episodes. The reactive policy runs converged most quickly. By episode 4,551 the last of the 100 runs reached a locally optimal policy which it did not leave for the remainder of the run. In the one-step history case, only at episode 3,608,635 had all the runs settled on a final, locally optimal policy. Some of the 2-state memory runs had not settled on a policy even after 10 million episodes. The runs that were not at locally optimal policies were, however, near to such a policy. A typical behavior was flip-flopping between 2 or 3 adjacent policies of nearly equal value, one of which was locally optimal. To some extent, these results may simply reflect the sizes of the policy spaces involved. With four base observations and actions, there are $4^4 = 256$ reactive policies. Examining the task reveals that there is only one possible initial observation and 10 possible pairs of successive observations. Thus there are $4^{11} = 4,194,304$ 1-step history policies. With a two state memory there are 8 possible partitions and 8 actions, thus $8^8 = 16,777,216$ possible policies.

Though settling on a policy was somewhat slow, Figure 1c shows that the value of the policies learned, averaged across runs, approaches an asymptotic level much earlier. For the reactive policy runs, value was near its final level by episode 1000, and for the other two conditions, by episode 10,000. Interestingly, all the reactive policy runs settled on the same policy, which had value 0.243253. Of the one-step history runs, 75 settled on a policy with value 0.622275 and the remainder on a policy with value 0.303774. Of the 2-state memory runs that ended at locally optimal policies, 11 distinct policies were observed, ranging in value from 0.118621 to 0.622275.

Although not all of the 2-state memory runs converged to a single policy, the runs that did not converge seemed to be near to local optima. And because the curve in Figure 1 has an upward trend, we do not take this as contradicting our conjecture that Algorithm 1 converges, in general, to a locally optimal policy. Since the values of the policies learned increases quickly, we believe the algorithm is a feasible one. Naturally, comparison to other techniques is in order.

7 Conclusion

We have presented a novel definition of action value and a Monte Carlo learning algorithm based on this definition, which is intended for learning control strategies for POMDPs. This new algorithm learns partition-to-action mappings for deterministic tabular policies—a class which includes policies conditioning action choice on past events and policies with internal finite-state memories. The algorithm bears significant resemblance to Q-learning, Sarsa(λ), TD(λ)-control, and particularly Monte Carlo control with exploring starts, in that it maintains a table of action values which are updated from sample trajectories, it continually updates its current policy to be greedy with respect to the action values, and it occasionally explores actions not estimated to be optimal. However, because it uses our new definition of action value, the algorithm also satisfies theoretical conditions that are necessary, though not sufficient, to guarantee convergence to a locally optimal policy. The

other RL algorithms mentioned above, for example, do not satisfy these necessary conditions, and for some POMDPs they converge to policies that are not locally optimal or fail to converge to any policy. Experiments showed the feasibility of our new algorithm, though studies on more POMDPs and comparisons to existing algorithms are in order. An obvious next step in the theoretical analysis is to establish conditions under which the algorithm we have presented can be guaranteed to converge.

Acknowledgments

This work was funded by the National Science Foundation under grant numbers ECS-9980062 and ECS-0070102.

References

[1] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. 1*. Athena Scientific, 1995.

[2] G. Gordon. Chattering in sarsa(λ). CMU Learning Lab Internal Report. Available at www.cs.cmu.edu/∼ggordon, 1996.

[3] T. Jaakkola, S. Singh, and M. Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *Advances in Neural Information Processing Systems 7*, Cambridge, MA, 1994. MIT Press.

[4] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[5] M. L. Littman. Memoryless policies: Theoretical limitations and practical results. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, Cambridge, MA, 1994. MIT Press.

[6] J. Loch and S. Singh. Using eligibility traces to find the best memoryless policy in a partially observable markov decision process. In *Proceedings of the Fifteenth International Conference on Machine Learning*, San Francisco, CA, 1998. Morgan Kaufmann.

[7] A. K. McCallum. *Reinforcement learning with selective perception and hidden state*. PhD thesis, Computer Science Department, University of Rochester, 1995.

[8] N. Meuleau, L. Peshkin, K. E. Kim, and L. P. Kaelbling. Learning finite-state controllers for partially observable environments. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 427–436, San Francisco, CA, 1999. Morgan Kaufmann.

[9] R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, San Francisco, CA, 1995. Morgan Kaufmann.

[10] M. Pendrith. Personal communication.

[11] M. D. Pendrith and M. J. McGarity. An analysis of direct reinforcement learning in non-markovian domains. In *Machine Learning: Proceedings of the 15th International Conference*, pages 421–429, 1998.

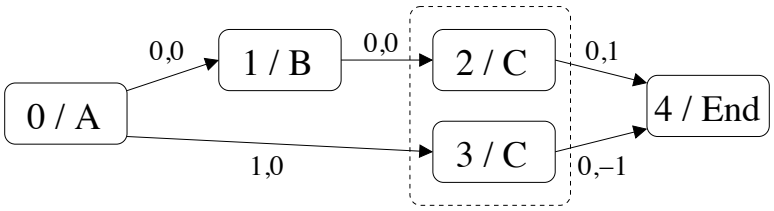


Figure 2: Counterexample to convergence of Q-Learning and Sarsa

Appendix A: Non-Convergence of Q-Learning and Sarsa

Consider the POMDP depicted in Fig. 2. The rounded boxes represent states, and are labelled with a state number (0...4) and with the observation that the agent receives when the environment is in that state (A, B, C, or End). The dashed box around states 2 and 3 shows the fact that they appear the same to the agent. The arrows represent actions, and are labelled with an action number and the immediate reward received upon taking that action. For example, from state 0, there are two actions available, 0 and 1, and both give immediate reward 0. Episodes start in state zero and end in state 4. We assume $\gamma < 1$.

Suppose Q-learning with $\varepsilon-$greedy action selection is used to learn a reactive policy for this POMDP. Q-learning maintains an observation-action value function $Q(o,a)$. It updates this function as follows. When the agent receives observation o, takes action a, receives reward r and reaches a terminal state, it updates Q as: $Q(o,a) \leftarrow (1-\alpha)Q(o,a) + \alpha r$. If, instead of reaching a terminal state, the agent receives a non-terminal observation o', it updates Q as: $Q(o,a) \leftarrow (1-\alpha)Q(o,a) + \alpha(r + \gamma \max_{a'} Q(o',a'))$. Under $\varepsilon-$greedy action selection, the agent chooses the action with highest Q-value with probability $(1-\varepsilon)$ and chooses a different action with probability ε. We assume $\varepsilon \in (0, 0.5)$.

State 0 is the only state with any choice of action. Let A0 denote the policy of choosing action 0 from this state ("A0" because the agent sees observation A). Let A1 denote the policy of choosing action 1 from state 0. Suppose Q-learning converges to policy A0—meaning that at some time, A0 is the greedy policy and remains greedy forever after. Because of its action selection procedure, state 2 would occur in an episode with probability $(1-\varepsilon)$. State 3 would occur with probability ε. Thus observation C would be followed by a reward of $(1-\varepsilon)1 + \varepsilon(-1) = 1 - 2\varepsilon$ on average. The partition-action value $Q(C,0)$ would converge to this value. Working backwards, we find that Q-learning would learn the observation-action values: $Q(B,0) = \gamma(1-2\varepsilon)$, $Q(A,0) = \gamma^2(1-2\varepsilon)$, and $Q(A,1) = \gamma(1-2\varepsilon)$. Since $\gamma < 1$ and $\varepsilon < 0.5$, then $\gamma^2(1-2\varepsilon) < \gamma(1-2\varepsilon)$. This contradicts the supposition that Q-learning converges to policy A0.

By similar reasoning, if we assume Q-learning converges to policy A1, it would learn partition-action values: $Q(C,0) = \varepsilon 1 + (1-\varepsilon)(-1) = 2\varepsilon - 1$, $Q(B,0) = \gamma(2\varepsilon - 1)$, $Q(A,0) = \gamma^2(2\varepsilon - 1)$, and $Q(A,1) = \gamma(2\varepsilon - 1)$. Since $\gamma < 1$ and $\varepsilon < 0.5$, $\gamma^2(2\varepsilon - 1) > \gamma(2\varepsilon - 1)$. This contradicts the assumption that Q-learning converges to policy A1. Thus, Q-learning cannot converge to either policy. Note that these are policies of much different value according to our definition; this is not an example of relatively harmless chattering among policies of equal value.

We note that the same argument works for Sarsa with $\varepsilon-$greedy action selection, because for this POMDP the updates that Sarsa and Q-learning make are the same. The argument can be generalized to any form of action selection, as long as the greedy action is always more likely to be selected than the non-greedy action, and as long as the non-greedy action is selected infinitely often (so that its value can be learned.)

This example does not rely on our definition of action value. Action 0 from observation A is always followed by a discounted return of γ^2. Action 1 is always followed by a discounted return of $-\gamma$. Standard Monte Carlo action evaluation would quickly uncover which action is better. This is simply an example where Monte Carlo updating outperforms temporal difference-style updating. Other researchers have noted this phenomenon before, e.g. [11].

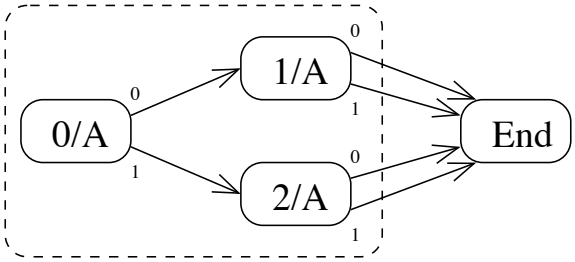


Figure 3: Counterexample to convergence of Sarsa(λ).

Appendix B: Non-Convergence of Sarsa(λ)

In this appendix we demonstrate a POMDP on which ε-greedy Sarsa(λ) for learning reactive policies cannot converge to a single policy, for any λ and all $0 < \varepsilon < \varepsilon_0$ for some $\varepsilon_0 > 0$.

Consider the POMDP depicted in Figure 3. The rounded boxes represent states. There are three non-terminal states: 0, 1, and 2, which all give observation A, and there is a terminal state, End. From any non-terminal state there are two actions allowed, 0 and 1. Transitions are deterministic as indicated by the arrows. From state 0, either action receives zero immediate reward. Actions 0 and 1 from state 1 receive immediate rewards r_0 and r_1 respectively. Actions 0 and 1 from state 2 receive immediate rewards r_2 and r_3 respectively. In other words, the POMDP can be viewed as implementing the following reward structure:

Action sequence taken	Terminal reward
0 0	r_0
0 1	r_1
1 0	r_2
1 1	r_3

Let $r_0 < r_3 < r_1 = r_2$ and let there be no discounting of returns. The gist of the proof is as follows. Since all states yield the same observation, there are only two deterministic reactive policies: take action 0 always or take action 1 always. Now, suppose ε-greedy Sarsa(λ) *were* to converge to the policy of always taking action 0, and that ε is small. The algorithm learns action values for both actions, let us call them Q_0 and Q_1. Action 0 occurs most frequently in the action sequence 0 0, which happens with probability $(1-\varepsilon)^2$. The action sequences 0 1 and 1 0 are relatively much less frequent, and so the learned value of Q_0 would be very near to r_0. On the other hand, action 1 occurs most frequently via the action sequences 0 1 or 1 0, and relatively rarely via the sequence 1 1. So the valued learned for Q_1 would be near to $r_1 = r_2$. But then Q_1 would be greater than Q_0, contradicting our assumption that the algorithm converges to the policy of always taking action 0. Likewise, it is also inconsistent for the algorithm to converge to the policy of always taking action 1. Thus, the algorithm must chatter between the two policies indefinitely. The remainder of this document formalizes this argument.

First, consider the case of replacing traces. Let us suppose the learner takes action 0 with probability p and action 1 with probability q=1-p. ε-greedy behavior when $Q_0 \geq Q_1$ is then represented by the choice $p = 1-\varepsilon$. For $Q_1 > Q_0$, $p = \varepsilon$ describes the learner's action selection. The table below lists the possible trajectories, along with their probabilities of occurrence, and the updates made to Q_0 and Q_1.

Action sequence	Probability	Updates
0 0	p^2	$Q_0 \leftarrow Q_0 + \alpha(0 + Q_0 - Q_0)$, $Q_0 \leftarrow Q_0 + \alpha(r_0 - Q_0)$
0 1	pq	$Q_0 \leftarrow Q_0 + \alpha(0 + Q_1 - Q_0)$, $Q_0 \leftarrow Q_0 + \alpha\lambda(r_1 - Q_1)$, $Q_1 \leftarrow Q_1 + \alpha(r_1 - Q_1)$
1 0	pq	$Q_1 \leftarrow Q_1 + \alpha(0 + Q_0 - Q_1)$, $Q_1 \leftarrow Q_1 + \alpha\lambda(r_2 - Q_0)$, $Q_0 \leftarrow Q_0 + \alpha(r_2 - Q_0)$
1 1	q^2	$Q_1 \leftarrow Q_1 + \alpha(0 + Q_1 - Q_1)$, $Q_1 \leftarrow Q_1 + \alpha(r_3 - Q_1)$

Collecting the updates for Q_0, we find the fixed point is at:

$$p^2(r_0 - Q_0) + pq(Q_1 - Q_0) + pq\lambda(r_1 - Q_1) + pq(r_2 - Q_0) = 0$$

or,

$$Q_0 = \frac{pr_0 + q((1-\lambda)Q_1 + \lambda r_1 + r_2)}{p + 2q}$$

And for Q_1 we have:

$$pq(r_1 - Q_1) + pq(Q_0 - Q_1) + pq\lambda(r_2 - Q_0) + q^2(r_3 - Q_1)$$

or,

$$Q_1 = \frac{qr_3 + p(r_1 + (1-\lambda)Q_0 + \lambda r_2)}{q + 2p}$$

Note that as $p \to 1$, $Q_0 \to r_0$, and $Q_1 \to \frac{1}{2}(r_1 + (1-\lambda)Q_0 + \lambda r_2) = \frac{1}{2}(r_1 + (1-\lambda)r_0 + \lambda r_2) \geq \frac{1}{2}(r_1 + r_0) > r_0 = Q_0$. Note that since this inequality is strict and holds as $p \to 1$, then it holds for some range of p near 1, by continuity of the solutions Q_0 and Q_1 to the above equations. The punchline is this: if we assume Sarsa(λ) does ε-greedy exploration and that ε is small, and that the algorithm converges to the policy of always taking action 0, then it should learn a value of Q_1 greater than Q_0. This means action 0 is not greedy, contradicting the convergence to that policy.

Similarly, as $p \to 0$, $Q_1 \to r_3$ and $Q_0 \to \frac{1}{2}((1-\lambda)Q_1 + \lambda r_1 + r_2) = \frac{1}{2}((1-\lambda)r_3 + \lambda r_1 + r_2) \geq \frac{1}{2}(r_3 + r_2) > r_3 = Q_1$. Thus, the algorithm cannot converge to the policy of always taking action 1. The algorithm cannot converge to either policy, and instead must chatter between the two.

The argument for accumulating traces is similar in form, differing only in the exact calculations. The table below lists the possible trajectories, along with their probabilities of occurrence, and the updates made to Q_0 and Q_1.

Action sequence	Probability	Updates
0 0	p^2	$Q_0 \leftarrow Q_0 + \alpha(0 + Q_0 - Q_0)$, $Q_0 \leftarrow Q_0 + \alpha(1+\lambda)(r_0 - Q_0)$
0 1	pq	$Q_0 \leftarrow Q_0 + \alpha(0 + Q_1 - Q_0)$, $Q_0 \leftarrow Q_0 + \alpha\lambda(r_1 - Q_1)$, $Q_1 \leftarrow Q_1 + \alpha(r_1 - Q_1)$
1 0	pq	$Q_1 \leftarrow Q_1 + \alpha(0 + Q_0 - Q_1)$, $Q_1 \leftarrow Q_1 + \alpha\lambda(r_2 - Q_0)$, $Q_0 \leftarrow Q_0 + \alpha(r_2 - Q_0)$
1 1	q^2	$Q_1 \leftarrow Q_1 + \alpha(0 + Q_1 - Q_1)$, $Q_1 \leftarrow Q_1 + \alpha(1+\lambda)(r_3 - Q_1)$

Collecting the updates for Q_0, we find the fixed point is at:

$$p^2(1+\lambda)(r_0 - Q_0) + pq(Q_1 - Q_0) + pq\lambda(r_1 - Q_1) + pq(r_2 - Q_0) = 0$$

or,

$$Q_0 = \frac{p(1+\lambda)r_0 + q((1-\lambda)Q_1 + \lambda r_1 + r_2)}{p(1+\lambda) + 2q}$$

And for Q_1 we have:

$$pq(r_1 - Q_1) + pq(Q_0 - Q_1) + pq\lambda(r_2 - Q_0) + q^2(1+\lambda)(r_3 - Q_1)$$

or,

$$Q_1 = \frac{q(1+\lambda)r_3 + p(r_1 + (1-\lambda)Q_0 + \lambda r_2)}{q(1+\lambda) + 2p}$$

Examining the limits as $p \to 1$ reveals that they are the same as in the replacing traces case, thus the same argument that completed the previous argument works here as well.

Discussion

Note that since $r_0 \neq r_1$, the example presented is not one of relatively harmless chattering among policies of equal value. The least flattering way of describing the example is that it shows Sarsa(λ) oscillating between the worst and best policies in the space.

In some cases, one desires to let the exploration parameter ε go to zero at a suitable rate. The theorem we have proved shows that for any fixed, small ε, the algorithm will chatter. Since this is true of all $\varepsilon \in (0, \varepsilon_0)$ for some $\varepsilon_0 > 0$, it is not difficult to extend the result to the case where ε goes to zero in a suitable fashion.

The proof relies on the fact that ε-greedy action selection is discontinuous in the action values. It is not clear whether a counterexample can be constructed for a method of action selection that is smooth in the action values, such as Gibbs/Boltzmann action selection.

Gordon [2] has presented a similar counterexample and argument (not quite a formal proof) to the one here, along with simulation experiments. The example here improves on his in at least one important way. Gordon demonstrates a POMDP on which Sarsa(0) does not converge, and claims similar counterexamples can be constructed for any $\lambda \in [0, 1)$. Our single counterexample applies for any $\lambda \in [0, 1]$. This is an important distinction because it has been conjectured (by, among others, Pendrith [10]) that for any POMDP, Sarsa(λ) might be convergent for λ sufficiently close to 1, where "sufficiently close" has a problem-dependent meaning. Our example shows that this is not the case.

We also note that the non-convergence proven here can be trivially interpreted as a non-convergence result for Sarsa(λ) on MDPs with state-aggregation function approximation, or more generally, with linear function approximation, of the action values.