Exercise 9

1. 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)

a. Write the Bellman equations for these formulations.

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

The key here is to get the max and summation in the right place

For R(s, a) ?

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

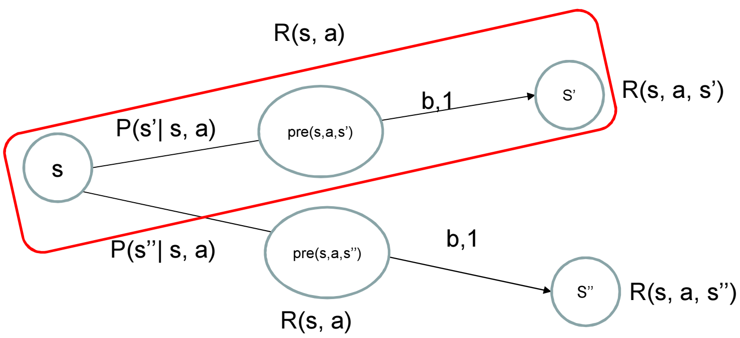
For R(s, a, s') ?

$$U(s) = max_a \sum_{s'} \boldsymbol{P}(s'|\, s,a)[R(s,a,s') + U(s')]$$

3

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.

Many solutions are possible.



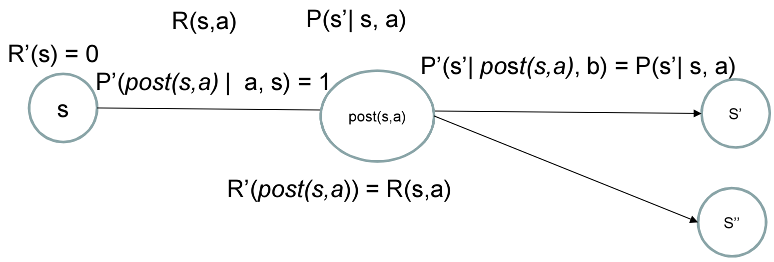
P'(*pre(s,a,s')*| a, s) = P(s'| s, a)
P'(s'| *pre(s,a,s')*, b) = 1
R'(s, a) = 0
R'(*pre(s,a,s')*, b) = R(s,a,s')

4

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

Follow the pattern of b.

P'(*post(s,a)* | a, s) = 1
P'(s'| *post(s,a)*, b) = P(s'| s, a)
R'(s) = 0
R'(*post(s,a)*) = R(s,a)



2. In this exercise we will consider two-player MDPs that correspond to zero-sum, turn taking games. Let the players be A and B, and let R(s) be the reward for player A in 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 (equations used for value iteration) defining $U_A(s)$ and $U_B(s)$.

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

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

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

Take the equations from a. and add t+1 and t respectively

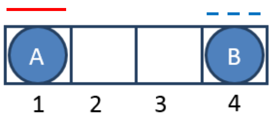
$$U_{A:t+1}(s) = R(s) + max_a \sum_a P(s'|a,s) U_{B:t}(s')$$

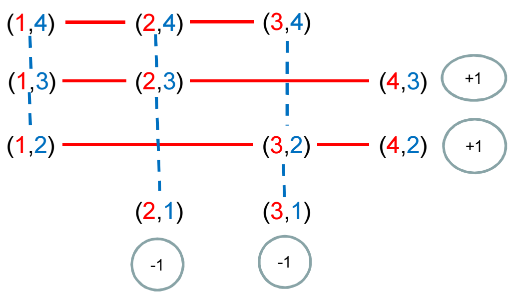
$$U_{B:t+1}(s) = R(s) + min_a \sum_a P(s'|a,s) U_{A:t}(s')$$

Stop if the utility vector of one player does not change for the player

9

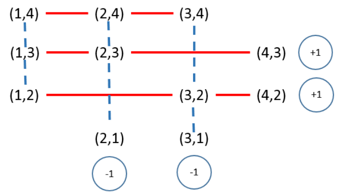
c. Consider the game described in the following figure. Draw the state space (rather than the game tree), showing the moves by A as solid lines and moves by B as dashed lines. Mark each state with R(s). You will End it helpful to arrange the states (s_A, s_B) on a two dimensional grid, using s_A and s_B as "coordinates." Assume A moves first.

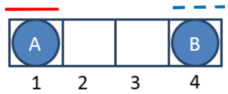




10

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





$$U_{A:t+1}(s) = R(s) + max_a \sum_a P(s'|a,s)U_{B:t}(s')$$

$$U_{B:t+1}(s) = R(s) + min_a \sum_a P(s'|a,s)U_{A:t}(s')$$

	(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

Is the same

The optimal policy

	(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)			

11

3. Give the pseudo code for policy iteration.
Explain how the major steps can be implemented

Initialize the policy vector π_0. An action for each state.
Set the initial utilities of non final states to 0.

repeat

Policy evaluation: given a policy π_i, calculate $U_i = U_{\pi i}$, the utility of each state if π_i were to be executed.

Policy improvement: Calculate a new MEU policy π_{i+1}

until *the policy does not change*

Policy improvement: Compute the best action based on U_i with one step look ahead as in value iteration.

Policy evaluation:

Solve the linear equations for π_i : $\mathcal{U}_t(i) \leftarrow \mathcal{R}(i) + \sum_k \mathbf{P}(k \mid \Pi(i).i)\, \mathcal{U}_t(k)$

Do *k* steps of value iteration for π_i :

$\mathcal{U}_{t+1}(i) \leftarrow \mathcal{R}(i) + \sum_k \mathbf{P}(k \mid \Pi(i).i)\, \mathcal{U}_t(k)$ 12

4. Consider an undiscounted Markov Decision Process (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:

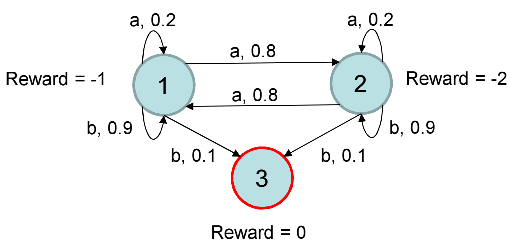
- In state 1, action a moves the agent to state 2 with probability 0.8 and makes the agent stay put with probability 0.2
- In state 2, action a moves the agent to state 1 with probability 0.8 and makes the agent stay put with probability 0.2
- In either state 1 or state 2, action b move the agent to 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 state 1 and state 2?
b. Apply policy iteration, showing each step in full, to determine the optimal policy and the values of state 1 and state 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?
d. Now, use value iteration.

13

Leads to the following state diagram.



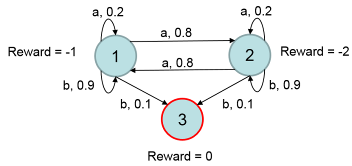
a. What can be determined qualitatively about the optimal policy in state 1 and state 2?

In state one choose b and in state 2 choose a. This is because reaching state 3 has a small probability and the cost of state 2 is higher than the cost of state 1.

14

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

I choose solving linear equations.



$\pi(b, b)$

1. Policy evaluation for U

$$\mathcal{U}(s) \leftarrow \mathcal{R}(s) + \sum_k \mathbf{P}(k \mid \Pi(s))\, \mathcal{U}(k)$$

$u_1 = -1 + 0.1 * u_3 + 0.9 * u_1 \rightarrow u_1 = -1 + 0.9 * u_1 \rightarrow 0.1 u_1 = -1 \rightarrow u_1 = -10$
$u_2 = -2 + 0.1 * u_3 + 0.9 * u_2 = -20$
$u_3 = 0$

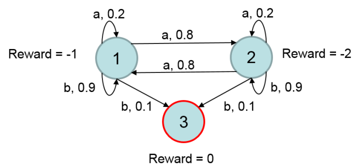
2. Policy improvement. Compute the EU for each state action pair:

$$\Pi'(s) = \arg\max_a \sum_k \mathbf{P}(k \mid s,a)\, \mathcal{U}(k)$$

$u(1,a) = 0.8 * -20 + 0.2 * -10 = -18$
$u(1,b) = 0.1 * 0 + 0.9 * -10 = -9$ $\rightarrow$ b

$u(2,a) = 0.8 * -10 + 0.2 * -20 = -12$ $\rightarrow$ a, policy update $\pi(b, a)$
$u(2,b) = 0.1 * 0 + 0.9 * -20 = -18$

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



$\pi(b, a)$

1. Policy evaluation for U

$u_1 = -1 + 0.1 * u_3 + 0.9 * u_1 = -10$
$u_2 = -2 + 0.8 * u_1 + 0.2 * u_2 = -12.5$
$u_3 = 0$

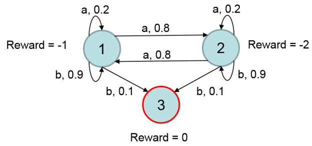
2. Policy improvement. Compute the EU for each state action pair:

$u(1,a) = 0.8 * -12.5 + 0.2 * -10 = -12$
$u(1,b) = 0.1 * 0 + 0.9 * -10 = -9$ $\rightarrow$ b

$u(2,a) = 0.8 * -10 + 0.2 * -12.5 = -10.5$ $\rightarrow$ a
$u(2,b) = 0.1 * 0 + 0.9 * -12.5 = -11.25$

The policy was not changed, the process terminates with $\pi(b, a)$.

c. Now the initial policy us π(a,a)



π(a, a)

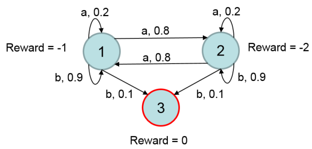
1. Value determination for U

$u_1 = -1 + 0.2 * u_1 + 0.8 * u_2$
$u_2 = -2 + 0.8 * u_1 + 0.2 * u_2$
$u_3 = 0$

The first two equations are inconsistent. There is no solution.
One can solve it iteratively with a small discount factor γ.

17

d. Now, use value iteration. Set γ =1.



$\mathcal{U}_{t+1}(s) \leftarrow \mathcal{R}(s) + \max_a \Sigma_k \mathbf{P}(k \mid s,a)\, \mathcal{U}_t(k)$
if $\mathcal{U}_{t+1}(i) - \mathcal{U}_t(k) < \varepsilon$

return $\Pi^*(s) = \arg\max_a \Sigma_k \mathbf{P}(k \mid s,a)\, \mathcal{U}(k)$

```
Round: 25
State 1= -9.28210201230815
State 2= -11.679545156923599

Policy: (b,a)
```

18

```java
public class ValueIteration_Ex9 {
    int state1 = 0;
    int state2 = 1;
    int state3 = 2;
    double R_State1 = -1.0;
    double R_State2 = -2.0;
    double R_State3 = 0.0;
    double utility_a = 0;
    double utility_b = 0;
    double[] U_Current = { 0.0, 0.0, 0.0 };
    double[] U_Last = { 0.0, 0.0, 0.0 };
    private String state1Action;
    private String state2Action;

    double U_1() {
        utility_a = R_State1 + 0.8 * U_Last[state2] + 0.2 * U_Last[state1];
        utility_b = R_State1 + 0.1 * U_Last[state3] + 0.9 * U_Last[state1];
        return Math.max(utility_a, utility_b);

    }

    double U_2() {
        utility_a = R_State2 + 0.8 * U_Last[state1] + 0.2 * U_Last[state2];
        utility_b = R_State2 + 0.1 * U_Last[state3] + 0.9 * U_Last[state2];
        return Math.max(utility_a, utility_b);
    }


    private void start() {
        double error = 0.1;
        boolean cont = true;
        int round = 1;
        while (cont) {
            U_Current[state1] = U_1();
            U_Current[state2] = U_2();
            printUtilities(round);
            if ((Math.abs(U_Last[state1] - U_Current[state1]) < error)
                    && (Math.abs(U_Last[state2] - U_Current[state2]) < error))
                cont = false;
            round++;
            U_Last[state1] = U_Current[state1];
            U_Last[state2] = U_Current[state2];
        }
        printPolicy();

    }
```