

Consistent exploration improves convergence of reinforcement learning on POMDPs

Paul A. Crook

Institute of Perception, Action and Behaviour,
School of Informatics, University of Edinburgh,
Edinburgh. EH9 3JZ, UK
paulc@dai.ed.ac.uk

Gillian Hayes

Institute of Perception, Action and Behaviour,
School of Informatics, University of Edinburgh,
Edinburgh. EH9 3JZ, UK
gmh@inf.ed.ac.uk

ABSTRACT

This paper sets out the concept of consistent exploration of observation-action pairs. We present a new temporal difference algorithm, CEQ(λ), based on this concept and demonstrate using a randomly generated set of partially observable Markov decision processes (POMDPs) that it outperforms SARSA(λ). This result should generalise to any POMDP where satisficing policies which map observations to actions exists. We also set out reasons for preferring CEQ(λ) over an alternative Monte-Carlo style algorithm, MCESP, when working in the robotics domain.

1. INTRODUCTION

Partially observable Markov decision processes (POMDPs) represent a broad class of everyday problems faced in the real world. For example, to a robot navigating an office complex, two T-junctions in different parts of the building may appear identical. The underlying building navigation problem would be Markovian if the robot could uniquely identify every location, but sensor limitations make the problem a POMDP.

Reinforcement learning (RL) is popular for solving Markov decision processes (MDPs), but if we apply RL to solving POMDPs all the nice policy convergence guarantees are lost [12]. Nor does the performance of RL degrade gracefully as the degree of non-Markovianity increases – [12] demonstrates a simple example in which the inability of an agent to distinguish between just two states leads to an arbitrarily high reduction in the value of the policies that it is possible to learn. Despite these discouraging results, empirical tests using the temporal difference algorithm SARSA(λ) [14, p181], have demonstrated good results on many POMDP tasks [5, 9]. In fact [9] refers to results with SARSA(λ) as a “high standard to compare to,” and [5] “that SARSA(λ) may be hard to beat in problems where there exists a good policy that maps observation space to actions.” We seek to demonstrate below an algorithm which, in terms of the *reliability* with which policies converge to satisficing solutions,

outperforms SARSA(λ).

In MDPs, there is no differentiation between the agent’s current observation and current state. This does not hold for POMDPs, so we explicitly differentiate between the current observations o and the current states s , where $o \in$ set of all possible observations O and $s \in$ set of all possible world states S . For POMDPs a RL algorithm typically learns observation-action values $Q(o, a)$ which are then used to define the agent’s policy π . Generally the policy is one of selecting the action which maximises future estimated reward given the current observation o , *i.e.* $a = \arg \max_b Q(o, b)$ (where $a \in$ the set of actions available to the agent A). We use the term *reactive policies* to refer to policies that map observations directly to actions without attempting to discern the underlying state of the problem. Such policies are also known as memoryless policies but we consider that reactive provides a better description. For examples of RL algorithms that do attempt to discern the underlying Markovian state of a POMDP, see [16, 6].

2. CONSISTENT EXPLORATION

The concept that we term *consistent exploration* originates in work by Perkins [9] which presented a new algorithm *Monte-Carlo Exploring Starts for POMDPs* (MCESP). One of the key steps in setting up the MCESP algorithm is a redefinition of observation-action values. They are defined as “the expected return that follows the first occurrence of o_e , if the agent takes action a_e whenever it observes o_e and adheres to π otherwise” [9], where, in the context of this paper, a_e is an action in A which differs from the action which would be selected if the agent were simply following its policy π . Thus a_e is an exploratory action - we signify this with the subscript ‘e’. The *particular* observation for which this exploratory action was select is represented by o_e . The key part of this definition is the concept that instead of the agent executing an exploratory action a_e and then following π , it takes action a_e and then follows modified policy $\pi \leftarrow (o_e, a_e)$. Even though [9] does not dwell on this issue, we note that the maintaining consistency in the association of actions with observations improves the estimation of observation-action values. Consistency ensures that the values returned are based on the agent acting as if it has permanently modified its policy.

Figure 1 illustrates the difference between the conventional definition of state-action values and the revised definition of observation-action values. Figure 1(a) is a POMDP with two locations that appear identical to an agent. The agent has randomly chosen exploratory action move South

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS’07 May 14–18 2007, Honolulu, Hawai’i, USA.

Copyright 2007 IFAAMAS .

in the leftmost of these aliased locations. Having executed this action the agent then follows policy and arrives at the goal state in a total of five moves. Moving South from this state thus appears to be better than moving North which takes a total of nine moves to reach the goal. However, if the agent were to adopt moving South when it encounters this observation as policy, it would be impossible for it to reach the goal, see figure 1(b). We argue that the value of the exploratory action move South should not be estimated on the basis of following the current policy, but instead following the modified policy that incorporates move South for this observation, *i.e.* $\pi \leftarrow (o_e, a_e)$. In this example the estimated cost of moving South should reflect the impossibility of reaching the goal.

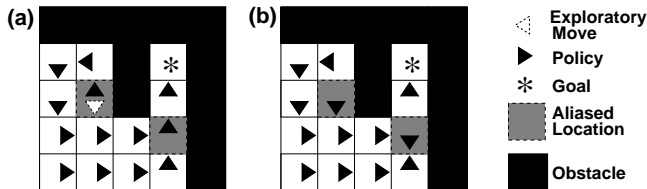


Figure 1: Two examples which demonstrate how consistent exploration can improve observation-action value estimates.

The MCESP algorithm implements the idea of *consistent exploration* by selecting a single observation-action pair to explore and then following the modified policy $\pi \leftarrow (o_e, a_e)$ for the duration of one episode. We generalise this by considering an RL algorithm that can select several exploratory actions during an episode, but remain consistent with itself between exploratory actions. For example, having selected exploratory action a_{e1} for observation o_{e1} it then follows the modified policy $\pi \leftarrow (o_{e1}, a_{e1})$ until at random it selects a new exploratory action a_{e2} for observation o_{e2} . From that point on it then follows policy $\pi \leftarrow (o_{e2}, a_{e2})$ where π is the original, unmodified policy which existed before the selection of any exploratory actions.

Algorithm 1 presents a consistent exploration version of Q-learning, CEQ(λ). This is based on the tabular versions of Watkins’s Q(λ) algorithm [14, p.184] with two modifications: (i) the addition of consistent exploration; (ii) the next action to be executed a' is not selected until after the observation-action values have been updated. The latter modification results in a computationally less efficient algorithm, as it now loops twice through the set of observation-action pairs; once to update the Q values and then later to update the eligibility trace values. The benefit of this arrangement is that more up to date Q-values are used to select the next action. However, empirical tests indicate that this in itself does not result in consistently better performance.

Consistent exploration is implemented by storing the last observation o_e for which an exploratory action a_e was executed. These two variables are used to modify the policy such that the policy being followed by the agent is $\pi \leftarrow (o_e, a_e)$. These two variables, a_e and o_e , are updated every time an action is chosen which does not match $\pi \leftarrow (o_e, a_e)$. When such an exploratory move occurs, the eligibility trace is also reset. Resetting the eligibility trace ensures that Q-values are only backed-up over observation-action pairs that occurred when a particular modified policy was being fol-

Algorithm 1 Consistent Exploration form of Q-learning, CEQ(λ).

```

Initialise  $Q(o, a)$  arbitrarily for all  $o, a$ 
Repeat (for each episode)
  Initialise eligibility trace  $e(o, a) = 0$  for all  $o, a$ ;
  Initialise start location and get observation  $o$ 
  Select  $a$  using policy derived from Q values (e.g.
     $\epsilon$ -greedy)
  # initiate  $a_e$  and  $o_e$  according to whether or not
    algorithm starts with an exploratory action
  if  $a \neq \arg \max_b Q(o, b)$  then  $a_e \leftarrow a$ ;  $o_e \leftarrow o$ 
    else  $a_e \leftarrow \text{null}$ ;  $o_e \leftarrow \text{null}$ 
  Repeat (for each step of episode)
    take action  $a$ , obtain  $r$  and observe  $o'$ 
    # update Q-values based on optimal policy
     $\pi^* = \arg \max_b Q(o', b)$  modified to be consistent
    with the current exploratory observation
    and action  $\pi^* \leftarrow (o_e, a_e)$ 
    if  $o' = o_e$  then  $a^* \leftarrow a_e$ 
      else  $a^* \leftarrow \arg \max_b Q(o', b)$ 
     $\delta = r + \gamma Q(o', a^*) - Q(o, a)$ 
     $e(o, a) \leftarrow 1$ 
    for all  $o, a$ ;  $Q(o, a) \leftarrow Q(o, a) + \alpha \delta e(o, a)$ 
    choose next action to execute  $a'$  for current
      observation  $o'$  using policy derived from up-
      dated Q values (e.g.  $\epsilon$ -greedy)
    # update eligibility trace
    for all  $o, a$ ;
      if  $a' = a^*$  then  $e(o, a) \leftarrow \gamma \lambda e(o, a)$ 
        else  $e(o, a) \leftarrow 0$ 
      # reset trace if action  $a'$  is exploratory, i.e.
        not consistent with  $\pi^* \leftarrow (o_e, a_e)$ 
    # if action exploratory record new exploratory
      action and observation
    if  $a' \neq a^*$  then  $a_e \leftarrow a'$ ,  $o_e \leftarrow o'$ 
       $o \leftarrow o'$ ;  $a \leftarrow a'$ 
  until terminal state is reached

```

lowed, *i.e.* values for $Q(o_{e1}, a_{e1})$ would be updated based on rewards which occurred whilst the policy $\pi \leftarrow (o_{e1}, a_{e1})$ is being followed but not when policy $\pi \leftarrow (o_{e2}, a_{e2})$ is subsequently followed.

3. COMPARISON WITH SARSA(λ)

Our contention is that CEQ(λ) in receiving more consistent estimates of the value of policy changes, will, when exploration ceases¹, tend to converge on satisfying policies more reliable than SARSA(λ). We define satisfying policies as policies that successfully reach the goal from every possible starting location.

To test the above statement we compare CEQ(λ)’s performance with that of SARSA(λ) using a set of randomly generated grid worlds, three agents that have differing abilities, across a range of values of three parameters; the learning

¹Examples presented by [15] and [10] demonstrate POMDP problems on which there can be no stable policy whilst there remains some possibility of exploration. However, experimental work that we have done (not presented here as it lies outside the scope of this paper) suggest that stable policies do exist when exploration ceases.

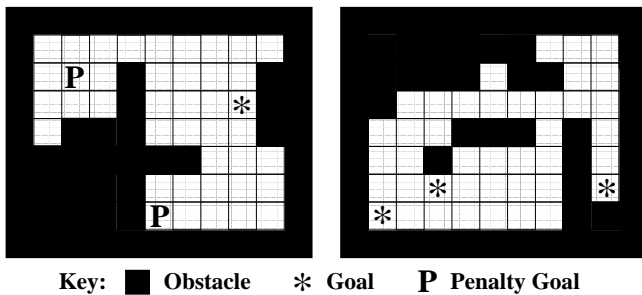


Figure 2: Two example random grid worlds; left is random world number 9, right is random world number 10

rate α , discount rate γ and eligibility trace value λ . We use a set of random generated grid worlds in order to establish an indication of the generality of the results.

3.0.1 Randomly Generated Grid Worlds

Each grid world used was generated through the random placement of positive goals, penalty goals and obstacles on a 7×9 grid of cells bounded by walls on all four sides. The number of positive goals (minimum of one), penalty goals and obstacles, varying in size from 1×5 cells down to a single grid cell, were selected at random using Gaussian distributions. The task that the RL algorithm is expected to learn is that of reaching a positive goal from all locations in the world. A check is therefore made that it is possible for the agent to reach a positive goal from each location. Any location where this is not possible is filled in to prevent the agent starting there. Finally, isolated penalty goals are also removed and filled in. Two of the randomly generated worlds are shown in figure 2.

An agent in these worlds can choose between four physical actions; move North, South, East and West. State transitions are deterministic and each action moves it one square in the appropriate direction. If an agent tries to move towards an obstacle or wall its location remains unaltered, although it receives the same penalty as if the action had succeeded. On executing a physical action the agent receives a reward of +10 if that action results in it immediately arriving at a positive goal location (indicated by an asterisk), -100 if it immediately arrives at a penalty goal location (indicated by the letter P), otherwise it receives a penalty of -1. When the agent reaches a goal, it is relocated to a start location selected at random using a uniform distribution across all locations that do not contain obstacles.

The worlds which were generated contained between 3 and 49 locations. The number of locations in each world is, however, not an indication of the complexity of the task the RL algorithm is faced with. This will depend on the number of different observations that a particular agent perceives and the amount of aliasing of locations that occurs. In the ten random grid worlds used here the number of distinct observations per world ranged from 3 up to 38 for the first agent, between 18 to 200 for the second type of agent and the third agent perceives double the number of distinct observations seen by the first agent. Of the two example grid worlds shown in figure 2, the left hand world is the hardest of the ten generated, in terms of the ability of

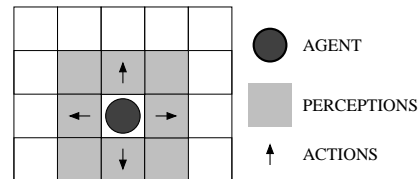


Figure 3: Fixed perception agent

SARSA(λ) and CEQ(λ) to learn satisficing solutions, whilst the right hand world is among the easiest.

3.0.2 The Three Agents

The three agents used differ in the observations that they receive. The first agent is identical as that introduced in Littman’s [4] variation of Sutton’s Grid World [13]. Its perception is formed by testing whether each of the eight squares adjacent to its current location contain obstacles, see figure 3. Limiting the agent’s perception allows the possibility of it being unable to distinguish between various locations and thus, from the agent’s point of view, makes the task of reaching a positive goal state into a POMDP.

The second type of agent used has the same basic input perception as the first, *i.e.* an eight bit string indicating the presence of obstacles immediately surrounding it, but has the ability to obtain alternative views of its current location. This is achieved through the selection of eight *perceptual actions* that are available to it; look North, look North East, look East, etc. On selecting one of these perceptual actions the observation it receives changes to include additional information from three additional squares in the direction of it choosing (*e.g.* the three hatched squares in figure 4). These eight actions are in addition to the four physical actions allowed by the worlds. All actions attract the same penalty (-1) when selected and the RL algorithms make no distinction between physical and perceptual actions. A policy could select a sequence of perceptual actions one after another; in such an event the information is not additive, but limited to the basic eight squares plus the three selected by the last perceptual action. The agent’s observation reverts to the basic eight adjacent squares after it selects a physical action.

The third agent, a 1-bit Memory Agent, generates an observation consisting of the same basic input perception as the first plus an additional internal memory bit. This memory bit can be set or reset by the agent. The memory agent architecture is similar to those found in the literature, *e.g.* [4, 11, 3]. The agent has a set of physical actions that allow it to move through the world. In addition it has a set of “memory + physical actions” that couple movement with “flipping” the state of the memory bit, *i.e.* a bit which has value 0 will be flipped to have value 1, and a bit which has value 1 will be flipped to have value 0.

Use of the latter two agents with any of the worlds has the potential to make the goal seeking task more complex as it increases the observation and action space within which the RL algorithm has to operate. In addition, the agent’s perceptual actions increases the number of ways that each location in the world can be viewed, thus increasing the number of locations where perceptual aliasing can occur. This increase in the observation and action space can sometimes

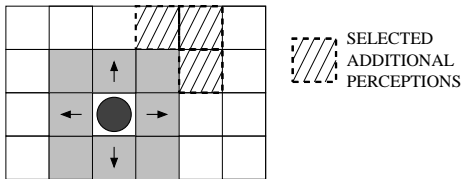


Figure 4: Active-perception agent selecting perceptual action “look North East”

be of advantage to the RL algorithm in allowing alternative satisficing policies to exist.

Overall the combination worlds and agents create a set of problems which are sufficiently complex to contain examples of the different problems that can occur when applying reinforcement learning to POMDPs. There are multiple locations where the agent obtains the same observation, and each of these aliased locations can present different challenges to RL algorithms; see [2] for more details.

3.0.3 Parameter Values

We compare both RL algorithms over all combinations of the following parameter values: learning rate (α) values of 0.01, 0.02, 0.05, 0.1 and 0.3; discount rate (γ) values of 0.8, 0.9, 0.99 and 1.0; eligibility trace (λ) values of 0.8, 0.9 0.99 and 1.0. For each combination of world, agent and parameter values twenty independent trials were run allowing us to record the number of times each algorithm converged to a satisficing policy and the mean total steps required by each satisficing policy. At the start of each trial the Q-values were initiated to random values drawn from a Gaussian distribution with mean zero and standard deviation of five. Initial policies were based on these Q values $\pi \leftarrow \arg \max_{a_x} Q(o_x, a_x)$. Actions were selected in accordance with ϵ -greedy exploration [14, p.122], with ϵ initially valued at 0.2 and decaying linearly to zero by the 500,000th action-learning step. Each trial was run for a total of one million action-learning steps. Both algorithms used replacing traces [14, p.186] for eligibility trace.

3.1 Results and Discussion

First we compare the reliability with which the two direct reinforcement learning algorithms, CEQ(λ) and SARSA(λ) converge to satisficing solutions. Averaging across all combinations of worlds, agents and parameter values, the mean number of satisficing solutions learnt by CEQ(λ) is 19.21 out of a maximum of 20 (or 96%), with a standard deviation of 3.44 (17%). This is noticeably better than SARSA(λ) which scores a mean of 17.16 (86%) with standard deviation 6.29(31%). Student’s t-test for unequal variance indicates this difference is statistically significant with the probability of both sets of results being drawn from the same distribution being equal to 2×10^{-43} .

As well as having better overall averages, CEQ(λ)’s averages are better than SARSA(λ) for each agent design tried, see [1] for details. CEQ(λ)’s performance on this measure is reasonably uniform and better than SARSA(λ) across most of the parameter space that we tested. However there is sudden fall off in its performance for the lowest value of α . Examining the raw data for CEQ(λ) reveals that at this low level of α the policies converge much slower and several have

yet to converge after one million action learning steps. It appears that these policies would have eventually converged to satisficing solutions had the trials been extended and thus CEQ(λ)’s average performance would improve further.

It is interesting to note that CEQ(λ) performs well at values of λ or γ which are equal to 1.0. By contrast SARSA(λ)’s performance tends to fall off if either λ or γ and especially if both equal 1.0. SARSA(λ) appears to have converged across most of the policy space and further improvement in the number of satisficing policies for high values of λ and γ appears unlikely. Although having both γ and λ equal to 1.0 is unusual when applying reinforcement learning to MDPs, theoretical results [7] demonstrate that when learning policies for POMDPs, using undiscounted and Monte Carlo style returns will ensure that the optimal reactive policy (where it exists) lies at a stationary point in terms of Q-value updates. With λ equal 1.0 eligibility traces approximate to a Monte Carlo style approach [14, p.188]. Thus it is of interest to observe the effect of setting both close to, or at 1.

As the policies learnt are not necessarily optimal we also examine the quality of the satisficing policies to see if either algorithm is, on average, significantly worse than the other. The evaluation of a policy consists of placing the agent at every possible location in the grid world where there is no obstacle, and counting how many steps it takes to reach a positive goal state. The total steps taken by the agent is then summed up over all start states. Note that we only include policies that are satisficing. Table 1 reports the average total path length for satisficing policies for each world (averaged over the three agent designs and all parameter combinations). Also shown is the percentage difference in mean path length for CEQ(λ) compared to SARSA(λ) and the statistical significance of comparing the CEQ(λ) and SARSA(λ) samples. As can be seen the average satisficing policy path length learnt by the two algorithms is very close. For three of the worlds, 1, 4 and 10, the results are so similar that we can not even determine with certainty that the samples for CEQ(λ) and SARSA(λ) are drawn from distinct distributions. For the remaining worlds the maximum increase when using CEQ(λ) compared SARSA(λ) is 2.3% and largest reduction is -6.6% . Overall this suggests that the improvement in the number of satisficing policies learnt by CEQ(λ) has had negligible impact on the quality of the policies learnt when compared to SARSA(λ).

Overall the results show that the probability of CEQ(λ) learning satisficing policies is better than SARSA(λ), and with the exception of slow convergence for low α values CEQ(λ)’s performance is robust over a wide range of parameter values. Robustness to variation in parameter values is useful in practical applications, like say training a real world robot, where having to repeat training with various parameter settings in order to find those that converge is time consuming and sometimes impractical.

We have also run a few trials with MDPs. These show that CEQ(λ) reliably learns satisficing though not necessarily optimal policies on this class of problem.

4. COMPARISON WITH MCESP

The main reason for preferring CEQ(λ) over MCESP is that when using CEQ(λ) we avoid the issue of imposing an artificial maximum length on episodes. Consider the case when MCESP explores a poor policy $\pi \leftarrow (o_e, a_e)$ which fails to reach the goal and ends up in a infinite loop. MCESP only

Random World	Mean total SARSA(λ)	path length CEQ(λ)	percentage difference	statistical significance
1	257.0 \pm 50.9	255.2 \pm 56.7	-0.7%	0.11
2	137.1 \pm 44.7	132.9 \pm 38.8	-3.1%	9×10^{-7}
3	52.2 \pm 9.0	51.1 \pm 6.4	-2.1%	3×10^{-12}
4	395.2 \pm 65.4	396.2 \pm 61.2	+0.3%	0.44
5	350.6 \pm 73.1	327.3 \pm 51.9	-6.6%	3×10^{-66}
6	393.7 \pm 88.7	385.9 \pm 63.1	-2.0%	7×10^{-5}
7	277.0 \pm 49.7	283.4 \pm 53.2	+2.3%	2×10^{-9}
8	2.1 \pm 0.3	2 \pm 0	-4.0%	4×10^{-64}
9	258.0 \pm 53.2	261.9 \pm 56.9	+1.5%	0.02
10	170.2 \pm 31.6	170.5 \pm 35.9	+0.1%	0.73

Table 1: Mean path length of satisficing solutions for SARSA(λ) and CEQ(λ)

updates its policy and chooses a new action to explore after each episode. Hence a limit on the episode is required to ensure that loops are broken. CEQ(λ) on the other hand is free to select a new exploratory action at any point and thus will break out of infinite loops of its own accord. Imposing a maximum limit tends to require some knowledge of the potential length of solutions. Setting this limit too high will slow learning, setting this limit too low will restrict the solutions that can be explored. We prefer to sidestep this issue by using CEQ(λ) for which no distinct segregation of the problem into episodes is required.

One would expect policies learnt by MCESP to converge more slowly than CEQ(λ). The former only updates its observation-action values and policies at the end of episodes, while the latter updates both after each action-learning step. To test this we compared their performance on POMDP version of Sutton’s Grid World as developed by [4]. Both algorithms were tested with α values of 0.01 and 0.1, with $\gamma = 1$. For CEQ(λ) $\lambda = 1$ and exploration controlled using ϵ -greedy; ϵ initiated at 0.2 and decaying linearly to zero by the 500,000th action-learning step. For MCESP policy update threshold is 0.001 and maximum episode length is 50 steps. Policy and observation-action values were initiated at random and for each combination of parameters 100 independent runs were made. Each run was evaluated every 10,000 action-learning steps.

Results are shown in figure 5. In this figure we plot the mean total steps against the number of action-learning steps. The mean total steps is calculated across *all* policies. Evaluation of policies that fail to reach the goal from a given start location is curtailed after 1,000 steps. There are a total of 46 starting locations giving a maximum evaluation cost of 46,000 steps for a policy that fails to reach the goal from every start. The optimal reactive policy takes a total of 416 steps from these 46 start locations. As can be observed from figure 5 the average policy learnt by both MCESP and CEQ(λ) quickly converges over time, with MCESP lagging behind CEQ(λ).

5. CONCLUSIONS AND FUTURE WORK

Where deterministic reactive policies exist CEQ(λ) tends to learn satisficing policies more reliably than SARSA(λ). In addition, the policies learnt by CEQ(λ) converge quicker than those learnt MCESP, and CEQ(λ) avoids the issue of fixing the maximum episode length which is required for MCESP. These results support our more general conjecture

that ensuring consistent exploration of observation-action pairs improves the convergence of RL algorithms to satisficing policies on POMDPs.

CEQ(λ) was not specially designed to perform well on grid world so we believe that its performance should generalise to other problems for which deterministic reactive policies exist. One area of future work is to establish the generality of these results on non grid world problems.

It would also be useful to examine CEQ(λ) from a theoretical viewpoint and see if it is possible to establish similar convergence guarantees to those shown by Perkins [9] for MCESP. Pendrith and McGarity [7] and Perkins [8] present simple POMDPs on which SARSA(λ) fails to converge. We show in [1] that similar limitations hold for CEQ(λ). This suggests that *if* convergence guarantees can be shown to exist for CEQ(λ) they will require that $\lambda = \gamma = 1$. In addition, for a guarantee of convergence to *deterministic* policies to exist exploration has to be curtailed, *e.g.* in ϵ -greedy exploration, ϵ needs to decay to zero.

Finally, the concept of consistent exploration can be applied to other RL algorithms, *e.g.* a consistent exploration version of SARSA(λ) where unlike CEQ(λ) the eligibility traces are not reset when a new exploratory action is selected. It would be interesting to test how such algorithms perform.

Acknowledgements.

Paul Crook would like to thank EPSRC without whose funding this work would not be possible.

6. REFERENCES

- [1] P. A. Crook. *Learning in a state of confusion: using active perception and reinforcement learning to learn good POMDP policies based solely on observation*. PhD thesis, Informatics, University of Edinburgh, 2007.
- [2] P. A. Crook and G. Hayes. Could active perception aid navigation of partially observable grid worlds? In *Proceedings of the Fourteenth European Conference on Machine Learning (ECML 2003)*, volume 2837 of *Lecture Notes in Artificial Intelligence*, pages 72–83. Springer-Verlag, 2003.
- [3] P. L. Lanzi. Adaptive agents with reinforcement learning and internal memory. In J.-A. Meyer et al., editors, *From Animals to Animats 6: Proceedings of the Sixth International Conference on the Simulation*

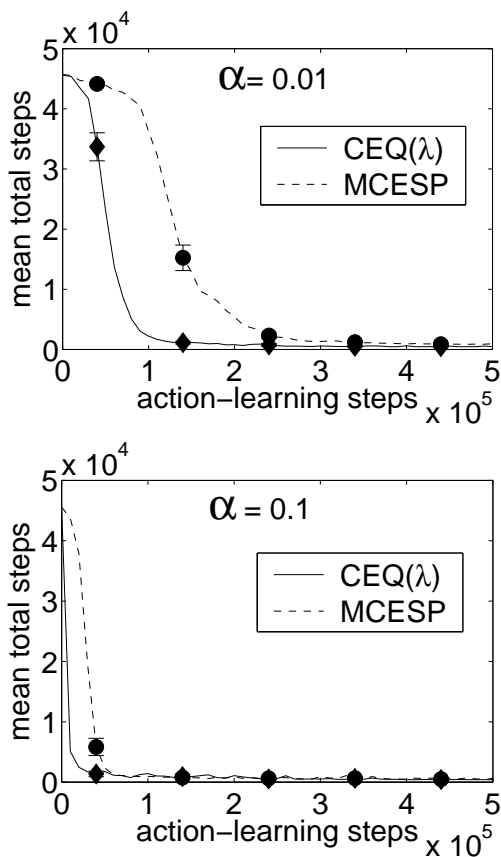


Figure 5: Plot of total steps averaged across all policies against action-learning steps. Error bars show 95% confidence intervals.

of Adaptive Behavior (SAB 2000), pages 333–342. The MIT Press, Cambridge, MA, 2000.

- [4] 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 (SAB'94)*, pages 238–245. The MIT Press, Cambridge, MA, 1994.
- [5] 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 (ICML'98)*, pages 323–331. Morgan Kaufmann, San Francisco, CA, 1998.
- [6] A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Department of Computer Science, Rochester, New York, 1995.
- [7] M. D. Pendrith and M. J. McGarity. An analysis of direct reinforcement learning in non-Markovian domains. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML'98)*, pages 421–420. Morgan Kaufmann, San Francisco, CA, 1998.
- [8] T. J. Perkins. Action value based reinforcement learning for POMDPs. Technical Report UM-CS-2001-20, Department of Computer Science, University of Massachusetts Amherst, 2001.
- [9] T. J. Perkins. Reinforcement learning for POMDPs based on action values and stochastic optimization. In *Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, pages 199–204, Menlo Park, CA, 2002. AAAI Press/MIT Press.
- [10] T. J. Perkins and M. D. Pendrith. On the existence of fixed points for Q-learning and SARSA in partially observable domains. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, pages 490–497, 2002.
- [11] L. Peshkin, N. Meuleau, and L. Kaelbling. Learning policies with external memory. In *Proceedings of the Sixteenth International Conference in Machine Learning (ICML'99)*, pages 307–314. Morgan Kaufmann, 1999.
- [12] 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 (ICML'94)*, pages 284–292. Morgan Kaufmann, San Francisco, CA, 1994.
- [13] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning (ICML'90)*, pages 216–224. Morgan Kaufmann, San Francisco, CA, 1990.
- [14] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.
- [15] S. D. Whitehead and D. H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, 1991.
- [16] S. D. Whitehead and L.-J. Lin. Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73(1-2):271–306, February 1995.