Decision Making Under Uncertainty: Homework 2

Julian Lema Bulliard*
University of Colorado Boulder

I. Questions

A. Question 1

a) The difference between a reward function and the state-action (Q) value function is as follows. A reward function will tell us our reward at the current state. For example, if I am in high school and I decide to go out with friends on a school night my mom would give me a negative reward (punishment), while if I decide to stay in and study I will be positively rewarded (she won't be nagging me). Now, a state-action value function is a way to predict what the reward for my next action will be based on previous actions.

b) The state-action value function can be written as follows

$$Q(s,a) = R(s,a) + \gamma T(s,a,s')V(s)$$

- Q(s,a) is the value function dependent on the current state and the action we take.
- R(s,a) is the reward function dependent on the current state and the action we take.
- γ is the discount factor that we choose for out learning model
- T(s,a,s') is the transition function which tells us how likely we are to reach a future state *s'* after taking action *s*
- V(s) is the state-value function that we hashed out previously

Where Q(s,a) is the value function dependent on the current state and the action we take.

B. Question 2

a) Writing problem down as an MDP

- S = [1st Square , 2nd Square , 3rd Square]
- A = [Reset, Roll Die]

*Graduate Student, Aerospace Engineering Department

- T_{Rolls} =

Action = Rolls	1st Square	2nd Square	3rd Square
1st Square	0	0.5	0.5
2nd Square	0	0	1
3rd Square	0	0	1

- T_{Reset} =

Action = Reset	1st Square	2nd Square	3rd Square
1st Square	1	0	0
2nd Square	1	0	0
3rd Square	1	0	0

- Reward =

R(s,a)	Reward
R(3rd,reset)	-1
R(2nd,reset)	2
R(Otherwise)	0

b) The optimal policy can be calculated using the policy iteration scheme that was shown in class.

First, we will set up what should be done at each state.

State	Action
1st	Roll
2nd	Reset
3rd	Reset

We can then construct the Transition Matrix

$T_{optimal}$ =

Optimal	1st Square	2nd Square	3rd Square
1st Square	0	0.5	0.5
2nd Square	1	0	0
3rd Square	1	0	0

We can also construct the Reward Matrix according to our ending space. And what action we would take there. For example, we can see that for a state equals two, we will be resetting therefore we get 2 'points'

State	Reward
1st	0
2nd	2
3rd	-1

From here, I was able to code the value iteration code in the slides and was given a maximum policy of 4.87

c) For this question we will start with the following equation.

$$\frac{\underline{r}}{1-\gamma} \leqslant \sum_{t=0}^{\infty} \gamma^t r_t \leqslant \frac{\bar{r}}{1-\gamma}$$

For $\underline{r}$ we can assume that we are getting the lowest reward possible which is -1. This coupled with the gamma, the first part of the equation becomes -20. Overbar 'r' becomes 2 as this is the maximum reward that can be obtained. The right side of the equation becomes 40. Therefore we obtain that the result of the weighted die simulation becomes

$$-20 \leqslant \sum_{t=0}^{\infty} 0.95^t r_t \leqslant 40$$

C. Question 3

For question three, I had a very similar approach to creating the Value Iteration function as we used in question 2. The only change that had to be made was the difference in the type of data that we were inputting. In question 2 we were mainly working with arrays and matrices that I had created, while on this problem we started using dictionaries, and I had to split them up into matrix components, and then iterate to multiply each action with each transition matrix. This can be seen in the code appendix. The output of my value iteration function can be seen below. This is the output of the

'grid world' environment with my policy applied.

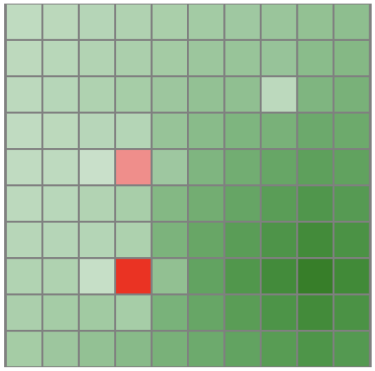

Fig. 1 Grid World after running the Value Iteration Julia Script

D. Question 4

Luckily, in question four, I was able to apply the same exact equation for my value iteration as I had used in the previous question. The results of my code can be seen in the '.json' file that I also submitted on Gradescope in which I received a score of 15. I tried reaching for high points by my computer could no longer handle it.

II. Appendix

```
using DMUStudent.HW2
using POMDPs: states, actions
using POMDPTools: ordered_states
using LinearAlgebra

##############
# Question 2
##############

# Create a state and an action matrix
# First row = State | Second Row = Action (1 = roll , 0 = reset)
 Q = [1 2 3; 1 0 0]

# Create a transition Matrixs
T = [0 0.5 0.5; 1 0 0 ; 1 0 0]

# Reward Matrix
R = [0 ; 2 ; -1] # Reward given in order of what actions we would take at each step

# Conditions for value_iteration function
discount = 0.95
epsilon = 1e-10
max_iterations = 300
function ValueIteration(T, R, gamma, epsilon, n_max)
    # Julian Lema
    #=
    Inputs;
    T - Transition Function
    R - Column Vector of the Rewards
    gamma - Scalar of given discount factor
    epsilon - tolerance for convergence
    n_max = Max iterations if we do not converge under epsilon
```

```
    =#
    V = zeros(size(R, 1))
    i = 0 # Iteration

    # Repeat until convergence or maximum iterations are reached
    while i < n_max
        i = i + 1 # Update counter
        V_new = R + gamma * T * V

        # Check for convergence
        if norm(V_new - V,Inf) < epsilon #Infinity norm
            break
        end

        V = V_new
    end

    return V[1], i
end

ValueIteration(T, R, discount, epsilon, max_iterations)



############
# Question 3
############



S = states(grid_world)
A = actions(grid_world)
T = transition_matrices(grid_world)
R = reward_vectors(grid_world)
gamma = 0.95
```

```
epsilon = 1e-10
n_max = 1000 # Not used
gym = grid_world # Inputing for measurments

function value_iteration(S,A,T,R,gamma,epsilon,n_max,gym)
    # Value_iteration function for the grid search algorithm
    # Julian Lema
    #=
    Inputs:
    S - States possible
    A - Actions possible (left,right,etc.)
    T - Transition Function
    R - Column Vector of the Rewards
    gamma - Scalar of given discount factor
    epsilon - tolerance for convergence
    n_max = Max iterations if we do not converge under epsilon (Not Used)
    gym - this in inputting the environment into the function so we can call states on it
    =#

    # Initialize everything first
    V = rand(length(states(gym)))
    V_prime = rand(length(states(gym)))


    while maximum(abs.(V-V_prime)) > epsilon
        V = V_prime # Switch em'
        Vhold = fill(-Inf,length(S)) # Make -Inf so will be overwritten no matter what

        for i in A # Iterate through action since we working with dicts
            Vhold = max.(Vhold,R[i] + gamma * T[i] * V_prime) # Find max of iterations
        end
        # Establish max and restart
        V_prime = Vhold # Store max and repeat if needed
```

```
    end

    return V_prime # Output optimal policy
end

V_prime = value_iteration(S,A,T,R,gamma,epsilon,n_max,gym)
# # You can use the following commented code to display the value. If you are in an environment with mul
display(render(grid_world, color= V_prime ))

############
# Question 4
############
# Inputs

function value_iteration(S,A,T,R,gamma,epsilon,n_max,gym)
    # Value_iteration function for the grid search algorithm
    # Julian Lema
    #=
    Inputs:
    S - States possible
    A - Actions possible (left,right,etc.)
    T - Transition Function
    R - Column Vector of the Rewards
    gamma - Scalar of given discount factor
    epsilon - tolerance for convergence
    n_max = Max iterations if we do not converge under epsilon
    =#

    # Initialize everything first
    V = rand(length(states(gym)))
    V_prime = rand(length(states(gym)))
```

```
    while maximum(abs.(V-V_prime)) > epsilon
        V = V_prime # Switch em'
        Vhold = fill(-Inf,length(S)) # Make -Inf so will be overwritten no matter what

        for i in A # Iterate through action since we working with dicts
            Vhold = max.(Vhold,R[i] + gamma * T[i] * V_prime)
        end
        # Establish max and restart
        V_prime = Vhold
    end

    return V_prime
end
# You can create an mdp object representing the problem with the following:
gym = UnresponsiveACASMDP(15)
S = states(gym)
A = actions(gym)
T = transition_matrices(gym, sparse = true)
R = reward_vectors(gym)
gamma = 0.99
epsilon = 1e-8
n_max = 1000 # Not used

V = value_iteration(S,A,T,R,gamma,epsilon,n_max,gym)

@show HW2.evaluate(V,"julian.lemabulliard@colorado.edu")
```