1 Planning and Learning with Tabular Methods

1.1 Exercise 8.1

Q

The non-planning method looks particularly poor in Figure 8.3 because it is a one-step method; a method using multi-step bootstrapping would do better. Do you think one of the multi-step bootstrapping methods from Chapter 7 could do as well as the Dyna method? Explain why or why not.

A

Dyna updates using all past experience so quickly synthesises this into an optimal trajectory. n-step bootstrapping might be slower because it only states visited in the last n steps.

1.2 Exercise 8.2

Q

Why did the Dyna agent with exploration bonus, Dyna-Q+, perform better in the first phase as well as in the second phase of the blocking and shortcut experiments?

A

Increased exploration means Dyna-Q+ finds the optimal policy quicker than Dyna-Q. Dyna-Q may find a trajectory that works but is suboptimal and then have to wait a long time for it to take enough exploratory actions to find an optimal policy.

1.3 Exercise 8.3

Q

Careful inspection of Figure 8.5 reveals that the difference between Dyna-Q+ and Dyna-Q narrowed slightly over the first part of the experiment. What is the reason for this?

A

Dyna-Q+ will take suboptimal actions in order to explore (when τ gets large). Dyna-Q will not do this so has better asymptotic performance.

1.4 Exercise 8.4 (programming)

Q

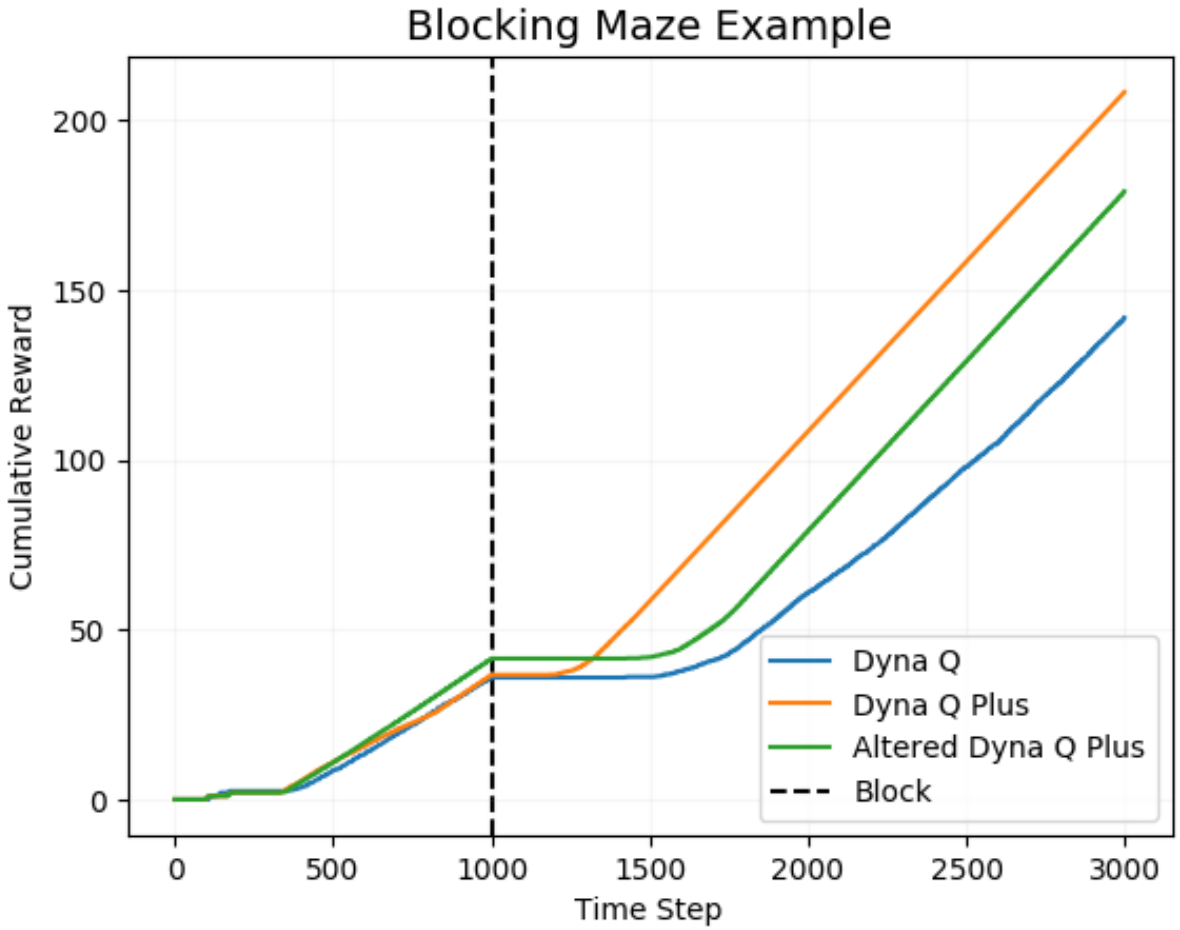
The exploration bonus described above actually changes the estimated values of states and actions. Is this necessary? Suppose the bonus $\kappa\sqrt{\tau}$ was used not in updates, but solely in action selection. That is, suppose the action selected was always that for which $Q(S_t, a) + \kappa\sqrt{\tau(S_t, a)}$ was maximal. Carry out a gridworld experiment that tests and illustrates the strengths and weaknesses of this alternate approach.

A

This is a programming exercise. For the relevant code please see the repo.

The change means that exploration only takes into account the next action, not whole trajectories. In the Dyna-Q+ algorithm can explore whole new paths through the planning stage.

This is backed up by the results. The altered Dyna-Q+ learns the new path slower because it can't plan new trajectories.



1.5 Exercise 8.5

Q

How might the tabular Dyna-Q algorithm shown on page 164 be modified to handle stochastic environments? How might this modification perform poorly on changing environments such as considered in this section? How could the algorithm be modified to handle stochastic environments *and* changing environments?

A

- You could take frequency of occurrences of transitions to estimate the transition probability for the model. (These would be the MLE estimates.)
- Make expected updates when planning.

- This would would present an issue if the environment changed since the changes would just be reflected in changing transition probabilities (which could take a long time to reflect the change in the environment.)
- A solution to this could be to use an exploration bonus to encourage the agent to continue to select various states and keep the model up to date.
- A better solution would be to add some notion of confidence to the model estimates of the transition probabilities. Could model the probabilities like

$$p(s, a, s') = \hat{p}(s, a, s')(1 - \sigma(\tau)) + \sigma(\tau)e,$$

where $\hat{p}$ is the MLE estimate of the probabilities, e is the equiprobable estimate and $\sigma(\tau)$ is a sigmoid of the time since the state-action pair (s, a) was list visited.

1.6 Exercise 8.6

Q

The analysis above assumed that all of the *b* possible next states were equally likely to occur. Suppose instead that the distribution was highly skewed, that some of the *b* states were much more likely to occur than most. Would this strengthen or weaken the case for sample updates over expected updates? Support your answer.

A

If the transition probabilities are skewed then the expected updates perform the same while sample updates get accurate on the most probably outcomes very quickly. This strengthens the case for sample updates.

1.7 Exercise 8.7

Q

Some of the graphs in Figure 8.8 seem to be scalloped in their early portions, particularly the upper graph for $b = 1$ and the uniform distribution. Why do you think this is? What aspects of the data shown support your hypothesis?

A

In the case of the uniform distribution of updates and $b = 1$, the start state is visited roughly once every $|\mathcal{S}|$ updates. When this happens, the action values of the neighbourhood of the start state are updated and their undergo a greater change than when the states that are not in the neighbourhood are the start state are updated. Thus, when the policy is evaluated, the value of the start state changes a lot if the start state has been visited recently, and not so much otherwise (since the change comes from values backed up from states far away from the start state).

In the on-policy case, the start state is visited much more often (on average more than once every 10 updates, since $\mathbb{P}(\texttt{terminate}) = 0.1$) so it does not exhibit this behaviour. When b is larger there are more connections between states, so the neighbourhood of the start state is larger, so this feature is also reduced.

1.8 Exercise 8.8 (programming)

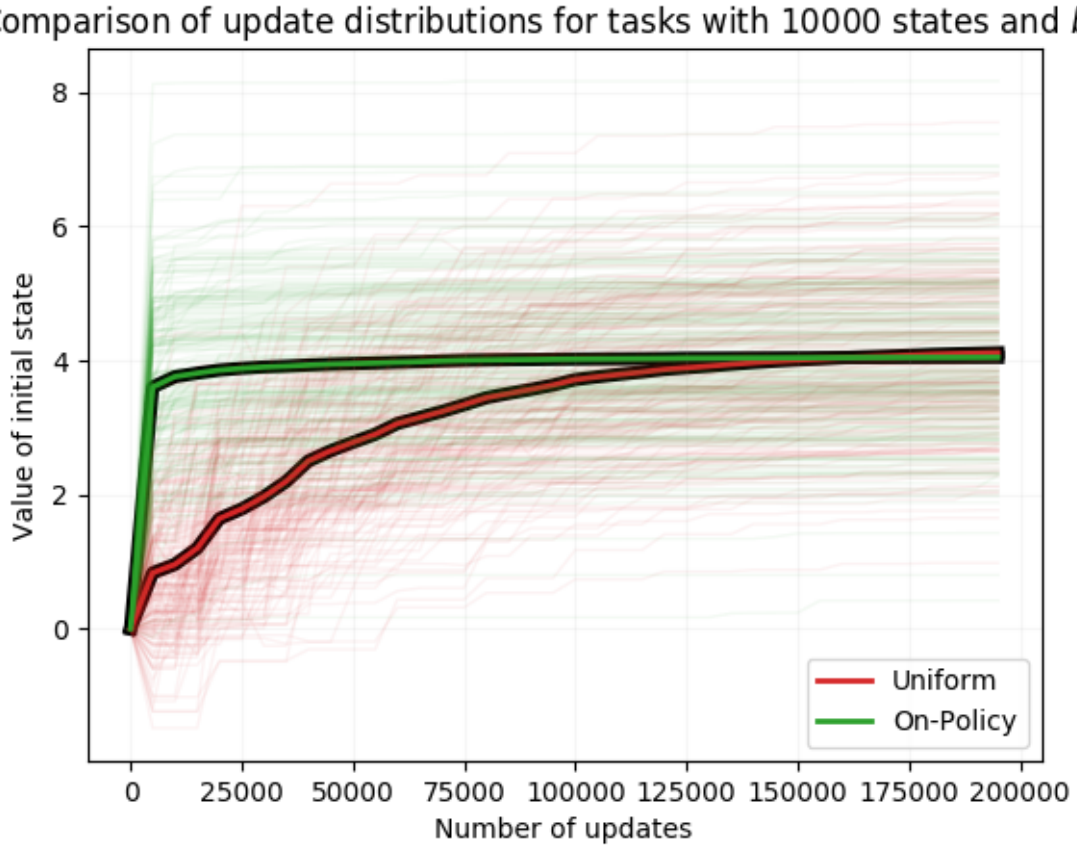
Q

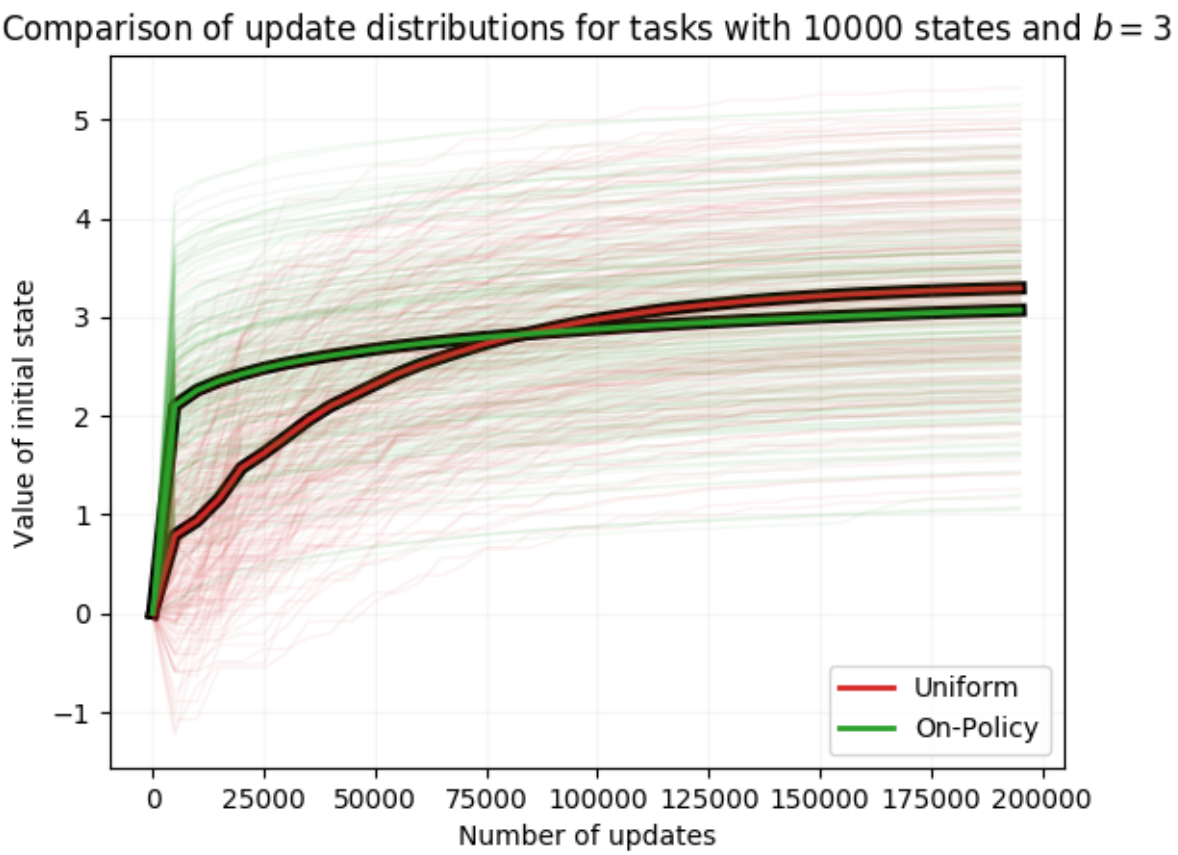
Replicate the experiment whose results are shown in the lower part of Figure 8.8, then try the same experiment but with $b = 3$. Discuss the meaning of your results.

A

This is a programming exercise. For the relevant code please see the repo.

Charts show averages of 200 runs. We see that in the $b = 3$ case the uniformly distributed updates overtakes the on-policy updates much quicker. This is due to the greater complexity of the state-space (the number of states on which the value of the starting state depends is exponential in b), of which the on-policy updates neglects large portions.




Comparison of update distributions for tasks with 10000 states and $b = 3$
Value of initial state
Number of updates
Uniform
On-Policy
5
4
3
2
1
0
−1
0
25000
50000
75000
100000
125000
150000
175000
200000