2018 – 2019

# Question 1

This question is quite morbid when you think about it. (And doesn’t sound legal)

## Part a

We have the following set of states for a given patient:

{Diseased, Survived, Died}

Our set of actions would be the following:

{give patient X, give patient Y, give patient Z}

With two arcs for each coming out of the initial “diseased” state and going to each of the terminal states “Survived” and “Died”. The transition probabilities could initially be inferred from data from the drug trials of each pharmaceutical (assuming they have all been trialed).

The immediate reward could be set to a positive constant N, say 100, for the patient taking an action and going to the” Survived” state and –N for dying.

Note that here we don’t need a discount factor as we do not have any cycles or self-loops in our MDP (we would set it to 1).

Alternative: Every patient responds the same. The state-space can be just {status=Untreated, Alive, Dead}. The action space is {X, Y, Z, change patient}. The transition probabilities we’re trying to infer are P(S=Untreated, S’=Dead, X}, P(S=Untreated, S’=Dead, Y}, P(S=Untreated, S’=Dead, Z}, with the P(S=Untreated, S’= Survived, X, Y, Z) being 1-the others, and every other transition being 0, for actions X, Y, Z. For action change patient we move with P=1 from Dead or Alive to Untreated. The reward function is –1 for each dead patient, 0 if the patient survives or if we change patient, and is dependent only on S and S’, not A. Gamma is 0 I would say, there is no scenario where killing the current patient leads to higher rewards down the line. The greedy policy is choosing the medicine that leads to S=Survived with the highest probability.

*Alternative using counters*: note that the question asks for an MDP with multiple patients. For the Markov property to hold, we need each state to capture ALL information about ALL patients. We can efficiently define the state space by keeping a counter for the number of healed patients (H) and another counter for the number of dead patients (D). Then the state space is:

s\_0: H=0, D=0  
s\_1: H=1, D=0  
s\_2: H=1, D=1  
s\_3: H=1, D=1  
...

Since each state captures all information about all patients in the problem, we have an MDP. Going out from each state we have actions {X, Y, Z}, each of which lead to some other state where either H increases by 1 or D increases by 1.

Btw: Tony has great notes from the Simulation & Modelling course on constructing Markov processes. His advice is that each state is described by the values of the variables you’d use to model this problem in a program: so, you can use int counters, boolean flags, etc.

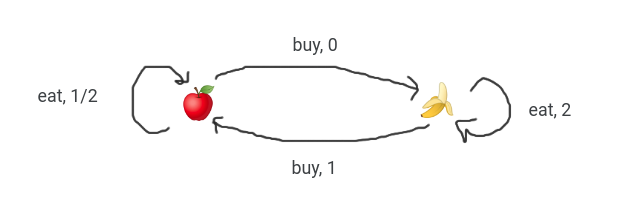
## Part b

Continually select a patient from the population at random and use an epsilon-greedy policy for selecting the policy, initially with epsilon = 1 (random). Take the action and observe the reward. Store the reward for the state action pair Q(Diseased, A) in a list. After a reasonable amount of time (and number of people sampled) average the rewards for each of the state-action values to get an estimate of the reward for taking each action.

An alternative to this is to use TD learning, however the question does not demonstrate any previous knowledge of estimates of the reward and TD learning uses this. Therefor we must use sampling in order to observe the empirical return rather than get a better approximation of the estimated return.

# Question 2

## Part a



From this, we can infer the transition matrix, this is obtained by counting the transitions and normalising the rows to add up to 1

|  |  |  |
| --- | --- | --- |
|  | s' | |
| (s, a) | 🍎 | 🍌 |
| 🍎, eat | 1 | 0 |
| 🍎, buy | 0 | 1 |
| 🍌, eat | 0 | 1 |
| 🍌, buy | 1 | 0 |

and reward matrix

†

## Part b

Kossai Sbai ©

This is how I have solved it.

V\*(🍌) = max(2+0.5\*V\*(🍌), 1+0.5\*V\*(🍎))

V\*(🍎) = max(0.5+0.5\*V\*(🍎), 0.5\*V\*(🍌))

Those two equations above +are obtained applying the Bellman optimality equation on the given MDP, note that the discount factor here is 0.5.

Let’s now have x=V\*(🍌) and y = V\*(🍎)

That gives us the following equations:

x= max(2+0.5\*x, 1+0.5\*y)

y= max(0.5+0.5\*y, 0.5x)

Then we can decompose each max into two equations. That yields us:

x= max

{

2+0.5\*xt

1+0.5\*y

}

And y = max

{

0.5+0.5\*y

0.5x

}

Now we need to find the combination of equations that gives us the max x and y.

When I say combination, I mean pair of equations such that one equation is from the system associated with x and the second one in the system associated with y.

For instance we could have, the resulting following system:

{

x= 2+0.5\*x

y= 0.5+0.5\*y

}

Solving this system would give us x = 4 and y = 1. So repeating this process for all combinations gives us the optimal solution below:

{

x= 2+0.5\*x

y= 0.5\*x

}

Giving us x= 4 and y = 2 and therefore:

V\*(🍎) = 2

V\*(🍌) = 4

What do you guys think of this reasoning?

From the revision lecture:

V(🍎) = 2

V(🍌) = 4

I think.

So, the optimal greedy policy is pi(A) = buy, pi(B) = eat (based on which side of the max expression we picked for each).

## Part c

(Ignore the question typo, confirmed on piazza they just included gamma twice)



Everyone please note, a gamma of 0.5 is used here.

For Q(🍎, eat) our returns are [¾].

For Q(🍎, buy) our returns are [½, 0].

For Q(🍌, eat) our returns are [3].

For Q(🍌, buy) our returns are [1, 1].

So, our table is

|  |  |  |
| --- | --- | --- |
|  | eat | buy |
| 🍎 | 3/4 | 1/4 |
| 🍌 | 3 | 1 |

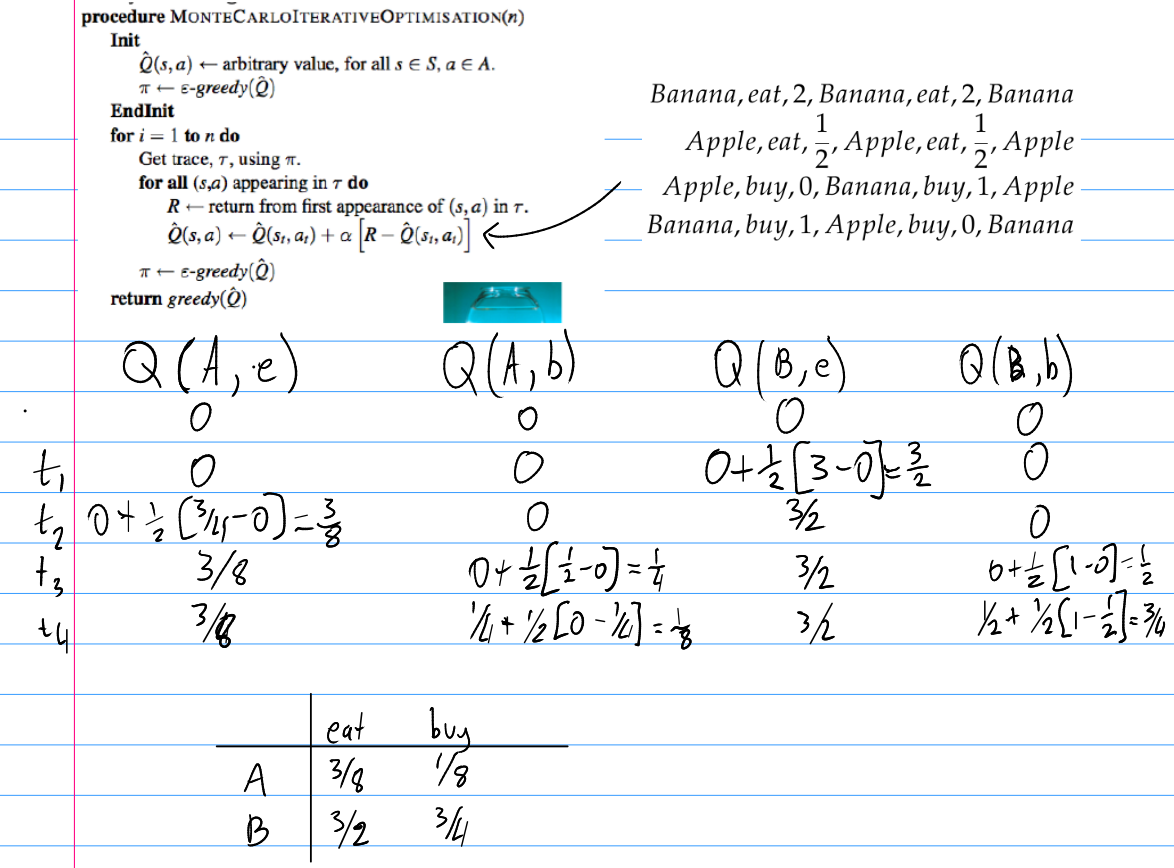
Kossai Sbai ©,

For gamma = 0.5, 100% agree with the above solution got the same results.

Please find below the values I worked out, **with gamma = 1**, let me know what you guys think! Refer to piazza post: <https://piazza.com/class/kf7uhzvqmwa14k?cid=554> for further details.

* (Apple,Buy) has a returns list as follows [0+1,0] = [1,0]
* (Banana,Buy) has a returns list as follows [1,1+0] = [1,1]
* (Apple,Eat) has a returns list as follows [0.5 + 0.5] = [1]
* (Banana,Eat) has a returns list as follows [2+2] = [4]  
  Now if we do the average of each of those lists in order to get the respective Q(s,a) values:
* (Apple,Buy) has a Q(s,a) = (1 + 0) / 2 = 1/2
* (Banana,Buy) has a Q(s,a) = (1 + 1) / 2 = 1
* (Apple,Eat) has a Q(s,a) = 1 / 1 = 1
* (Banana,Eat) has a Q(s,a) = 4/1 = 4

Alternative with Iterative Learning Control using “α”



# Question 3

## Part a

In Policy Iteration we have two loops:

1. Policy evaluation
2. Policy improvement

In Value Iteration we do both at once.

<https://stackoverflow.com/questions/37370015/what-is-the-difference-between-value-iteration-and-policy-iteration>

Adele Wang answer:

* While policy iteration and value iteration both follows the evaluate -> improve -> evaluate cycle, the evaluation step in value iteration is only done once (one complete sweep, one update of every state), whereas the evaluation step in policy iteration is done iteratively until some threshold of accuracy is met. Both will still guarantee convergence to v\* under the same conditions that guarantee the existence of v\*

## Part b

I would define the state space as the state of the board, i.e. for each square what symbol is in that square: O, X or empty and the state would be this for each of the 9 squares.

Note that this creates many permutations of states but it’s not that bad (compared to Go or backgammon which are both feasible on a computer). This problem is suited well for an MCTS approach, however.

The state space might also include the next player to move (i.e. X or O).

Start state has all cells empty. Terminal states would be those where there is a line of Xs, Os, or where there are no more empty squares left.

I would model the policy of both myself and my opponent as purely greedy as there are only a few moves that need to be made in order to win (it is unlikely if the players are experienced that they “try out a new move” because there are only 9 possible actions in total).

## Part c

Cliff walking example: The goal of the agent is to traverse the edge of the cliff in as little steps as possible without falling off.

Since Sarsa is on-policy, it would learn that moving and falling off the cliff yields negative reward and this would cause it to learn a policy that initially moves away from the edge of the cliff and then goes past (essentially, the long but safe way round).

Q-Learning, on the other hand, is an Off-policy method and it would learn from the actions of the behavior policy in order to develop a target policy that takes the “more dangerous” route along the edge of the cliff, completing the task in less steps, and it would learn not to move in the direction that causes it to fall off the edge.