EXERCISES 16

MAKING COMPLEX DECISIONS

16.1 Sequential Decision Problems

Exercise 16.MDPX

For the 4×3 world shown in Figure 16.1, calculate which squares can be reached from (1,1) by the action sequence $[Right, Right, Right, Up, Up]$ and with what probabilities. Ex-plain how this computation is related to the prediction task (see Section 14.2.1) for a hidden Markov model.

The best way to calculate this is NOT by thinking of all ways to get to any given square and how likely all those ways are, but to compute the occupancy probabilities at each time step. These are as follows:

		Right	*Right*	*Right*	*Up*	*Up*
(1,1)	1	.1	.02	.011	.0075	.00854
(1,2)		.1	.09	.075	.0238	.01076
(1,3)			.01	.010	.0698	.08260
(2,1)		.8	.24	.064	.0779	.06883
(2,3)				.008	.0074	.01810
(3,1)			.64	.256	.0576	.01547
(3,2)				.064	.2112	.06720
(3,3)					.0520	.21130
(4,1)				.512	.0768	.01344
(4,2)					.4160	.49856
(4,3)						.00520

Projection in an HMM involves multiplying the vector of occupancy probabilities by the transition matrix. Here, the only difference is that there is a different transition matrix for each action.

Exercise 16.SEQP

Select a specific member of the set of policies that are optimal for $R(s) > 0$ as shown in Figure 16.2(b), and calculate the fraction of time the agent spends in each state, in the limit, if the policy is executed forever. (*Hint*: Construct the state-to-state transition probability matrix corresponding to the policy and see Exercise MARKOV-CONVERGENCE-EXERCISE.)

If we pick the policy that goes *Right* in all the optional states, and construct the corresponding transition matrix $\mathbf{T}$, we find that the equilibrium distribution—the solution to $\mathbf{Tx} = \mathbf{x}$—has occupancy probabilities of 1/12 for (2,3), (3,1), (3,2), (3,3) and 2/3 for (4,1). These can be found most simply by computing $\mathbf{T}^n\mathbf{x}$ for any initial occupancy vector $\mathbf{x}$, for n large enough to achieve convergence.

Exercise 16.NONS

Suppose that we define the utility of a state sequence to be the *maximum* reward obtained in any state in the sequence. Show that this utility function does not result in stationary preferences between state sequences. Is it still possible to define a utility function on states such that MEU decision making gives optimal behavior?

Stationarity requires the agent to have identical preferences between the sequence pair $[s_0, s_1, s_2, \ldots]$, $[s_0, s'_1, s'_2, \ldots]$ and between the sequence pair $[s_1, s_2, \ldots]$, $[s'_1, s'_2, \ldots]$. If the utility of a sequence is its maximum reward, we can easily violate stationarity. For example,

$$[4, 3, 0, 0, 0, \ldots] \sim [4, 0, 0, 0, 0, \ldots]$$

but

$$[3, 0, 0, 0, \ldots] \succ [0, 0, 0, 0, \ldots] .$$

We can still define $U^\pi(s)$ as the expected maximum reward obtained by executing π starting in s. The agent's preferences seem peculiar, nonetheless. For example, if the current state s has reward R_{max}, the agent will be indifferent among all actions, but once the action is executed and the agent is no longer in s, it will suddenly start to care about what happens next.

Exercise 16.FMDP

Can any finite search problem be translated exactly into a Markov decision problem such that an optimal solution of the latter is also an optimal solution of the former? If so, explain *precisely* how to translate the problem and how to translate the solution back; if not, explain *precisely* why not (i.e., give a counterexample).

A finite search problem (see Chapter 3) is defined by an initial state s_0, a successor function $S(s)$ that returns a set of action–state pairs, a step cost function $c(s, a, s')$, and a goal test. An optimal solution is a least-cost path from s_0 to any goal state.

To construct the corresponding MDP, define $R(s, a, s') = -c(s, a, s')$ unless s is a goal state, in which case $R(s, a, s') = 0$ (see Ex. 17.5 for how to obtain the effect of a three-argument reward function); define $T(s, a, s') = 1$ if $(a, s') \in S(s)$; and $\gamma = 1$. An optimal solution to this MDP is a policy that follows the least-cost path from each state to its nearest goal state.

Exercise 16.REWQ

Sometimes MDPs are formulated with a reward function $R(s, a)$ that depends on the action taken or with a reward function $R(s, a, s')$ that also depends on the outcome state.

a. Write the Bellman equations for these formulations.

b. Show how an MDP with reward function $R(s, a, s')$ can be transformed into a different MDP with reward function $R(s, a)$, such that optimal policies in the new MDP correspond exactly to optimal policies in the original MDP.

c. Now do the same to convert MDPs with $R(s, a)$ into MDPs with $R(s)$.

This is a deceptively simple exercise that tests the student's understanding of the formal definition of MDPs. Some students may need a hint or an example to get started.

a. The key here is to get the max and summation in the right place. For $R(s, a)$ we have

$$U(s) = \max_a [R(s, a) + \gamma \sum_{s'} T(s, a, s')U(s')$$

and for $R(s, a, s')$ we have

$$U(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma U(s')] \; .$$

b. There are a variety of solutions here. One is to create a "pre-state" $pre(s, a, s')$ for every s, a, s', such that executing a in s leads not to s' but to $pre(s, a, s')$. In this state is encoded the fact that the agent came from s and did a to get here. From the pre-state, there is just one action b that always leads to s'. Let the new MDP have transition T', reward R', and discount γ'. Then

$$\begin{aligned}
&T'(s, a, pre(s, a, s')) = T(s, a, s') \\
&T'(pre(s, a, s'), b, s') = 1 \\
&R'(s, a) = 0 \\
&R'(pre(s, a, s'), b) = \gamma^{-\frac{1}{2}} R(s, a, s') \\
&\gamma' = \gamma^{\frac{1}{2}}
\end{aligned}$$

c. In keeping with the idea of part (b), we can create states $post(s, a)$ for every s, a, such that

$$\begin{aligned}
&T'(s, a, post(s, a, s')) = 1 \\
&T'(post(s, a, s'), b, s') = T(s, a, s') \\
&R'(s) = 0 \\
&R'(post(s, a, s')) = \gamma^{-\frac{1}{2}} R(s, a) \\
&\gamma' = \gamma^{\frac{1}{2}}
\end{aligned}$$

Exercise 16.THRC

For the environment shown in Figure 16.1, find all the threshold values for $R(s)$ such that the optimal policy changes when the threshold is crossed. You will need a way to calculate the optimal policy and its value for fixed $R(s)$. (*Hint*: Prove that the value of any fixed policy varies linearly with $R(s)$.)

This can be done fairly simply by:

- Call `policy-iteration` (from `"uncertainty/algorithms/dp.lisp"`) on the Markov Decision Processes representing the 4x3 grid, with values for the step cost ranging from, say, 0.0 to 1.0 in increments of 0.02.
- For any two adjacent policies that differ, run binary search on the step cost to pinpoint the threshold value.
- Convince yourself that you haven't missed any policies, either by using too coarse an increment in step size (0.02), or by stopping too soon (1.0).

One useful observation in this context is that the expected total reward of any fixed policy is linear in r, the per-step reward for the empty states. Imagine drawing the total reward of a policy as a function of r—a straight line. Now draw all the straight lines corresponding to all possible policies. The reward of the *optimal* policy as a function of r is just the max of all these straight lines. Therefore it is a piecewise linear, convex function of r. Hence there is a very efficient way to find *all* the optimal policy regions:

- For any two consecutive values of r that have different optimal policies, find the optimal policy for the midpoint. Once two consecutive values of r give the same policy, then the interval between the two points must be covered by that policy.
- Repeat this until two points are known for each distinct optimal policy.
- Suppose (r_{a1}, v_{a1}) and (r_{a2}, v_{a2}) are points for policy a, and (r_{b1}, v_{b1}) and (r_{b2}, v_{b2}) are the next two points, for policy b. Clearly, we can draw straight lines through these pairs of points and find their intersection. This does not mean, however, that there is no other optimal policy for the intervening region. We can determine this by calculating the optimal policy for the intersection point. If we get a different policy, we continue the process.

The policies and boundaries derived from this procedure are shown in Figure S16.1. The figure shows that there are *nine* distinct optimal policies! Notice that as r becomes more negative, the agent becomes more willing to dive straight into the –1 terminal state rather than face the cost of the detour to the +1 state.

The somewhat ugly code is as follows. Notice that because the lines for neighboring policies are very nearly parallel, numerical instability is a serious problem.

```
(defun policy-surface (mdp r1 r2 &aux prev (unchanged nil))
  "returns points on the piecewise-linear surface
   defined by the value of the optimal policy of mdp as a
   function of r"
  (setq rvplist
    (list (cons r1 (r-policy mdp r1)) (cons r2 (r-policy mdp r2))))
```

```
  (do ()
      (unchanged rvplist)
    (setq unchanged t)
    (setq prev nil)
    (dolist (rvp rvplist)
      (let* ((rest (cdr (member rvp rvplist :test #'eq)))
             (next (first rest))
             (next-but-one (second rest)))
        (dprint (list (first prev) (first rvp)
                      '* (first next) (first next-but-one)))
        (when next
          (unless (or (= (first rvp) (first next))
                      (policy-equal (third rvp) (third next) mdp))
            (dprint "Adding new point(s)")
            (setq unchanged nil)
            (if (and prev next-but-one
                     (policy-equal (third prev) (third rvp) mdp)
                     (policy-equal (third next) (third next-but-one) mdp))
                (let* ((intxy (policy-vertex prev rvp next next-but-one))
                       (int (cons (xy-x intxy) (r-policy mdp (xy-x intxy)))))
                  (dprint (list "Found intersection" intxy))
                  (cond ((or (< (first int) (first rvp))
                             (> (first int) (first next)))
                         (dprint "Intersection out of range!")
                         (let ((int-r (/ (+ (first rvp) (first next)) 2)))
                           (setq int (cons int-r (r-policy mdp int-r))))
                         (push int (cdr (member rvp rvplist :test #'eq))))
                        ((or (policy-equal (third rvp) (third int) mdp)
                             (policy-equal (third next) (third int) mdp))
                         (dprint "Found policy boundary")
                         (push (list (first int) (second int) (third next))
                               (cdr (member rvp rvplist :test #'eq)))
                         (push (list (first int) (second int) (third rvp))
                               (cdr (member rvp rvplist :test #'eq))))
                        (t (dprint "Found new policy!")
                         (push int (cdr (member rvp rvplist :test #'eq))))))
                (let* ((int-r (/ (+ (first rvp) (first next)) 2))
                       (int (cons int-r (r-policy mdp int-r))))
                  (dprint (list "Adding split point" (list int-r (second int))))
                  (push int (cdr (member rvp rvplist :test #'eq))))))))
        (setq prev rvp))))

(defun r-policy (mdp r &aux U)
  (set-rewards mdp r)
  (setq U (value-iteration mdp
                           (copy-hash-table (mdp-rewards mdp) #'identity)
                           :epsilon 0.0000000001))
  (list (gethash '(1 1) U) (optimal-policy U (mdp-model mdp) (mdp-rewards mdp))))

(defun set-rewards (mdp r &aux (rewards (mdp-rewards mdp))
                               (terminals (mdp-terminal-states mdp)))
  (maphash #'(lambda (state reward)
               (unless (member state terminals :test #'equal)
                   (setf (gethash state rewards) r)))
           rewards))
```

```
(defun policy-equal (p1 p2 mdp &aux (match t)
                                    (terminals (mdp-terminal-states mdp)))
  (maphash #'(lambda (state action)
               (unless (member state terminals :test #'equal)
                 (unless (eq (caar (gethash state p1)) (caar (gethash state p2)))
                   (setq match nil))))
           p1)
  match)

(defun policy-vertex (rvp1 rvp2 rvp3 rvp4)
  (let  ((a (make-xy :x (first rvp1) :y (second rvp1)))
         (b (make-xy :x (first rvp2) :y (second rvp2)))
         (c (make-xy :x (first rvp3) :y (second rvp3)))
         (d (make-xy :x (first rvp4) :y (second rvp4))))
    (intersection-point (make-line :xy1 a :xy2 b)
                        (make-line :xy1 c :xy2 d))))

(defun intersection-point (l1 l2)
;;; l1 is line ab; l2 is line cd
;;; assume the lines cross at alpha a + (1-alpha) b,
;;;     also known as beta c + (1-beta) d
;;; returns the intersection point unless they're parallel
  (let* ((a (line-xy1 l1))
         (b (line-xy2 l1))
         (c (line-xy1 l2))
         (d (line-xy2 l2))
         (xa (xy-x a)) (ya (xy-y a))
         (xb (xy-x b)) (yb (xy-y b))
         (xc (xy-x c)) (yc (xy-y c))
         (xd (xy-x d)) (yd (xy-y d))
         (q (- (* (- xa xb) (- yc yd))
               (* (- ya yb) (- xc xd)))))
    (unless (zerop q)
      (let ((alpha (/ (- (* (- xd xb) (- yc yd))
                         (* (- yd yb) (- xc xd)))
                      q)))
        (make-xy :x (float (+ (* alpha xa) (* (- 1 alpha) xb)))
                 :y (float (+ (* alpha ya) (* (- 1 alpha) yb))))))))
```

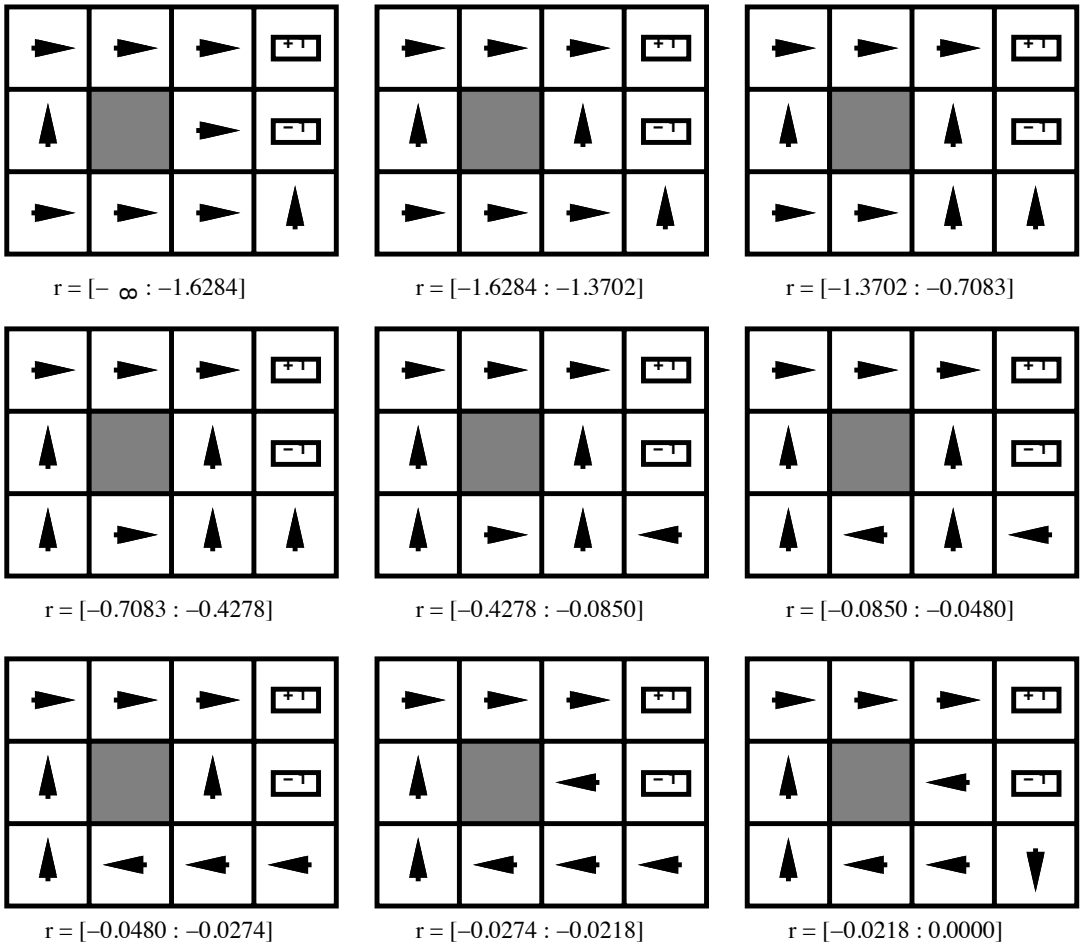

Figure S16.1 Optimal policies for different values of the cost of a step in the 4×3 environment, and the boundaries of the regions with constant optimal policy.

16.2 Algorithms for MDPs

Exercise 16.VICT

Equation (16.11) on page 565 states that the Bellman operator is a contraction.

a. Show that, for any functions f and g,

$$|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)| \, .$$

b. Write out an expression for $|(B\,U_i - B\,U_i')(s)|$ and then apply the result from (a) to complete the proof that the Bellman operator is a contraction.

a. To find the proof, it may help first to draw a picture of two arbitrary functions f and g and mark the maxima; then it is easy to find a point where the difference between the functions is bigger than the difference between the maxima. Assume, w.l.o.g., that $\max_a f(a) \geq \max_a g(a)$, and let f have its maximum value at a^*. Then we have

$$\begin{aligned}
|\max_a f(a) - \max_a g(a)| &= \max_a f(a) - \max_a g(a) \qquad \text{(by assumption)} \\
&= f(a^*) - \max_a g(a) \\
&\leq f(a^*) - g(a^*) \\
&\leq \max_a |f(a) - g(a)| \qquad \text{(by definition of max)}
\end{aligned}$$

b. From the definition of the B operator (Equation (16.10)) we have

$$\begin{aligned}
|(B\,U_i - B\,U'_i)(s)| &= |R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s') \\
&\quad - R(s) - \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U'_i(s')| \\
&= \gamma |\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s') - \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U'_i(s')| \\
&\leq \gamma \max_{a \in A(s)} |\sum_{s'} P(s' \mid s, a) U_i(s') - \sum_{s'} P(s' \mid s, a) U'_i(s')| \\
&= \gamma |\sum_{s'} P(s' \mid s, a^*(s)) U_i(s') - \sum_{s'} P(s' \mid s, a^*(s)) U'_i(s')| \\
&= \gamma |\sum_{s'} P(s' \mid s, a^*(s)) (U_i(s') - U'_i(s')|
\end{aligned}$$

Inserting this into the expression for the max norm, we have

$$\begin{aligned}
||B\,U_i - B\,U'_i|| &= \max_s |(B\,U_i - B\,U'_i)(s)| \\
&\leq \gamma \max_s |\sum_{s'} P(s' \mid s, a^*(s)) (U_i(s') - U'_i(s'))| \\
&\leq \gamma \max_s |U_i(s) - U'_i(s)| = \gamma ||U_i - U'_i||
\end{aligned}$$

Exercise 16.MDPZ

This exercise considers two-player MDPs that correspond to zero-sum, turn-taking games like those in Chapter 6. Let the players be A and B, and let $R(s)$ be the reward for player A in state s. (The reward for B is always equal and opposite.)

a. Let $U_A(s)$ be the utility of state s when it is A's turn to move in s, and let $U_B(s)$ be the utility of state s when it is B's turn to move in s. All rewards and utilities are calculated from A's point of view (just as in a minimax game tree). Write down Bellman equations defining $U_A(s)$ and $U_B(s)$.

b. Explain how to do two-player value iteration with these equations, and define a suitable termination criterion.

c. Consider the game described in Figure **??** on page **??**. Draw the state space (rather than

the game tree), showing the moves by A as solid lines and moves by B as dashed lines. Mark each state with $R(s)$. You will find it helpful to arrange the states (s_A, s_B) on a two-dimensional grid, using s_A and s_B as "coordinates."

d. Now apply two-player value iteration to solve this game, and derive the optimal policy.

a. For U_A we have

$$U_A(s) = R(s) + \max_a \sum_{s'} P(s'|a,s)U_B(s')$$

and for U_B we have

$$U_B(s) = R(s) + \min_a \sum_{s'} P(s'|a,s)U_A(s') \ .$$

b. To do value iteration, we simply turn each equation from part (a) into a Bellman update and apply them in alternation, applying each to all states simultaneously. The process terminates when the utility vector for one player is the same as the previous utility vector *for the same player* (i.e., two steps earlier). (Note that typically U_A and U_B are not the same in equilibrium.)

c. The state space is shown in Figure S16.2.

d. We mark the terminal state values in bold and initialize other values to 0. Value iteration proceeds as follows:

	(1,4)	(2,4)	(3,4)	(1,3)	(2,3)	(4,3)	(1,2)	(3,2)	(4,2)	(2,1)	(3,1)
U_A	0	0	0	0	0	**+1**	0	0	**+1**	**–1**	**–1**
U_B	0	0	0	0	–1	**+1**	0	–1	**+1**	**–1**	**–1**
U_A	0	0	0	–1	+1	**+1**	–1	+1	**+1**	**–1**	**–1**
U_B	–1	+1	+1	–1	–1	**+1**	–1	–1	**+1**	**–1**	**–1**
U_A	+1	+1	+1	–1	+1	**+1**	–1	+1	**+1**	**–1**	**–1**
U_B	–1	+1	+1	–1	–1	**+1**	–1	–1	**+1**	**–1**	**–1**

and the optimal policy for each player is as follows:

	(1,4)	(2,4)	(3,4)	(1,3)	(2,3)	(4,3)	(1,2)	(3,2)	(4,2)	(2,1)	(3,1)
π_A^*	(2,4)	(3,4)	(2,4)	(2,3)	(4,3)		(3,2)	(4,2)			
π_B^*	(1,3)	(2,3)	(3,2)	(1,2)	(2,1)		(1,3)	(3,1)			

Exercise 16.TMDP

Consider the 3×3 world shown in Figure **??**(a). The transition model is the same as in the 4×3 Figure 16.1: 80% of the time the agent goes in the direction it selects; the rest of the time it moves at right angles to the intended direction.

Implement value iteration for this world for each value of r below. Use discounted rewards with a discount factor of 0.99. Show the policy obtained in each case. Explain intu-

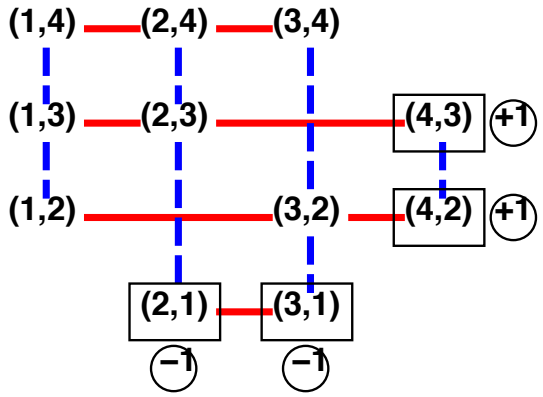

Figure S16.2 State-space graph for the game in Figure **??**.

itively why the value of r leads to each policy.

a. $r = 100$
b. $r = -3$
c. $r = 0$
d. $r = +3$

a. $r = 100$.

u	l	.
u	l	d
u	l	l

See the comments for part d. This should have been $r = -100$ to illustrate an alternative behavior:

r	r	.
d	r	u
r	r	u

Here, the agent tries to reach the goal quickly, subject to attempting to avoid the square $(1, 3)$ as much as possible. Note that the agent will choose to move Down in square $(1, 2)$ in order to actively avoid the possibility of "accidentally" moving into the square $(1, 3)$ if it tried to move Right instead, since the penalty for moving into square (1, 3) is so great.

b. $r = -3$.

r	r	.
r	r	u
r	r	u

Here, the agent again tries to reach the goal as fast as possible while attempting to avoid the square $(1, 3)$, but the penalty for square $(1, 3)$ is not so great that the agent will try to actively avoid it at all costs. Thus, the agent will choose to move Right in

square $(1, 2)$ in order to try to get closer to the goal even if it occasionally will result in a transition to square $(1, 3)$.

c. $r = 0$.

r	r	.
u	u	u
u	u	u

Here, the agent again tries to reach the goal as fast as possible, but will try to do so via a path that includes square $(1, 3)$ if possible. This results from the fact that square $(1, 3)$ does not incur the reward of -1 in all other non-goal states, so it reaching the goal via a path through that square can potentially have slightly greater reward than another path of equal length that does not pass through $(1, 3)$.

d. $r = 3$.

u	l	.
u	l	d
u	l	l

Exercise 16.MDPO

Consider the 101×3 world shown in Figure **??**(b). In the start state the agent has a choice of two deterministic actions, *Up* or *Down*, but in the other states the agent has one deterministic action, *Right*. Assuming a discounted reward function, for what values of the discount γ should the agent choose *Up* and for which *Down*? Compute the utility of each action as a function of γ. (Note that this simple example actually reflects many real-world situations in which one must weigh the value of an immediate action versus the potential continual long-term consequences, such as choosing to dump pollutants into a lake.)

The utility of Up is

$$50\gamma - \sum_{t=2}^{101} \gamma^t = 50\gamma - \gamma^2 \frac{1 - \gamma^{100}}{1 - \gamma},$$

while the utility of Down is

$$-50\gamma + \sum_{t=2}^{101} \gamma^t = -50\gamma + \gamma^2 \frac{1 - \gamma^{100}}{1 - \gamma}.$$

Solving numerically we find the indifference point to be $\gamma \approx 0.9844$: larger than this and we want to go Down to avoid the expensive long-term consequences, smaller than this and we want to go Up to get the immediate benefit.

Exercise 16.MDPU

Consider an undiscounted MDP having three states, (1, 2, 3), with rewards -1, -2, 0, respectively. State 3 is a terminal state. In states 1 and 2 there are two possible actions: a and b. The transition model is as follows:

- In state 1, action a moves the agent to state 2 with probability 0.8 and makes the agent stay put with probability 0.2.
- In state 2, action a moves the agent to state 1 with probability 0.8 and makes the agent stay put with probability 0.2.
- In either state 1 or state 2, action b moves the agent to state 3 with probability 0.1 and makes the agent stay put with probability 0.9.

Answer the following questions:

a. What can be determined *qualitatively* about the optimal policy in states 1 and 2?

b. Apply policy iteration, showing each step in full, to determine the optimal policy and the values of states 1 and 2. Assume that the initial policy has action b in both states.

c. What happens to policy iteration if the initial policy has action a in both states? Does discounting help? Does the optimal policy depend on the discount factor?

a. Intuitively, the agent wants to get to state 3 as soon as possible, because it will pay a cost for each time step it spends in states 1 and 2. However, the only action that reaches state 3 (action b) succeeds with low probability, so the agent should minimize the cost it incurs while trying to reach the terminal state. This suggests that the agent should definitely try action b in state 1; in state 2, it might be better to try action a to get to state 1 (which is the better place to wait for admission to state 3), rather than aiming directly for state 3. The decision in state 2 involves a numerical tradeoff.

b. The application of policy iteration proceeds in alternating steps of value determination and policy update.

- *Initialization*: $U \quad \langle -1, -2, 0 \rangle$, $P \quad \langle b, b \rangle$.
- *Value determination*:

$$\begin{aligned} u_1 &= -1 + 0.1u_3 + 0.9u_1 \\ u_2 &= -2 + 0.1u_3 + 0.9u_2 \\ u_3 &= 0 \end{aligned}$$

That is, $u_1 = -10$ and $u_2 = -20$.

Policy update: In state 1,

$$\sum_j T(1, a, j)u_j = 0.8 \times -20 + 0.2 \times -10 = -18$$

while

$$\sum_j T(1, b, j)u_j = 0.1 \times 0 \times 0.9 \times -10 = -9$$

so action b is still preferred for state 1.

In state 2,

$$\sum_j T(1, a, j)u_j = 0.8 \times -10 + 0.2 \times -20 = -12$$

while

$$\sum_j T(1, b, j)u_j = 0.1 \times 0 \times 0.9 \times -20 = -18$$

so action a is preferred for state 1. We set $unchanged?$ $\leftarrow$ false and proceed.

- *Value determination*:

$$\begin{aligned} u_1 &= -1 + 0.1u_3 + 0.9u_1 \\ u_2 &= -2 + 0.8u_1 + 0.2u_2 \\ u_3 &= 0 \end{aligned}$$

Once more $u_1 = -10$; now, $u_2 = -15$. *Policy update*: In state 1,

$$\sum_j T(1, a, j)u_j = 0.8 \times -15 + 0.2 \times -10 = -14$$

while

$$\sum_j T(1, b, j)u_j = 0.1 \times 0 \times 0.9 \times -10 = -9$$

so action b is still preferred for state 1.

In state 2,

$$\sum_j T(1, a, j)u_j = 0.8 \times -10 + 0.2 \times -15 = -11$$

while

$$\sum_j T(1, b, j)u_j = 0.1 \times 0 \times 0.9 \times -15 = -13.5$$

so action a is still preferred for state 1. $unchanged?$ remains true, and we terminate.

Note that the resulting policy matches our intuition: when in state 2, try to move to state 1, and when in state 1, try to move to state 3.

c. An initial policy with action a in both states leads to an unsolvable problem. The initial value determination problem has the form

$$\begin{aligned} u_1 &= -1 + 0.2u_1 + 0.8u_2 \\ u_2 &= -2 + 0.8u_1 + 0.2u_2 \\ u_3 &= 0 \end{aligned}$$

and the first two equations are inconsistent. If we were to try to solve them iteratively,

we would find the values tending to $-\infty$.

Discounting leads to well-defined solutions by bounding the penalty (expected discounted cost) an agent can incur at either state. However, the choice of discount factor will affect the policy that results. For γ small, the cost incurred in the distant future plays a negligible role in the value computation, because γ^n is near 0. As a result, an agent could choose action b in state 2 because the discounted short-term cost of remaining in the non-terminal states (states 1 and 2) outweighs the discounted long-term cost of action b failing repeatedly and leaving the agent in state 2. An additional exercise could ask the student to determine the value of γ at which the agent is indifferent between the two choices.

Exercise 16.MDPF

Consider the 4×3 world shown in Figure 16.1.

a. Implement an environment simulator for this environment, such that the specific geography of the environment is easily altered. Some code for doing this is already in the online code repository.

b. Create an agent that uses policy iteration, and measure its performance in the environment simulator from various starting states. Perform several experiments from each starting state, and compare the average total reward received per run with the utility of the state, as determined by your algorithm.

c. Experiment with increasing the size of the environment. How does the run time for policy iteration vary with the size of the environment?

The framework for this problem is in `uncertainty/domains/4x3-mdp.lisp`. There is still some synthesis for the student to do for answer b. For c. some experimental design is necessary.

Exercise 16.POLL

How can the policy evaluation algorithm be used to calculate the expected loss experienced by an agent using a given set of utility estimates U and an estimated model P, compared with an agent using correct values?

The policy evaluation algorithm calculates $U^\pi(s)$ for a given policy π. The policy for an agent that thinks $U$is the true utility and P is the true model would be based on Equation (16.4):

$$\pi(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s') \ .$$

Given this policy, the policy loss compared to the true optimal policy, starting in state s, is just $U^{\pi^*}(s) - U^\pi(s)$.

Exercise 16.UCTT

In this exercise we explore the application of UCT to Tetris.

a. Create an implementation the Tetris MDP as described in Figure 16.5. Each action simply places the current piece in any reachable location and orientation.
b. Estimate the reward for a purely random policy by running rollouts.
c. Implement a version of UCT (Section 5.4) suitable for MDPs.
d. Apply your algorithm to Tetris and measure its performance as a function of the number of rollouts per move, assuming a purely random policy for rollouts and a value C = $\sqrt{2}$ for the paramater that controls the exploration/exploitation tradeoff.
e. Come up with a better rollout policy and measure its performance as a function of the number of rollouts and CPU time.

The implementation of this exercise left up to interested readers.

Exercise 16.VAIT

Value iteration:

(i) Is a model-free method for finding optimal policies.
(ii) Is sensitive to local optima.
(iii) Is tedious to do by hand.
(iv) Is guaranteed to converge when the discount factor satisfies $0 < \gamma < 1$.

(iii) and (iv). Value iteration requires a model (an specified MDP), and is not sensitive to getting stuck in local optima.

Exercise 16.MDPT

Please indicate whether the following statements are true of false

a. If the only difference between two MDPs is the value of the discount factor then they must have the same optimal policy.
b. When using features to represent the Q-function (rather than having a tabular representation) it is possible that Q-learning does not find the optimal Q-function Q^*.
c. For an infinite horizon MDP with a finite number of states and actions and with a discount factor γ, with $0 < \gamma < 1$, value iteration is guaranteed to converge.
d. When getting to act only for a finite number of steps in an MDP, the optimal policy is stationary. (A stationary policy is a policy that takes the same action in a given state, independent of at what time the agent is in that state.)

a. False. A counterexample suffices to show the statement is false. Consider an MDP with two sink states. Transitioning into sink state A gives a reward of 1, transitioning into

sink state B gives a reward of 10. All other transitions have zero rewards. Let A be one step North from the start state. Let B be two steps South from the start state. Assume actions always succeed. Then if the discount factor $\gamma < 0.1$ the optimal policy takes the agent one step North from the start state into A, if the discount factor $\gamma > 0.1$ the optimal policy takes the agent two steps South from the start state into B.

b. True. Whenever the optimal Q-function, Q^*, cannot be represented as a weighted combination of features, then the feature-based representation would not even have the expressiveness to find the optimal Q-function, Q^*.

c. True. V_k and V_{k+1} are at most $\gamma^k \max |R|$ different so value iteration converges for any $0 \leq \gamma < 1$. See lecture slides for more details.

d. False. A simple counter-example suffices. Assume that the agent can reach a reward of \$10 within 2 time steps or a reward of \$1 withing one time step. If the horizon is less than two then the agent will aim for the latter. See lecture slides for more details.

Exercise 16.CUGD

Consider an $(N+1) \times (N+1) \times (N+1)$ cubic gridworld. Luckily, all the cells are empty – there are no walls within the cube. For each cell, there is an action for each adjacent facing open cell (no corner movement), as well as an action *stay*. The actions all move into the corresponding cell with probability p but stay with probability $1-p$. *Stay* always stays. The reward is always zero except when you enter the goal cell at (N, N, N), in which case it is 1 and the game then ends. The discount is $0 < \gamma < 1$.

a. How many iterations k of value iteration will there be before $V_k(0,0,0)$ becomes non-zero? If this will never happen, write *never*.

b. If and when $V_k(0,0,0)$ first becomes non-zero, what will it become? If this will never happen, write *never*.

c. What is $V^*(0,0,0)$? If it is undefined, write *undefined*.

a. $3N$. At V_0 the value of the goal is correct. At V_1 all cells next to the goal are non-zero, at V_2 all cells next to those are nonzero, and so on.

b. $(\gamma p)^{3N}/\gamma$. The value update of a cell c in this problem is $V'(c) = p(r_{c'} + \gamma V(c')) + (1-p)V(c)$. The first time the value of a state becomes non-zero, the value is $V'(c) = p(r_{c'} + \gamma V(c'))$.

$$V'(g) = p(1 + \gamma V(goal) = p \qquad \text{for a cell } g \text{ adjacent to the goal.}$$
$$V'(c) = p\gamma V(c') \qquad \text{for other cells since the reward is } 0.$$

Carrying out the value recursion, the goal reward $+1$ is multiplied by p for the step to the goal and $p\gamma$ for each further step. The number of steps from the start to the goal is $3N$, the Manhattan distance in the cube. The first non-zero $V(0,0,0)$ is $(\gamma p)^{3N}/\gamma$ since every step multiplies in $p\gamma$ except the last step to the goal, which multiplies in p. Equivalently, the first non-zero value is $p(\gamma p)^{3N-1}$ with $(\gamma p)^{3N-1}$ for all the steps from the start to a cell adjacent to the goal and p for the transition to the goal.

c. $\frac{1}{\gamma}\left(\frac{\gamma p}{1-\gamma+\gamma p}\right)^{3N}$

To see why, let $V^*(d)$ be the value function of states whose Manhattan distance from the goal is d. By symmetry, all states with the same Manahttan distance from the goal will have the same value. Write the Bellman equations:

$$V^*(d) = \gamma(1-p)V^*(d) + \gamma p V^*(d-1) \qquad \text{for all } d > 1$$
$$V^*(1) = p + \gamma(1-p)V^*(1) + \gamma p V^*(0)$$
$$V^*(0) = \gamma V^*(0)$$

and solve starting with $V^*(0) = 0$ to get the answer.

Exercise 16.VALG

Suppose we run value iteration in an MDP with only non-negative rewards (that is, $R(s,a,s') \geq 0$ for any (s,a,s')). Let the values on the kth iteration be $V_k(s)$ and the optimal values be $V^*(s)$. Initially, the values are 0 (that is, $V_0(s) = 0$ for any s).

a. Mark *all* of the options that are *guaranteed* to be true.

(i) For any s, a, s', $V_1(s) = R(s,a,s')$
(ii) For any s, a, s', $V_1(s) \leq R(s,a,s')$
(iii) For any s, a, s', $V_1(s) \geq R(s,a,s')$
(iv) None of the above are guaranteed to be true.

b. Mark *all* of the options that are *guaranteed* to be true.

(i) For any k, s, $V_k(s) = V^*(s)$
(ii) For any k, s, $V_k(s) \leq V^*(s)$
(iii) For any k, s, $V_k(s) \geq V^*(s)$
(iv) None of the above are guaranteed to be true.

a. (iv) only.

$V_1(s) = \max_a \sum_{s'} T(s,a,s')R(s,a,s')$ (using the Bellman equation and setting $V_0(s') = 0$).

Now consider an MDP where the best action in state X is clockwise, which goes to state Y with a reward of 6 with probability 0.5 and goes to state Z a reward of 4 with probability 0.5. Then $V_1(X) = 0.5(6) + 0.5(4) = 5$.

Notice that setting (s,a,s') = $(X, clockwise, Z)$ gives a counterexample for the second option and (s,a,s') = $(X, clockwise, Y)$ gives a counterexample for the third option.

b. (ii) only.

Intuition: Values can never decrease in an iteration. In the first iteration, since all rewards are positive, the values increase. In any other iteration, the components that contribute to $V_{k+1}(s)$ are $R(s,a,s')$ and $V(s')$. $R(s,a,s')$ is the same across all iterations, and $V(s')$ increased in the previous iteration, so we expect $V_{k+1}(s)$ to increase as well.

More formally, we can prove $V_k(s) \leq V_{k+1}(s)$ by induction.
Base Case: $V_1(s) = \max_a \sum_{s'} T(s,a,s')R(s,a,s')$.
Since $R(s,a,s') \geq 0, T(s,a,s') \geq 0$, we have $V_1(s) \geq 0$, and so $V_0(s) \leq V_1(s)$.
Induction: $V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$
$\geq \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_{k-1}(s')]$ (using $V_k(s') \geq V_{k-1}(s')$ from the inductive hypothesis)
$= V_k(s)$.
This immediately leads to $V_k(s) \leq V^*(s)$ (since we can think of $V^*(s)$ as $V_\infty(s)$).

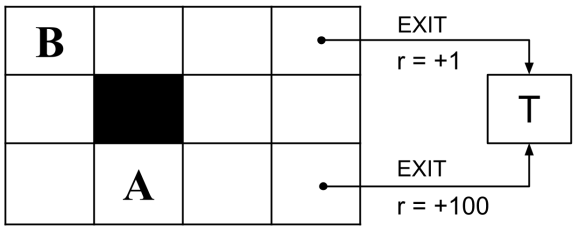

Figure S16.3 MDP for Exercise 16.GMDP.

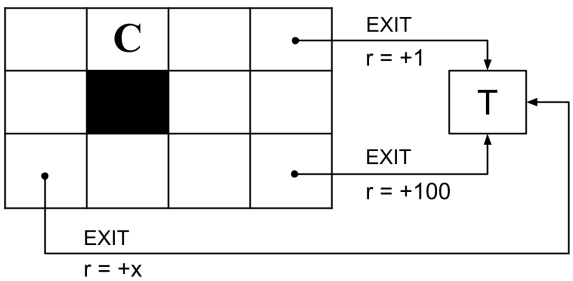

Figure S16.4 MDP for Exercise 16.GMDP.

Exercise 16.GMDP

a. Please indicate if the following statements are true or false.

(i) Let A be the set of all actions and S the set of states for some MDP. Assuming that $|A| \ll |S|$, one iteration of value iteration is generally faster than one iteration of policy iteration that solves a linear system during policy evaluation.

(ii) For any MDP, changing the discount factor does not affect the optimal policy for the MDP.

The following problem will take place in various instances of a grid world MDP. Shaded cells represent walls. In all states, the agent has available actions $\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$.

Performing an action that would transition to an invalid state (outside the grid or into a wall) results in the agent remaining in its original state. In states with an arrow coming out, the agent has an *additional* action $EXIT$. In the event that the $EXIT$ action is taken, the agent receives the labeled reward and ends the game in the terminal state T. Unless otherwise stated, all other transitions receive no reward, and all transitions are deterministic.

For all parts of the problem, assume that value iteration begins with all states initialized to zero, i.e., $V_0(s) = 0 \ \forall s$. **Let the discount factor be $\gamma = \frac{1}{2}$ for all following parts**.

b. Suppose that we are performing value iteration on the grid world MDP in Figure S16.3.

(i) What's the optimal values for A and B?

(ii) After how many iterations k will we have $V_k(s) = V^*(s)$ for all states s? If it never occurs, write "never".

(iii) Suppose that we wanted to re-design the reward function. For which of the following new reward functions would the optimal policy **remain unchanged**? Let $R(s, a, s')$ be the original reward function.

(I) $R_1(s, a, s') = 10R(s, a, s')$
(II) $R_2(s, a, s') = 1 + R(s, a, s')$
(III) $R_3(s, a, s') = R(s, a, s')^2$
(IV) $R_4(s, a, s') = -1$
(V) None

c. For the problem in Figure S16.4, we add a new state in which we can take the $EXIT$ action with a reward of $+x$.

(i) For what values of x is it *guaranteed* that our optimal policy π^* has $\pi^*(C) = \leftarrow$?

(ii) For what values of x does value iteration take the **minimum** number of iterations k to converge to V^* for all states? Write ∞ and $-\infty$ if there is no upper or lower bound, respectively.

(iii) Fill the box with value k, the **minimum** number of iterations until V_k has converged to V^* for all states.

a. Fill out the following True/False questions.

(i) True. One iteration of value iteration is $O(|S|^2|A|)$, whereas one iteration of policy iteration is $O(|S|^3)$, so value iteration is generally faster when $|A| \ll |S|$

(ii) False. Consider an infinite horizon setting where we have 2 states A, B, where we can alternate between A and B forever, gaining a reward of 1 each transition, or exit from B with a reward of 100. In the case that $\gamma = 1$, the optimal policy is to forever oscillate between A and B. If $\gamma = \frac{1}{2}$, then it is optimal to exit.

b. (i) $V^*(A) = 25, V^*(B) = \frac{25}{8}$

(ii) 6

(iii) (I), (II), (III)

R_1: Scaling the reward function does not affect the optimal policy, as it scales all Q-values by 10, which retains ordering

R_2: Since reward is discounted, the agent would get more reward exiting then infinitely cycling between states

R_3: The only positive reward remains to be from exiting state $+100$ and $+1$, so the optimal policy doesn't change

R_4: With negative reward at every step, the agent would want to exit as soon as possible, which means the agent would not always exit at the bottom-right square.

c. (i) $50 < x < \infty$.

We go left if $Q(C, \leftarrow) > Q(C, \rightarrow)$. $Q(C, \leftarrow) = \frac{1}{8}x$, and $Q(C, \rightarrow) = \frac{100}{16}$. Solving for x, we get $x > 50$.

(ii) $50 \leq x \leq 200$. The two states that will take the longest for value iteration to become non-zero from either $+x$ or $+100$, are states C, and D, where D is defined as the state to the right of C. C will become nonzero at iteration 4 from $+x$, and D will become nonzero at iteration 4 from $+100$. We must bound x so that the optimal policy at D does not choose to go to $+x$, or else value iteration will take 5 iterations. Similar reasoning for D and $+x$. Then our inequalities are $\frac{1}{8}x \geq \frac{100}{16}$ and $\frac{1}{16}x \leq \frac{100}{8}$. Simplifying, we get the following bound on x: $50 \leq x \leq 200$

(iii) 4. See the explanation for the part above

Exercise 16.LQRX

Consider the following deterministic MDP with 1-dimensional continuous states and actions and a finite task horizon:

State Space $\mathcal{S}$: $\mathbb{R}$
Action Space $\mathcal{A}$: $\mathbb{R}$
Reward Function: $R(s, a, s') = -qs^2 - ra^2$ where $r > 0$ and $q \geq 0$
Deterministic Dynamics/Transition Function: $s' = cs + da$ (i.e., the next state s' is a deterministic function of the action a and current state s)
Task Horizon: $T \in \mathbb{N}$
Discount Factor: $\gamma = 1$ (no discount factor)

Hence, we would like to maximize a quadratic reward function that rewards small actions and staying close to the origin. In this problem, we will design an optimal agent π_t^* and also solve for the optimal agent's value function V_t^* for all timesteps.

By induction, we will show that V_t^* is quadratic. Observe that the base case $t = 0$ trivially holds because $V_0^*(s) = 0$ For all parts below, assume that $V_t^*(s) = -p_t s^2$ (Inductive Hypothesis).

a. (i) Write the equation for $V_{t+1}^*(s)$ as a function of s, q, r, a, c, d, and p_t. If your expression contains max, you do not need to simplify the max.

(ii) Now, solve for $\pi_{t+1}^*(s)$. Recall that you can find local maxima of functions by computing the first derivative and setting it to 0.

b. Assume $\pi_{t+1}^* = k_{t+1}s$ for some $k_{t+1} \in \mathbb{R}$. Solve for p_{t+1} in $V_{t+1}^*(s) = -p_{t+1}s^2$.

a. (i) $V^*_{t+1}(s) = \max_{a\in\mathbb{R}} -qs^2 - ra^2 - p_t(cs+da)^2$

$$\begin{aligned} V^*_{t+1}(s) &= \max_{a\in\mathbb{R}} R(s,a,s') + V^*_t(s') \\ &= \max_{a\in\mathbb{R}} -qs^2 - ra^2 - p_t(s')^2 \\ &= \max_{a\in\mathbb{R}} -qs^2 - ra^2 - p_t(cs+da)^2 \end{aligned}$$

(ii) $\pi^*_{t+1}(s) = -\frac{cdp_t}{r+p_td^2}s$

We want to solve for the a that maximizes the value of $V^*_{t+1}(s)$. This is equivalent to maximizing $-ra^2 - p_{t+1}(cs+da)^2$. Differentiate this function, set it to 0 and solve for a.

$$\frac{d}{da}[-ra^2 - p_t(cs+da)^2] = -2ra - 2(cs+da)p_td = 0$$

Solving for a, we find that:

$$\pi^*_{t+1}(s) = -\frac{cdp_t}{r+p_td^2}s = k_{t+1}s$$

Observe that the optimal policy is a linear function of the state.

b. $p_{t+1} = q + rk^2_{t+1} + p_t(c+dk_{t+1})^2$

$$\begin{aligned} V^*_{t+1}(s) &= R(s,\pi^*_{t+1}(s),s') + V^*_t(s') \\ &= -qs^2 - r(k_{t+1}s)^2 - p_ts'^2 \\ &= -qs^2 - r(k_{t+1}s)^2 - p_t(cs+d(k_{t+1}s))^2 \\ &= -(q + rk^2_{t+1} + p_t(c+dk_{t+1})^2)s^2 \\ &= -p_{t+1}s^2 \end{aligned}$$

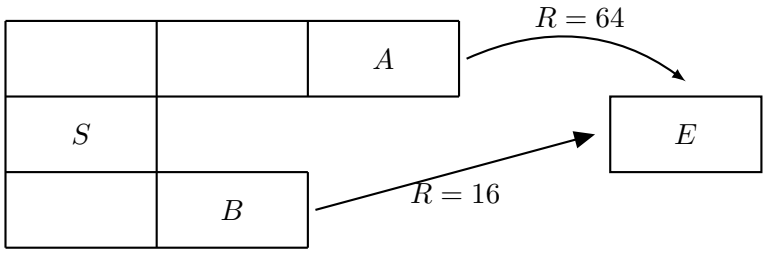

Figure S16.5 MDP for Exercise 16.DLR.

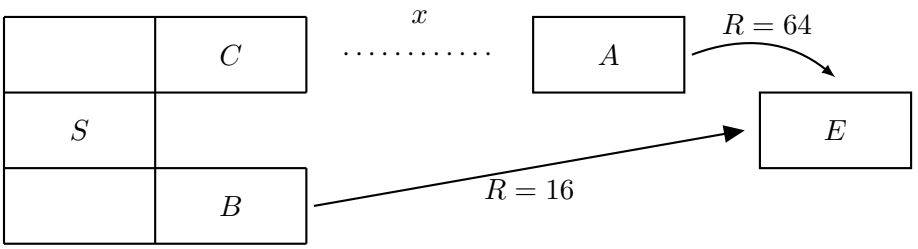

Figure S16.6 Second MDP for Exercise 16.DLR.

Exercise 16.DLR

Pacman finds himself inside the grid world MDP depicted in Figure S16.5. Each rectangle represents a possible state. At each state, Pacman can take actions up, down, left or right. If an action moves him into a wall, he will stay in the same state. At states A and B, Pacman can take the exit action to receive the indicated reward and enter the terminal state, E. Note $R(s, a, s') = 0$ otherwise. Once in the terminal state the game is over and no actions can be taken. Let the discount factor $\gamma = \frac{1}{2}$ for this problem, unless otherwise specified.

a. (i) What is the optimal action at state S?

(ii) How many iterations k will it take before $V_k(S) = V^*(S)$?

(iii) What are all the values that $V_k(S)$ will take on during the entire process of value iteration.

b. Now Ghost wants to mess with Pacman. She wants to change some of the rules of this grid world so that Pacman does not exit from state A. **All subquestions are independent** of each other so consider each change on its own.

(i) First, Ghost wants to change the discount factor. Write a bound on the discount factor that guarantees Pacman exits from B instead of A. Any valid value of $\gamma \in (0, 1]$ that satisfies your inequality should cause Pacman to exit from B instead of A.

(ii) Next, Ghost thinks she can change the reward function to accomplish this. Write a bound on the reward from A, $R(A, exit, E)$, that guarantees Pacman exits from B instead of A? Any value of $R(A, exit, E)$ that satisfies your inequality should cause Pacman to exit from B instead of A.

(iii) Ghost came up with a bunch of reward functions, $R'(s, a, s')$. Select the new reward functions that cause Pacman not to exit from state A. Note $R(s, a, s')$ is the original reward function from the problem description so the reward from every state is going to be affected.

(A) $R'(s, a, s') = 1 + R(s, a, s')$

(B) $R'(s, a, s') = 100 + R(s, a, s')$

(C) $R'(s, a, s') = -1 + R(s, a, s')$

(D) $R'(s, a, s') = -100 + R(s, a, s')$

(E) $R'(s, a, s') = -R(s, a, s')$

(F) $R'(s, a, s') = 2R(s, a, s')$

(iv) Ghost realizes she can stop Pacman from exiting from A by adding a certain amount of grids, x, in between C and A as depicted in Figure S16.6. Give a lower bound for x that **guarantees** Pacman does not exit from A.

c. Another way that Ghost can mess with Pacman is by choosing parameters such that Pacman's value iteration never converges. Select which reward function and discount factor pairs cause value iteration to never converge.

(A) $R'(s, a, s') = 100 + R(s, a, s')$, $\gamma = 0.9$
(B) $R'(s, a, s') = -100 + R(s, a, s')$, $\gamma = 0.9$
(C) $R'(s, a, s') = -1 + R(s, a, s')$, $\gamma = 1.0$
(D) $R'(s, a, s') = 1 + R(s, a, s')$, $\gamma = 1.0$

a. (i) Up.
Pacman can get $64 * .5^3 = 8$ from exiting from A and he can get $16 * .5^2 = 4$ from exiting from B so he should move up to go towards A.

(ii) $k = 4$
It takes one iteration for A to learn its value. One more iteration for the node to the left of A. Another iteration for the node to the left of that (and above S). And one last iteration for S to learn its value of 8 from the node above it. That is 4 iterations in total.

(iii) 0, 4, 8
All values start at 0. Then $V_k(S)$ will equal 4 when it learns it can get 4 from exiting from B after 3 iterations (since B is closer than A). Finally it will update to 8 when it learns it can get 8 from exiting from A.

b. (i) $\gamma < \frac{1}{4}$
Since A is farther away from S than B, the rewards from A have the discount factor applied more than the rewards from B. Thus if we can find a discount factor where the rewards are equal any discount factor less than (¡) that will cause Pacman to exit from B. If we choose a discount factor of .25 then the node to the left of A will have a value of $64 * .25 = 16$ which is the same as B (and both of those nodes are the same distance from S). So .25 causes the rewards to be the same and is our answer.

(ii) $R(A, exit, E) < 32$
Similarly to part bi) we just need to choose a reward value that causes the rewards from A to equal the rewards from B. If we choose 32 the total rewards Pacman can get from A is $32 * .5^3 = 4$ which is the same as the rewards from B. Thus is the rewards from A are any amount less than (¡) that Pacman will exit from B.

(iii) (B), (D), (E)
For this problem we need to analyze which reward functions will cause Pacman to either oscillate infinitely or exit from B. The rewards that add 1 and -1 do not change the rewards enough to accomplish this. Also scaling the rewards by 2

changes nothing comparatively so it also doesn't work. Adding 100 to every reward causes Pacman to change states infinitely instead of exiting because he can get $\frac{100}{1-.5} = 200$ from going back and forth. Thus he won't exit from A in this case. The reward function that adds -100 to every reward and the function that is just the negative of the old reward both cause Pacman to want to exit the game as fast as possible which is through B.

(iv) $x \geq 2$

By adding grids we can apply the discount factor to the rewards from A as many times as we want. If we add one grid the reward from C $64 * .5^2 = 16$ which is the same as from B (and C and B are the same distance from S). Since the problem said guaranteed, we need to add one more grid to make sure the reward from A is strictly less than the reward from A so the answer is 2.

c. D.

Note any discount factor less than 1 will always converge by a proof from lecture. The function that added -1 to every reward will also converge because Pacman will want to exit the game instead of accumulating an infinite negative reward. The function that added 1 to every reward will never converge because Pacman can accumulate infinite rewards by going back and forth.

16.3 Bandit Problems

Exercise 16.KATV

Figure 16.13 shows two MDPs: one, M, represents a two-armed bandit where one has the choice to continue with the first arm or to switch permanently to a second arm with fixed reward λ; the other, a **restart MDP** M^s, gives one a choice to continue with the first arm or restart the sequence. The figure illustrates the construction of M^s for the case where M has a deterministic reward sequence and just two arms (including the λ-arm). Explain how to construct M_s when M has $k + 1$ arms (including the λ-arm) and each arm is a general MRP. Show that the value of M^s equals the minimum value of λ such that one would be indifferent in M between pulling the best arm and switching to the λ-arm forever.

See Katehakis and Veinott (1987) for the original proof and Cowan and Katehakis (2015) for a more up-to-date formulation of the problem.

Exercise 16.SELC

At each step in a **selection problem** (Section 16.3.4), an agent samples from one of k arms and obtains some information about the true utility of that arm. Selection problems differ from bandit problems in two ways: (1) the cost c of each sample is fixed, independent of the value of the arm, and (2) at some point the agent must stop sampling and commit to one of the arms. Let μ_i be the true utility of arm i and let $e_1, \ldots, e_N$ be the evidence obtained in the first N samples, after which the agent stops and commits. Then the agent's overall

expected utility, assuming it commits to what appears to be the best arm, is given by

$$E[-cN + max_i E[\mu_i \mid e_1, \ldots, e_N]] \; .$$

For parts (a–d) we will assume that the arms are Bernoulli arms, i.e., each sample returns 0 or 1 and the true utility μ_i is the probability that the arm returns a 1 on any given sample. The agent is assumed to have a prior probability $P_i(\mu_i)$ for each arm.

a. Give a precise formulation of the Bernoulli selection problem as an undiscounted MDP M with countably many states.

b. Because the state space of M is unbounded, some policies can keep sampling forever. A policy π that samples forever with finite probability (bounded away from zero) incurs infinite expected cost cN. Explain why an optimal policy for M stops with probability 1.

c. Give an explicit upper bound on the expected number of samples taken by an optimal policy.

d. Consider the two-armed selection problem where the agent knows that $\mu_1 = 1/2$ and that $\mu_2 = 1/3$ or $2/3$ with equal probability. Give a qualitative description of the optimal policy for this problem and show that it is *possible* for the optimal policy to keep sampling forever without ever committing to one arm or the other. Explain how this result is consistent with your answers in (b) and (c).

e. Now consider a selection problem with three *deterministic* arms that always return a fixed amount. (Such arms need be sampled at most once!) The agent knows that $\mu_1 = -1.5$ or $+1.5$ with equal probability, $\mu_2 = 0.25$ or $+1.75$ with equal probability, and $\mu_3 = \lambda$. There are three possible actions: sample arm 1, sample arm 2, and stop. Formulate and solve the corresponding MDP and plot the Q-value of each action as a function of λ for $\lambda \in [-2, +2]$. Show that the optimal choice between arms 1 and 2 can change depending on the value of λ, and hence that there is no index function for this problem.

[[TBC]]
[[need exercises]]

16.4 Partially Observable MDPs

Exercise 16.POMD

Let the initial belief state b_0 for the 4×3 POMDP on page 578 be the uniform distribution over the nonterminal states, i.e., $\langle \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, 0, 0 \rangle$. Calculate the exact belief state b_1 after the agent moves *Left* and its sensor reports 1 adjacent wall. Also calculate b_2 assuming that the same thing happens again.

The belief state update is given by Equation (16.16), i.e.,

$$b'(s') = \alpha\, P(e \mid s') \sum_s P(s' \mid s, a) b(s) \ .$$

It may be helpful to compute this in two stages: update for the action, then update for the observation. The observation probabilities $P(e \mid s')$ are all either 0.9 (for squares that actually have one wall) or 0.1 (for squares with two walls). The following table shows the results. Note in particular how the probability mass concentrates on (3,2).

		Left	1 wall	*Left*	1 wall
(1,1)	.11111	.20000	.06569	.09197	.02090
(1,2)	.11111	.11111	.03650	.04234	.00962
(1,3)	.11111	.20000	.06569	.09197	.02090
(2,1)	.11111	.11111	.03650	.27007	.06136
(2,3)	.11111	.11111	.03650	.05985	.01360
(3,1)	.11111	.11111	.32847	.06861	.14030
(3,2)	.11111	.11111	.32847	.30219	.61791
(3,3)	.11111	.02222	.06569	.03942	.08060
(4,1)	.11111	.01111	.00365	.00036	.00008
(4,2)	0	.01111	.03285	.03321	.06791
(4,3)	0	0	0	0	0

Exercise 16.PDPT

Consider a version of the two-state POMDP on page 581 in which the sensor is 90% reliable in state 0 but provides no information in state 1 (that is, it reports 0 or 1 with equal probability). Analyze, either qualitatively or quantitatively, the utility function and the optimal policy for this problem.

Policies for the 2-state MDP all have a threshold belief p, such that if $b(0) > p$ then the optimal action is Go, otherwise it is $Stay$. The question is, what does this change do to the threshold? By making sensing more informative in state 0 and less informative in state 1, the change has made state 0 more desirable, hence the threshold value p *increases*.

[[need exercices]]

16.5 Algorithms for Solving POMDPs

Exercise 16.TPDP

What is the time complexity of d steps of POMDP value iteration for a sensorless environment?

In a sensorless environment, POMDP value iteration is essentially the same as ordinary state-space search—the branching ocurs only on acton choices, not observations. Hence the time

complexity is $O(|A|^d)$.

Exercise 16.POMC

Silver and Veness (2011) describe the POMCP algorithm for approximate online decision-making in POMDPs. It is essentially a combination of the UCT algorithm and a particle filter to represent and update belief states.

a. Implement the POMCP algorithm as described. The algorithm itself should make no assumptions about how the POMDP model is implemented; that is, view it as an abstract data type.

b. For this part, use simple, atomic representations for states and percepts, with matrices for transition and sensor models. (See, for example, Section 14.3.) Create a model for the 4x3 POMDP and apply the algorithm to solve it. Measure the POMCP agent's performance as a function of the number of particles and the number of rollouts.

c. Now consider models represented as dynamic decision networks. Apply POMCP to solve vacuum worlds with stochastic dirt (see, for example, Figure 14.20), with a reward function based on the number of clean squares at the current step.

[[TBC]]
[[need exercises]]