Imperial College London

Imperial College London

Department of Bioengineering

Bits, Brains and Behaviour

Coursework 1

Author:

Jorge Jacobo Bennasar Vázquez

CID: 01809233

Question 1: Understanding of MDPs

1.

The Markov Decision Process, of states $S = \{s_0, s_1, s_2\}$ and $\gamma = 1$, will be associated to my CID (01809233). Omitting the leading 0: CID (t) = 1809233, being: CID (1) = 1 … CID (7) = 3. The formulas to apply to find the trace are:

$$s(t) = (CID(t) + 2)\ mod\ 3 \qquad r(t+1) = CID(t)\ mod\ 2$$

Using them, the trace (τ) turns out to be: $\tau = s_0\ 1\ s_1\ 0\ s_2\ 0\ s_2\ 1\ s_1\ 0\ s_2\ 1\ s_2\ 1$

2.

a)

The partially known Markov Decision Process diagram is (Figure 1):

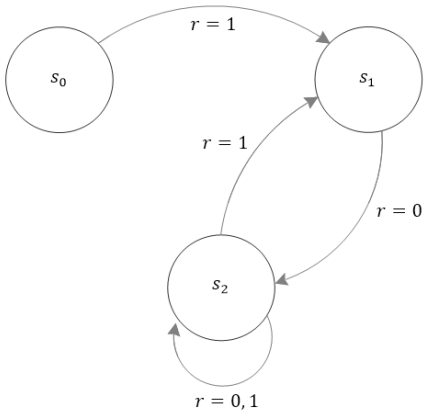

Figure 1. Partially known MDP diagram

About the structure of the transition matrix ($P_{ss'}$) it is possible to infer:

- There is no absorbing state. This is, there is no state in which the probability of leaving is null. This can be proved taking into account that:

$$P_{01} > 0 \rightarrow \sum_{s'=1,2} P_{0s'} > 0, \qquad P_{12} > 0 \rightarrow \sum_{s'=0,2} P_{1s'} > 0, \qquad P_{21} > 0 \rightarrow \sum_{s'=0,1} P_{2s'} > 0$$

- It is also possible to narrow the value of each of the matrix's components:

$$\begin{array}{lll} 0 \le P_{00} < 1, & 0 < P_{01} \le 1, & 0 \le P_{02} < 1 \\ 0 \le P_{10} < 1, & 0 \le P_{11} < 1, & 0 < P_{12} \le 1 \\ 0 \le P_{20} < 1, & 0 < P_{21} < 1, & 0 < P_{22} < 1 \end{array}$$

Furthermore, we can also infer about the reward function (R_s):

- Taking into account that there is "no choice" (only one action possible), the rewards are determined as a function of s and s' ($r(s, s')$). However, as it is observable in the diagram, at least $r(s_2, s_2)$ is stochastic, which means that for the same movement (s_2 to s_2) it does not always give the same reward (0

or 1). For the other rewards it is not possible to know if they are deterministic or stochastic. The partially known reward matrix would look like the following (r_i used to model stochasticity):

$$R_{ss'} = \begin{bmatrix} X & 1 & X \\ X & X & 0 \\ X & 1 & r_i \end{bmatrix} \; r_i \; with \; probability \; p_i; \; being \; \sum_i p_i = 1, r_1 = 0 \; (p_1 > 0) \; and \; r_2 = 1 \; (p_2 > 0)$$

b)

Taking into account that we do not have a complete model of the environment, DP cannot be used to calculate the value of s_0. Additionally, due to the fact that there is no absorbing state, the MC approach should not be applied either. Being TD the only approach left, there is not enough data to obtain a satisfactory result (state s_0 is only involved in one action). Therefore, even though TD is the correct approach to apply, more data of the environment should be collected before doing so.

Question 2: Understanding of Grid Worlds

1.

Taking into account my CID (01809233), then x = 2, y = 3 and z = 3. Applying the following formulas, we obtain p and γ:

$$p = 0.25 + 0.5 \times \frac{x}{10} = 0.35 \qquad\qquad \gamma = 0.2 + 0.5 \times \frac{y}{10} = 0.35$$

Additionally, we know that the state s_{11} has an arriving reward of -100 and that s_j has one of 10, j being: $j = \big((z+1) \; mod \; 3\big) + 1 = 2$

The rest of states have an arriving reward of -1. Therefore, the reward vector is (take into account that we receive the reward when we enter its assigned state):

$$R_s = [-1, 10, -1, -1, -1, -1, -1, -1, -1, -100]$$

2.

The problem was solved with a Value Iteration Algorithm. In it, the rewards are given when entering a state. This is the reason why the values of terminal states s_2 and s_{11} are null (starting in those states does not count as arriving at them). The algorithm followed the following logic:

Given the states ($S = \{s_1, \dots, s_{11}\}$), the actions ($A = \{east, west, north, south\}$), the trastition matrix ($P^a_{ss'}$), the reward vector (R), the discount factor (γ) and the probability of success (p):
$V(s_2) = V(s_{11}) = 0$ (as stated above, both states have value 0)
Arbitrary initialization of V for the rest of states and of O(optimal policy) for $s \in S$
$q = (1-p)/3$ (probability to do one of the not chosen actions)
Repeat
    $\Delta = 0$
    For each state s (except s_2 and s_{11} because their values are known)

$V_{aux}(s) = V(s)$ (auxiliary vector of state values)
$X = -\infty$ (auxiliary variable)
For each action a
$Y = 0$ (auxiliary variable)
For each state s' (including s_2 and s_{11})

$$Y = Y + \left(p \times P^{a}_{ss'} \times \left(r_{s'} + \gamma \times V_{aux}(s')\right)\right) + \left(q \times \sum_{b \in A \neq a} P^{b}_{ss'} \times \left(r_{s'} + \gamma \times V_{aux}(s')\right)\right)$$

If $Y \geq X$ then $X = Y$ and $O(s) = a$
$V(s) = X$
If $\Delta < |V(s) - V_{aux}(s)|$ then $\Delta = |V(s) - V_{aux}(s)|$
Until $\Delta < \theta$ (a small positive number)

From this algorithm we obtain the value of each state V and the optimal policy O. The results are shown in the following figure (Figure 2):

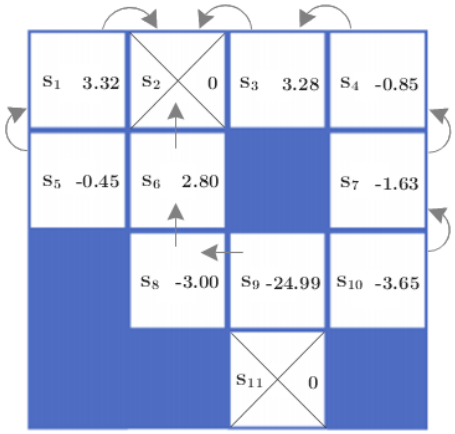

Figure 2. Optimal values for each state and optimal policy

3.

Intuitively we can guess that the optimal policy to execute in state s_9 is to turn left ($O\ (s_9) = west$). The algorithm gives that same result. However, even when executing the optimal policy, being in s_9 leads to a sensible situation due to the fact that the desired action is only executed with probability $p = 0.35$. That gives a big margin for error, with a considerable probability q of landing in terminal state s_{11}:

$$q = (1 - p)/3 \sim 0.217$$

It is interesting to notice that if p is very small, the algorithm would choose as the optimal action in s_9 to move south towards s_{11}. This would cause an increment in the value of s_9 due to the small probability of landing in s_{11}. On the other hand, if p is close to 1, $V(s_9)$ will also get bigger because of the smaller probability of error. Furthermore, it is deductible that the bigger the γ (with $p = 0.35$), the lower the value of s_9. This is because a lower γ reduces the effect of $Q\ (s_9, south)$ in the surrounding states (s_8, s_{10}...). When γ gets closer to 1 the values of these states become more negative, thus $V(s_9)$ too. Both of these tendencies can be observed in figures 3 and 4 (see appendix for the whole 3D plot):

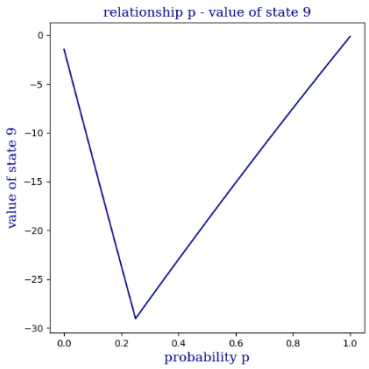

Figure 3. Relationship p - $V(s_9)$ ($\gamma = 0.35$)

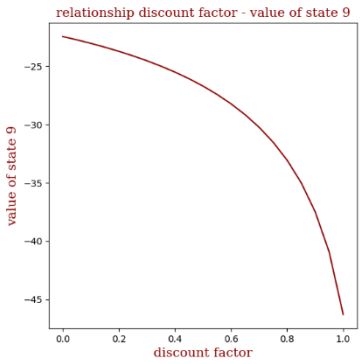

Figure 4. Relationship γ - $V(s_9)$ ($p = 0.35$)

4.

As it is observable in Figure 2, in this problem the algorithm gives the intuitively expected optimal policy. Due to the relevant discount factor ($\gamma = 0.35$), and with the exception of s_9 (not affected by γ when jumping to s_{11}), the state values are not as low as expected taking into account that: $|r_{11}| = |-100| = 100 \gg r_2 = 10$

The role of γ in attenuating the effect of the state-action value Q (s_9, *south*) = -100 in the value of the rest of states can be observed in the following figure (Figure 5):

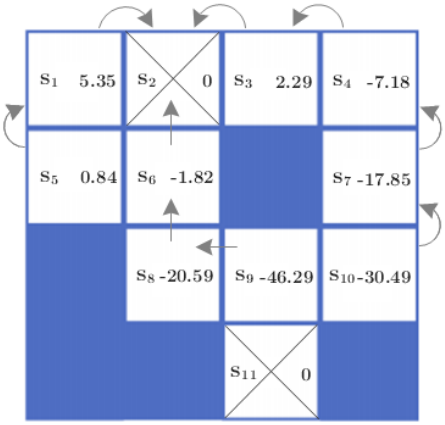

Figure 5. Optimal values and policy for $\gamma = 1$

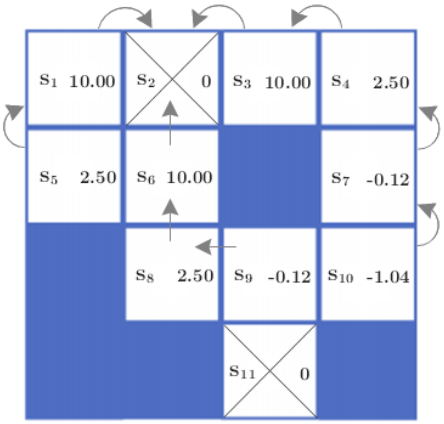

Figure 6. Optimal values and policy for $p = 1$

Furthermore, the values of s_2's surrounding states (s_1, s_3 and s_6) are significantly lower than 10 (the reward for arriving in s_2) due to p. In fact, the relevance of p in the result is so high that it is possible to do the following approximation (being v^* the value obtained for states s_1, s_3 and s_6 when $p = 1$):

$$v^* = r_2 = 10 \qquad V(s_1) = 3.32 \sim V(s_3) = 3.28 \sim V(s_6) = 2.80 \sim p \times v^* = 3.50$$

Being the value of s_6 slightly lower because it is more affected by the state-action value Q (s_9, *south*). The effect of p in s_2's surrounding states can be noticed in Figure 6, above. On the other hand, as it can be noticed in figures 2, 5 and 6, the optimal policy remains the same for different values of p and γ.

Appendix

Question 2's code (universal algorithm with plots):

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# states = {'1' = 0,'2' = 1,...,'11' = 10}
# actions = {'east' = 0,'west' = 1,'north' = 2,'south' = 3}

# The transition matrix is defined as:
# transitionMatrix[actions][current_state,next_state]

transitionMatrix = np.array([[[0,1,0,0,0,0,0,0,0,0,0],[0,0,1,0,0,0,0,0,0,0,0],
                              [0,0,0,1,0,0,0,0,0,0,0],[0,0,0,1,0,0,0,0,0,0,0],
                              [0,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,0,1,0,0,0,0,0],
                              [0,0,0,0,0,0,1,0,0,0,0],[0,0,0,0,0,0,0,0,1,0,0],
                              [0,0,0,0,0,0,0,0,0,1,0],[0,0,0,0,0,0,0,0,0,1,0],
                              [0,0,0,0,0,0,0,0,0,0,1]],
                              [[1,0,0,0,0,0,0,0,0,0,0],[1,0,0,0,0,0,0,0,0,0,0],
                              [0,1,0,0,0,0,0,0,0,0,0],[0,0,1,0,0,0,0,0,0,0,0],
                              [0,0,0,0,1,0,0,0,0,0,0],[0,0,0,0,1,0,0,0,0,0,0],
                              [0,0,0,0,0,0,1,0,0,0,0],[0,0,0,0,0,0,0,1,0,0,0],
                              [0,0,0,0,0,0,0,0,1,0,0],[0,0,0,0,0,0,0,0,1,0,0],
                              [0,0,0,0,0,0,0,0,0,1,0],[0,0,0,0,0,0,0,0,0,1,0],
                              [0,0,0,0,0,0,0,0,0,0,1]],
                              [[1,0,0,0,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0,0,0,0],
                              [0,0,1,0,0,0,0,0,0,0,0],[0,0,0,1,0,0,0,0,0,0,0],
                              [1,0,0,0,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0,0,0,0],
                              [0,0,0,1,0,0,0,0,0,0,0],[0,0,0,0,0,1,0,0,0,0,0],
                              [0,0,0,0,0,0,0,0,1,0,0],[0,0,0,0,0,0,1,0,0,0,0],
                              [0,0,0,0,0,0,0,0,0,1,0],[0,0,0,0,0,0,0,0,0,1,0],
                              [0,0,0,0,0,0,0,0,0,0,1]],
                              [[0,0,0,0,1,0,0,0,0,0,0],[0,0,0,0,0,1,0,0,0,0,0],
                              [0,0,1,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,1,0,0,0,0],
                              [0,0,0,0,1,0,0,0,0,0,0],[0,0,0,0,0,0,0,1,0,0,0],
                              [0,0,0,0,0,0,0,0,0,1,0],[0,0,0,0,0,0,0,0,1,0,0],
                              [0,0,0,0,0,0,0,0,0,0,1],[0,0,0,0,0,0,0,0,0,1,0],
                              [0,0,0,0,0,0,0,0,0,0,1]]])

# Universal algorithm (every CID)

CID = [0,1,8,0,9,2,3,3]

x = CID[5]
y = CID[6]
z = CID[7]

my_j = ((z + 1) % 3) + 1
my_p = 0.25 + 0.5 * (x / 10)
my_discountFactor = 0.2 + 0.5 * (y / 10)

# The rewards are received when ENTERING the state assigned to them

selectReward = [0,0,0,0,0,0,0,0,0,0]
selectReward[(my_j - 1)] = 1

rewardVector = [-1+selectReward[0]*11,-1+selectReward[1]*11,
                -1+selectReward[2]*11,-1+selectReward[3]*11,
                -1+selectReward[4]*11,-1+selectReward[5]*11,
                -1+selectReward[6]*11,-1+selectReward[7]*11,
                -1+selectReward[8]*11,-1+selectReward[9]*11,-100]

discountFactor = 0

valueState9 = []
myValueState = [0,0,0,0,0,0,0,0,0,0,0]
myOptimalPolicy = [0,0,0,0,0,0,0,0,0,0,0]

theta = 0.05 # for evaluation of different 'p' and 'discountFactor'
gamma = 0.001 # threshold for 'delta'

while discountFactor < 1 + theta:

    p = 0
```

```
while p < 1 + theta:

    q = (1 - p) / 3

    # Value iteration algorithm

    # Values of states j and '11' will always be 0, due to the fact that they
    # are terminal states and the reward is collected when we ENTER them

    valueState = [0,0,0,0,0,0,0,0,0,0,0]

    # 'delta' is the threshold to stop the algorithm

    delta = 1

    # 'optimalPolicy' is a vector which contain the optimal action for each state
    # (see 'states' and 'actions')
    # I initialize to be always optimal to move east

    optimalPolicy = [0,0,0,0,0,0,0,0,0,0,0]
    valueStateAux = [0,0,0,0,0,0,0,0,0,0,0]

    while delta > gamma:

        delta = 0

        for i in range(11):

            if i != (my_j - 1) and i != 10: # Values of states j and '11' are known

                valueStateAux[i] = valueState[i]
                x = -1000 # Initial condition (ideally -inf)

                for j in range(4):

                    y = 0

                    # I use auxiliary parameters a, b and c to compute the
                    # stochasticity of actions

                    if j == 0: a = 1; b = 2; c = 3
                    elif j == 1: a = 0; b = 2; c = 3
                    elif j == 2: a = 0; b = 1; c = 3
                    else: a = 0; b = 1; c = 2

                    for k in range(11):

                        y = (y + (p * transitionMatrix[j][i,k] *
                            (rewardVector[k] + discountFactor * valueStateAux[k]))
                            + (q * transitionMatrix[a][i,k] *
                            (rewardVector[k] + discountFactor * valueStateAux[k]))
                            + (q * transitionMatrix[b][i,k] *
                            (rewardVector[k] + discountFactor * valueStateAux[k]))
                            + (q * transitionMatrix[c][i,k] *
                            (rewardVector[k] + discountFactor * valueStateAux[k])))

                    if y >= x: x = y; optimalPolicy[i] = j

                valueState[i] = x

            dif = abs(valueState[i] - valueStateAux[i])

            if delta < dif:

                delta = dif

    if (int((p + 0.001)*100)/100 == my_p) and \
        (int((discountFactor + 0.001)*100)/100 == my_discountFactor):

        # Because of the problem of checking equality with floats

        myValueState = valueState
        myOptimalPolicy = optimalPolicy

    valueState9.append(valueState[8])

    p = p + theta
```

```
    discountFactor = discountFactor + theta

# Plots (for theta = 0.05):

X = [] # discountFactor
Y = [] # p
Z = valueState9

for i in range(21):

    for j in range(21):

        X.append(i/20)
        Y.append(j/20)

font = {'family': 'serif',
        'color':  'blue',
        'weight': 'normal',
        'size': 14,
        }

Axes3D = Axes3D
fig = plt.figure()
ax = plt.axes(projection = '3d')
ax.scatter3D(X, Y, Z, c = Z, cmap = 'Blues');
plt.title('value of state 9', fontdict=font)
plt.xlabel('discount factor', fontdict=font)
plt.ylabel('probability p', fontdict=font)

valueState9_p = []
valueState9_d = []

for i in range(441):

    if X[i] == my_discountFactor:

        valueState9_p.append(Z[i])

    if Y[i] == my_p:

        valueState9_d.append(Z[i])

font = {'family': 'serif',
        'color':  'darkblue',
        'weight': 'normal',
        'size': 14,
        }

plt.figure(figsize=(6,6))
plt.plot([0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,
          0.75,0.8,0.85,0.9,0.95,1],valueState9_p,'darkblue')
plt.title('relationship p - value of state 9', fontdict=font)
plt.xlabel('probability p', fontdict=font)
plt.ylabel('value of state 9', fontdict=font)

font = {'family': 'serif',
        'color':  'darkred',
        'weight': 'normal',
        'size': 14,
        }

plt.figure(figsize=(6,6))
plt.plot([0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,
          0.75,0.8,0.85,0.9,0.95,1],valueState9_d,'darkred')
plt.title('relationship discount factor - value of state 9', fontdict=font)
plt.xlabel('discount factor', fontdict=font)
plt.ylabel('value of state 9', fontdict=font)
```

3D plot of the relationship between $V(s_9)$, p and γ:

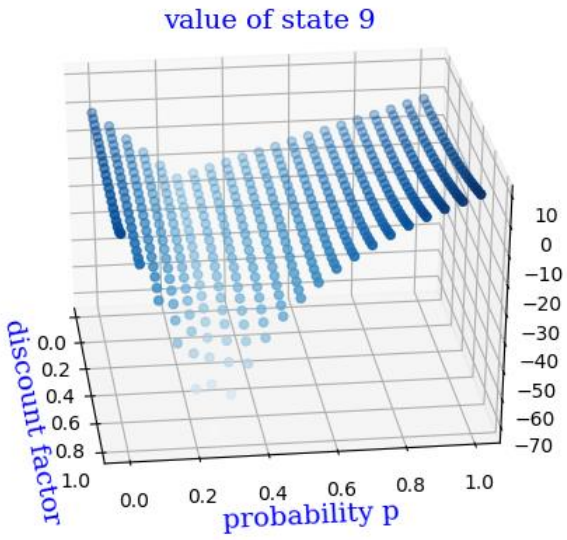

Figure 7. Relationship between $V(s_9)$, p and γ