ASSIGNMENT #2

Alireza Mostafizi

I coded MDP Planning Algorithm in Python 2.7. Attached to the email you can find two python codes. One is the algorithm for optimizing expected infinite-horizon discounted cumulative reward (HW2.py), and the other one is a code that you can enter certain parameters and probabilities and it builds the corresponding MDP input to feed the algorithm (Part3 - MDP Maker.py). I will go through the details of the second code in Part III of the homework. Please keep in mind that to have the code running you have to have “NumPy” installed for matrices operations. I am pretty sure that you have that, but in case, I also attached the installer to the email.

Part I: Build a Planner

Please note that as requested, the MDP description input format is the same as HW1. Besides, the discount factor should be entered in the program interface. The output consists two n-dimensional vectors, first for the optimal greedy policy, and the second one is value function associated with the mentioned policy. The ϵ is set to be 0.0000001 to acquire the desired precision, and to approximately get to the optimal policy. In addition, I implemented “value iteration” algorithm for this assignment.

Part II: Run the Planner

MDP 1

Algorithm is implemented on MDP 1, and the results are as following:

Optimal Policy – Beta = 0.9										
	States									
	1	2	3	4	5	6	7	8	9	10
Actions	4	4	3	1	1	1	2	3	2	4
Optimal Policy – Beta = 0.1										
	States									
	1	2	3	4	5	6	7	8	9	10
Actions	4	4	3	1	1	1	2	3	2	4

Value Function (V_g) – Beta = 0.9										
	States									
	1	2	3	4	5	6	7	8	9	10
Values	3.3210	2.9234	2.8914	2.9234	3.6900	2.8407	3.1564	2.9071	2.9889	3.2482
Value Function (V_g) – Beta = 0.1										
	States									
	1	2	3	4	5	6	7	8	9	10
Values	0.1001	0.0090	0.0088	0.0090	1.0010	0.0068	0.0683	0.0086	0.0100	0.0896

MDP 2

Algorithm is implemented on MDP 2, and the results are as following:

Optimal Policy – Beta = 0.9										
	States									
	1	2	3	4	5	6	7	8	9	10
Actions	1	1	1	2	3	3	3	2	2	3
Optimal Policy – Beta = 0.1										
	States									
	1	2	3	4	5	6	7	8	9	10
Actions	4	1	1	2	3	3	3	4	2	3

Value Function (V_g) – Beta = 0.9										
	States									
	1	2	3	4	5	6	7	8	9	10
Values	4.2632	4.2577	5.1885	3.8436	4.4169	4.7368	5.2632	3.8351	3.9752	4.7368
Value Function (V_g) – Beta = 0.1										
	States									
	1	2	3	4	5	6	7	8	9	10
Values	0.0114	0.0100	0.5733	0.0015	0.0604	0.1010	1.0101	0.0080	0.0060	0.1010

In general, since we cannot run the simulation forever, and we have to stop it at a certain point, it is impossible to reach to the exact optimal infinite-horizon policy. However, we can decrease the error to guarantee a certain precision. In this case, based on value of β and ϵ (=0.0000001), and looking at class notes, we can state that:

- If $\left|\left|V^k - V^{k-1}\right|\right| \leq \epsilon$ then $\left|\left|V^k - V^*\right|\right| \leq \frac{\epsilon\beta}{1-\beta}$
- If $\left|\left|V^k - V^*\right|\right| \leq \lambda$ then $\left|\left|V_g - V^*\right|\right| \leq \frac{2\lambda\beta}{1-\beta}$

So here are the bounds which show how suboptimal the mentioned policies are:

$\beta = 0.9$:

- $\left|\left|V^k - V^{k-1}\right|\right| \leq 10^{-7} \quad \rightarrow \quad \left|\left|V^k - V^*\right|\right| \leq 9 * 10^{-7} \quad \rightarrow \quad \left|\left|V_g - V^*\right|\right| \leq 1.62 * 10^{-5}$

$\beta = 0.1$:

- $\left|\left|V^k - V^{k-1}\right|\right| \leq 10^{-7} \quad \rightarrow \quad \left|\left|V^k - V^*\right|\right| \leq 1.1 * 10^{-8} \quad \rightarrow \quad \left|\left|V_g - V^*\right|\right| \leq 2.4 * 10^{-9}$

Part III: Parking Domain

States

Let's start this part with mapping the states in the code to the real states. If we assume n parking spots for each row, the MDP has $8n + 1$ states. Like suggested in the assignment, I considered $8n$ states, plus a final state. This is the final state that you can get to only if the car is parked or had an accident, and it acts as a terminal state.

In the program we have state 1 to state $8n + 1$. To depict how the states are mapped to the numbers, let's assume $n = 5$, and then we have 41 states as following:

	State Numbers																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Cell	A5				A4				A3				A2				A1			
Occupancy	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O
Parked	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P

	State Numbers																			
	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
Cell	B1				B2				B3				B4				B5			
Occupancy	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O
Parked	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P

State 41 is the final state.

NO = NOT OCCUPIED *O = OCCUPIED* *NP= NOT PARKED* *P = PARKED*

Occupancy Probability

There are some simple assumptions I made to prepare the MDP.

- The probability of ADA spot being empty is entered in the program.
- The closer the spot is to the store, the lower the probability of the spot being empty.
    - Therefore, the program asks for the probability of closest spot (A_2 and B_2) being empty (AKA Min probability) and probability of the farthest spot (A_n and B_n) being empty (AKA Max Probability)
    - I assumed that the probability of other spots not being occupied linearly decrease from max to min probability, based on their distance to the store.

Rewards

Here are assumption about the rewards. Most of them are suggested by the assignment.

- There is a big fine for parking in ADA spot, which you can enter in program
- There is a huge cost for accident, which you can enter in program
- There is a relatively small cost for driving from each cell to the next one, which you can enter in the program

- You can enter the reward of parking in the closest spot (A_2 and B_2), known as Max reward, and parking in the farthest spot (A_n and B_n), known as Min reward.
    - The other rewards are drawn from linear interpolation between max reward and min reward, based on their distance to the store.

Transition Matrices

As suggested by the assignment, I considered three actions:

- 1. Drive to the next spot
    - ACTION 1 in the program
    - Cannot be taken when the car is parked
    - Will take you to the next cell considering the circulation pattern
        - Note that there are 4 states for each cell. This action will take you to the next cell's state with NO PARKED condition. Ending up in OCCUPIED or NOT OCCUPIED state is based on the probability of occupancy.
- 2. Park
    - ACTION 2 in the program
    - Cannot be taken when the car is parked
    - Will take you from NOT PARKED state in each cell to the PARKED state in the same cell.
        - The probability of ending up in OCCUPIED or NOT OCCUPIED condition is based on the probability of occupancy.
- 3. Exit
    - ACTION 3 in the program
    - Cannot be taken when the car is not parked
    - And If the car is parked (or had an accident), the next action will definitely be the exit

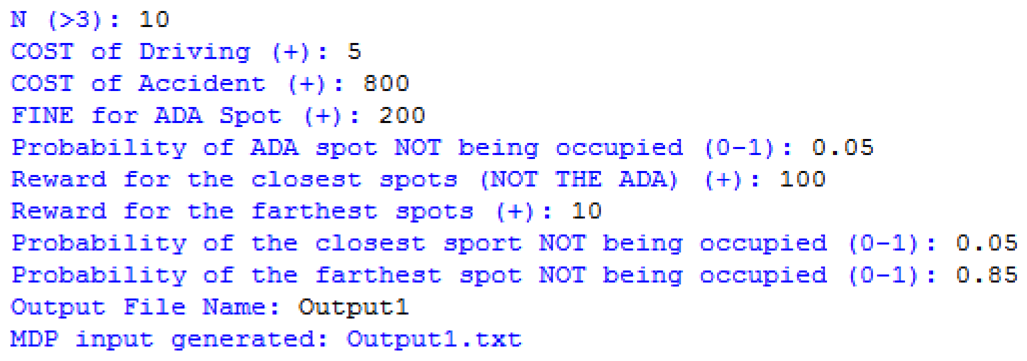
The supplementary program I coded is to take the mentioned probabilities and rewards, and to generate a corresponding MDP input format to feed the algorithm.

In the following pages, I will discuss two different MDPs which came from two different sets of parameters that I used to generate their initial input MDP format.

As noted in the assignment, in all the runs, β is set to be 0.99.

First MDP:

I generated an MDP input format considering the following probabilities and rewards.



```
N (>3): 10
COST of Driving (+): 5
COST of Accident (+): 800
FINE for ADA Spot (+): 200
Probability of ADA spot NOT being occupied (0-1): 0.05
Reward for the closest spots (NOT THE ADA) (+): 100
Reward for the farthest spots (+): 10
Probability of the closest sport NOT being occupied (0-1): 0.05
Probability of the farthest spot NOT being occupied (0-1): 0.85
Output File Name: Output1
MDP input generated: Output1.txt
```

You can find this MDP attached to the email. Or you can simply enter the above numbers into the MDP generator, and generate the file.

The results for optimal policy and value functions ($\beta = 0.99$) are as following.

A10				A9				A8				A7				A6				A5				A4				A3				A2				A1			
N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O
N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P
2	3	2	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3

B1				B2				B3				B4				B5				B6				B7				B8				B9				B10			
N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O	N O	N O	O	O
N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P	N P	P
1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	2	3	2	3

As you can see, in the all PARKED states, action 3, EXIT, should be taken.

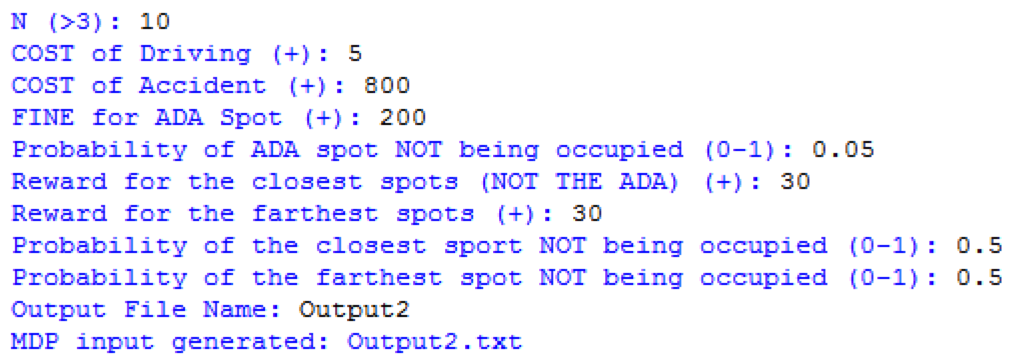
Here are some other intuitions:

- Optimal policy to park is to find a parking sport at the end of the rows since it has a good chance to be empty.
- Also, since driving cost is high with respect to the parking rewards.
- However, it does not make sense since one of them is occupied and there is a high cost for accident.
    - o There could be a bug in the code!

Value functions does not fit in this report. To take look at them, please run the program with output1.txt and check them.

Second MDP:

Now let's take a look at MDP with the same probability for the parking lots to be empty and the same rewards.



```
N (>3): 10
COST of Driving (+): 5
COST of Accident (+): 800
FINE for ADA Spot (+): 200
Probability of ADA spot NOT being occupied (0-1): 0.05
Reward for the closest spots (NOT THE ADA) (+): 30
Reward for the farthest spots (+): 30
Probability of the closest sport NOT being occupied (0-1): 0.5
Probability of the farthest spot NOT being occupied (0-1): 0.5
Output File Name: Output2
MDP input generated: Output2.txt
```

The results for optimal policy and value functions ($\beta = 0.99$) are as following.

A10				A9				A8				A7				A6				A5				A4				A3				A2				A1			
NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O
NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P
2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	1	3	1	3

B1				B2				B3				B4				B5				B6				B7				B8				B9				B10			
NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O	NO	NO	O	O
NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P	NP	P
1	3	1	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3

As you can see, in the all PARKED states, action 3, EXIT, should be taken.

Here are some other intuitions:

- Basically, in this case, when all the probabilities and all the rewards are the same, it is like park whenever you found a place.
- However, I am pretty sure that there is a bug in my code which does not capture the high accident costs. I should dig into that more.

Value functions does not fit in this report. To take look at them, please run the program with output2.txt and check them.