CS5180 Reinforcement Learning

Exercise 2: Markov Decision Processes

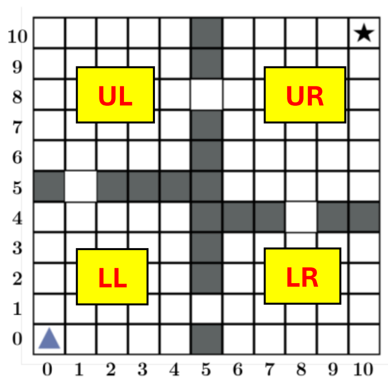
Anway Shirgaonkar

Q1 Formulating an MDP

(a) The state space of the four rooms environment is the set of all valid states which the agent can assume. Essentially, it is all the coordinates $(0, 0)$ through $(10, 10)$ except the walls.

The action space of the agent is the set of all possible actions that the agent can take, i.e. $\{UP, DOWN, LEFT, RIGHT\}$

(b) Considering the dynamics function $p(s', r|s, a)$



Consider the four quadrants as labelled in the figure above.

Type of cell	LL	LR	UL	UR	Total Cells	Possible s'	Total states s'
Normal	11	8	11	14	44	(44*4*3)	528
Edges	10	8	10	12	40	(40*2*2) + (40*2*3)	400
Corners	4	4	4	3	15	(15*2*2) + (15*2*2)	120
Doors	NA				4	(4*2*2) + (4*3*2)	40
Goal	NA				1	4	4
						Sum of total states s'	1088

For normal cells, the agent can move into any of the four neighboring cells depending on the action. An action in these cells can result in 3 future states s' (considering stochasticity). So, 4 actions will correspond to 12 possible future states for each cell.

Similarly, if we calculate this for all the types of cells, the total estimated number of non-zero rows in the $p(s', r|s, a)$ table is about 1088

(c) I have uploaded a separate file named probability_table.pdf which contains the probability table generated with code

```
import numpy as np
from enum import Enum
import random
```

```
class Action(Enum):
    UP      = [ 0,  1]
    DOWN    = [ 0, -1]
    LEFT    = [-1,  0]
    RIGHT   = [ 1,  0]

class Environment:
    def __init__(self) -> None:

        self.WALLS = [
            [0, 5], [2, 5], [3, 5], [4, 5], [5, 5],
            [5, 0], [5, 2], [5, 3], [5, 4], [5, 5], [5, 6], [5, 7], [5, 9], [5, 10],
            [6, 4], [7, 4], [9, 4], [10, 4]
        ]

        self.GOAL = [10, 10]

    def get_possible_states(self, state, action:Action):
        """
        This function returns an array with the all possible valid states, when the current state and an action is given as input
        """
        specified_state = list(map(sum, zip(state, action.value)))

        if not all(coordinate >= 0 and coordinate <= 10 for coordinate in specified_state) or specified_state in self.WALLS:
            specified_state = state

        noisy_actions = [Action.UP, Action.DOWN] if action == Action.RIGHT or action == Action.LEFT else [Action.LEFT, Action.RIGHT]
        noisy_states_unfiltered  = [list(map(sum, zip(state, noisy_action.value))) for noisy_action in noisy_actions]
        noisy_states = list(filter(lambda noisy_state: all(coordinate >= 0 and coordinate <= 10 for coordinate in noisy_state) and noisy_state not in self.WALLS,
                                   noisy_states_unfiltered))
        if noisy_states == []:
            noisy_states.append(state)

        possible_states = [specified_state] + noisy_states

        return(possible_states)
```

```
env = Environment()
dim_x = 10
dim_y = 10

for x in range(dim_x+1):
    for y in range(dim_y+1):
        if [x, y] in env.WALLS:
            pass

        else:
            for a in Action:
                possible_actions = env.get_possible_states([x, y], a)
                prob = [0.8, 0.1, 0.1] if len(possible_actions)==3 else [0.9, 0.1]
                for p_idx, p in enumerate(possible_actions):
                    r = 1 if p == env.GOAL else 0
                    print(f"{[x, y]}\t  {a.name}\t  {p}\t  {r}\t  {prob[p_idx]}")
```

Q2 The RL objective

(a) If pole balancing is treated as an episodic task with discounting, the resulting goal would be given by the following equation:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots + \gamma^{n-1} R_{t+n}$$

Since the reward at each step is 0, except for the last step, in which we receive a reward of -1, corresponding to failure. Hence,

$$\boxed{G_t \doteq -\gamma^{n-1}}$$

Where n is the number of steps in each episode.

For continuing cases, the goal would be given by the following equation:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$$

Assume that failure occurs at steps $\{t_1, t_2, t_3, \ldots.\}$ Hence,

$$\boxed{G_t \doteq \sum_{t \in \{t_1, t_2 \ldots\}} -\gamma^{t-1}}$$

In the episodic case, the episode ends upon failure. Hence, $0 \geq G_t \geq -1$. The more time steps we spend without failure, the more G_t would be closer to 0 (given $\gamma < 1$).

(b) The following are the reasons for the agent to not show any signs of improvement.

- The maximum reward that the agent can achieve at the end of each episode is $+1$ for escaping the maze. The reward is the same regardless of the time it takes to escape the maze.
- To make sure that the agent is learning to escape the maze as quickly as possible, a small negative reward could be added at each time step.
- So now, the agent's reward will keep reducing for the time he spends exploring the maze and would increase on escape. A good policy would be then able to escape the maze quickly.

Q3 Discounted Return

(a) $\gamma = 0.5,\ \ R_1 = -1,\ \ R_2 = 2,\ \ R_3 = 6,\ \ R_4 = 3,\ \ R_5 = 2,\ \ T = 5$

$G_5 = 0, since\ T = 5$ (episode ends at 5)

$G_4 = R_5 + \gamma G_5 = 2$

$G_3 = R_4 + \gamma G_4 = 4$

$G_2 = R_3 + \gamma G_3 = 8$

$G_1 = R_2 + \gamma G_2 = 6$

$G_0 = R_1 + \gamma G_1 = 2$

(b) $\gamma = 0.9,\ \ R_1 = 2$

$$G_1 = R_2 + \gamma R_3 + \gamma^2 R_4 + \cdots$$

Now, $R_1 = R_2 = R_3 = \cdots = 7$

$$G_1 = 7(\gamma^0 + \gamma^1 + \gamma^2 + \cdots)$$

$$G_1 = 7\sum_{k=0}^{\infty} \gamma^k = \frac{7}{1-\gamma}$$

$$\boxed{\Longrightarrow G_1 = \frac{7}{1-0.9} = 70}$$

Now,

$$\boxed{G_0 = R_1 + \gamma G_1 = 65}$$

Q4 Discount factor

For action = UP

$$G_t(s = start, a = UP) = 50 - \gamma^1 - \gamma^2 - \cdots - \gamma^{100}$$

$$G_t(s = start, a = UP) = 50 - \sum_{i=1}^{100} \gamma^i$$

Applying laws of a geometric progression for γ,

$$\boxed{G_t(s = start, a = UP) = 50 - \gamma\left(\frac{1 - \gamma^{100}}{1 - \gamma}\right)}$$

For action = $DOWN$

$$G_t(s = start, a = DOWN) = -50 + \gamma^1 + \gamma^2 + \cdots + \gamma^{100}$$

$$G_t(s = start, a = DOWN) = -50 + \sum_{i=1}^{100} \gamma^i$$

Applying laws of a geometric progression for γ,

$$\boxed{G_t(s = start, a = DOWN) = -50 + \gamma\left(\frac{1 - \gamma^{100}}{1 - \gamma}\right)}$$

Now, let's see when choosing the UP action would be better

$G_t(s = start, a = UP) > G_t(s = start, a = DOWN)$

$\Rightarrow 50 - \gamma\left(\frac{1 - \gamma^{100}}{1 - \gamma}\right) > -50 + \gamma\left(\frac{1 - \gamma^{100}}{1 - \gamma}\right)$

$\Rightarrow 50 > \gamma\left(\frac{1 - \gamma^{100}}{1 - \gamma}\right)$

$\Rightarrow 50(1 - \gamma) > \gamma(1 - \gamma^{100})$

$\Rightarrow 50 - 50\gamma > \gamma - \gamma^{101}$

$\Rightarrow \gamma^{101} - 51\gamma - 50 > 0$

Solving this equation in Wolfram Alpha, the valid solution is $\gamma < 0.9843$

Hence, choosing UP is better if $\gamma < 0.9843$, else choosing $DOWN$ is better.

Q5 Modifying the reward function

(a) Equation 3.8 is given as the discounted return for a continuing task:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

If we add a constant c to all the rewards, we get

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k (R_{t+k+1} + c)$$

$$\Longrightarrow G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} + \sum_{k=0}^{\infty} \gamma^k c$$

$$\Longrightarrow G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} + \sum_{k=0}^{\infty} \gamma^k c$$

$$\Longrightarrow G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} + \frac{c}{1-\gamma}$$

Hence,

$$\boxed{v_c = \frac{c}{1-\gamma}}$$

The constant term v_c depends only on the values of c and γ, and is added to the discounted return. Hence it is proven that adding a constant term to each individual reward does not affect relative values of any states under any policies.

(b) The episodic return is given by (where $t+n$ is the last time step):

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots + \gamma^{T-t-1} R_T$$

Adding a constant to reward at each step gives us:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1}(R_k + c) = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k + \sum_{k=t+1}^{T} \gamma^{k-t-1} c$$

Not a constant!

In case of a maze runner, let's say that the agent gets a small negative reward -0.1 for each time step that the agent spends exploring the maze, and a positive reward $+10$ for escaping the maze. Now, let's say that a constant $+1$ is added to reward at each state. So, the reward at each time step is now 0.9.

Now the agent has no incentive to escape the maze, and it would just be running around the maze to maximize its cumulative reward. Thus, adding a constant value to reward at each state has a significant effect on an episodic task.

Q6 Bellman equation

(a) The Bellman equation is:

$$v_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_\pi(s')] \;\; for\ all\ s \in S$$

$$v_\pi(s = centre) = 0.25\big[1[0 + 0.9 * 2.3]\big] + 0.25\big[1[0 + 0.9 * -0.4]\big] + 0.25\big[1[0 + 0.9 * 0.7]\big] + 0.25\big[1[0 + 0.9 * 0.4]\big]$$

$$v_\pi(s = centre) = 0.25[2.07 - 0.36 + 0.63 + 0.36]$$

$$\boxed{v_\pi(s = centre) = 0.675}$$

The actual value in the table is 0.7 (rounded off). Hence the bellman equation holds for the center state.

(b) The Bellman optimality equation is:

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')]$$

$$v_*(s) = max\{1[0 + 0.9 * 19.8], 1[0 + 0.9 * 19.8], 1[0 + 0.9 * 16.0], 1[0 + 0.9 * 16.0]\}$$
$$v_*(s) = max\{17.82, 17.82, 14.4, 14.4\}$$
$$\boxed{v_*(s) = 17.82}$$

The actual value in the table is 17.8 (rounded off). Hence the bellman equation holds for the center state.

Q7 Guessing and verifying value functions.

(a) The value function by definition is the expected return when starting in a state s and then following a policy π thereafter.

In our case, we can easily guess the value function $v_\pi(s = A)$ for equiprobable random policy.

Since there is equal probability of taking the left and right actions, and the reward on taking the right action is $+1$, we can estimate that $v_\pi(s = A) = 0.5$

Let us verify this using the Bellman equation:

$$v_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_\pi(s')] \; for\ all\ s \in S$$

Note that $v_\pi(s) = 0$ if s is a terminal state.

$\implies v_\pi(s = A) = 0.5\big[1[0 + 0]\big] + 0.5[1[1 + 0]]$

$\boxed{\implies v_\pi(s = A) = 0.5}$

Hence, our estimated value function is verified using the Bellman equation.

(b) I guess that the value function of this MDP should increase as we move from the left towards the right, i.e. closer to obtaining the +1 reward.
Let us verify this using the Bellman equation:

$v_\pi(s = A) = \ 0.5 * v_\pi(s = B)$
$v_\pi(s = B) = 0.5 * v_\pi(s = A) + 0.5 * v_\pi(s = C)$
$v_\pi(s = C) = 0.5 * v_\pi(s = B) + 0.5 * v_\pi(s = D)$
$v_\pi(s = D) = 0.5 * v_\pi(s = C) + 0.5 * v_\pi(s = E)$
$v_\pi(s = E) = 0.5 + 0.5 * v_\pi(s = D)$

Solving the above system of equations,

$$\boxed{\begin{aligned} v_\pi(s = A) &= \frac{1}{6} \\ v_\pi(s = B) &= \frac{2}{6} \\ v_\pi(s = C) &= \frac{3}{6} \\ v_\pi(s = D) &= \frac{4}{6} \\ v_\pi(s = E) &= \frac{5}{6} \end{aligned}}$$

Hence, it is verified that the value function increases as we move from left to right.

(c) For an arbitrary number of states n, the value function is:

$$v_\pi(s) = \frac{\alpha}{\beta}$$

Where,

α = number of transitions from the left most termination state to current state

β = total transitions from left most state to right most state

Q8 Solving for the value function

(a) The Bellman equation is:

$$v_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_\pi(s')] \ for\ all\ s \in S$$

For the recycling robot example, we can expand the Bellman equation as:

$$\begin{aligned} v_\pi(s = high) = {} & \pi(search|high) * \alpha * \ [r_{search} + \gamma v_\pi(s' = high)] \\ & + \pi(search|high) * (1 - \alpha) * [r_{search} + \gamma v_\pi(s' = low)] \\ & + \pi(wait|high) * [r_{wait} + \gamma v_\pi(s' = high)] \\ & + \pi(wait|high) * 0 * [r_{wait} + \gamma v_\pi(s' = low)] \end{aligned}$$

$$\begin{aligned} v_\pi(s = low) = {} & \pi(search|low) * (1 - \beta) * [r_{search} + \gamma v_\pi(s' = high)] \\ & + \pi(search|low) * \beta * \ [r_{search} + \gamma v_\pi(s' = low)] \\ & + \pi(wait|low) * 0 * [r_{wait} + \gamma v_\pi(s' = high)] \\ & + \pi(wait|low) * [r_{wait} + \gamma v_\pi(s' = low)] \\ & + \pi(recharge|low) * \left[r_{recharge} + \gamma v_\pi(s' = high)\right] \\ & + \pi(recharge|low) * 0 * \left[r_{recharge} + \gamma v_\pi(s' = low)\right] \end{aligned}$$

(b) Substituting the given values,

$$v_\pi(s = high) = 0.8\big(10 + 0.9 v_\pi(s = high)\big) + 0.2\big(10 + 0.9 v_\pi(s = low)\big)$$

$$\Longrightarrow 0.28 v_\pi(s = high) = 10 + 0.18 v_\pi(s = low)$$

$$v_\pi(s = low) = 0.5\big(3 + 0.9 v_\pi(s = low)\big) + 0.5(0.9 v_\pi(s = high))$$

$$\Longrightarrow 0.55 v_\pi(s = low) = 1.5 + 0.45 v_\pi(s = high)$$

Solving the above system of equations in Wolfram Alpha,

$$\boxed{\begin{aligned} & v_\pi(s = high) = 79.041 \\ & v_\pi(s = low) = 67.397 \end{aligned}}$$