Dynamic Distributed Decision Making
Project 1
MIE567

Hao Tan 999735728
Xiali Wu 999011322
David Molina 1005615318

University of Toronto — February 11, 2020

1 Modelling

First, you are tasked with modelling the Gridworld domain above as a Markov decision process. Then, you are asked to provide a complete programming description of the problem that will be used to solve it computationally.

1 Explain how you would model this navigation problem as a Markov decision process. In particular:

a) Why is this problem an MDP?

This problem can be described as an MDP because it can be modeled in terms of components such as a reward function, actions, transition probability, states, time steps and discount factor. Additionally, this problem is an infinite horizon MDP since we can take unlimited amounts of steps between cells, and possibly re-visit the same cell. In this case, unlimited amount of steps can potentially happen to maximize rewards; therefore, a discount factor is included for future rewards.

Furthermore, based on the definition of Markovian property, a stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it (ie. P[St+1 | St] = P[St+1 | S1, S2 ... St]). In the given problem description, the agent can only move one cell per action, while the next cell is only dependent on the current cell. In other words, the next cell is not dependent on previous cells before the current cell.

b) What are suitable state and action spaces for this problem? Are these the only possible choices? Why or why not?

The states can be represented as the coordinates of the grid world (i,j), where i= 0,..,4, and j=0,...,4, as seen in the following.

$$\text{states} = \begin{bmatrix} (0,0) & (0,1) & (0,2) & (0,3) & (0,4) \\ (1,0) & (1,1) & (1,2) & (1,3) & (1,4) \\ (2,0) & (2,1) & (2,2) & (2,3) & (2,4) \\ (3,0) & (3,1) & (3,2) & (3,3) & (3,4) \\ (4,0) & (4,1) & (4,2) & (4,3) & (4,4) \end{bmatrix} \tag{1}$$

The available set of actions are North, East, West, South, which can be implemented using the addition and subtraction of grid world coordinates. For example, the following shows the possible actions where (-1, 0) means North, (0,1) means East, (1,0) means South, and (0,-1) means West.

$$\text{actions } = [(-1,0),(0,1),(1,0),(0,-1)] \tag{2}$$

There are other alternatives for the representation of this problem, for example, each cells can be represented with a number from 1 to 25 and actions can be represented as letters A = N, E, W, S or up, right, down, left. These type of representations are tidieous to implement. For example, using continuous numbers to represent states would be difficult to stop the agent from moving off-grid without excessive *if-statements*. In terms of other possible actions, one may considerd diagonal actions such as those shown below.

$$\text{diagonal actions } = [(1,1),(-1,1),(1,-1),(-1,-1)] \tag{3}$$

Diagonal actions require the agent to move two steps per action to arrive at the intended cell. For example, to move North-West, the agent must move West then Easy or vice versa. This conflicts with the problem description where the agent is only allowed to move one cell at a time; therefore, this alternative not used in the model.

c) What is the transition probability matrix P? (You may describe just the non-zero entries.)

In a transition probability matrix, the sum of each row is equal to one. In the transition probability matrix presented in the Appendix, the vertical axis along the left represents the current state, s, while the horizontal axis along the top represents the potential next state, s'. Assuming a random policy with equal probability distribution for all actions, there is a 25% chance of arriving at the intended neighbor cell except for the cells along the border of the grid. For the cells at each of the corners, there are 2 actions each that would take the agent off grid. In this scenario, the problem description states that the agent would remain in its current position; therefore, the corner states have a 50% chance of transitioning to its own state and 25% chance of transitioning to a neighbor state. The same logic can be applied for the other border states, where there is always a probability of returning to its own state if an off-grid action is taken. Additionally, there are 2 special states; namely, State A and B where all actions would take the agent directly to A' and B', respectively. The transition probability matrix is presented in the **Appendix**

d) Is the reward function provided the only possible one? If so, explain why. If not, provide an example of a different reward function that would lead to the same optimal behaviour

The reward function privided is by the problem is shown below. In this case, a transition from A to A' and B to B' will yield a reward of 10 and 5, respectively. A transition that would take the agent off grid would result in a reward of -1 while all other transitions yield a reward of zero.

$$\text{Reward } = [0,-1,+5,+10] \tag{4}$$

Using the state and action representation described in previous questions, an off grid action is detected when elements within the next state coordinate holds a value greater or less than the column/row size. For example, the following show two cases of an agent going off grid. In equation (5), the agent is originally in state (0,0) and the chosen action is to the left/West (0, -1), this results in a coordinate (0, -1) that featured an element smaller than the grid column/row size. Conversely, in equation (6), the agent took an action going to the right when in state (0, 4) which resulted in the next state, (0, 5), with an element larger than the row/column size. In these cases, the agent receives a reward of 1.

By having a different reward function, the overall optimization problem is different. As a result, this is the only reward function that leads to optimal behaviour as described in the problem description.

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} 0 \\ 4 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \end{bmatrix} \tag{6}$$

e) Derive the discounted Bellman equation for the problem, and simplify as much as you can. (Hint: to avoid deriving a separate value for each state, try to find groups of states such that you can write a single expression for V for them) What do you think is/are the optimal policy/policies for this problem, and why (you do NOT need to solve the Bellman equations)?

To write the Bellman equations for this problem, we have grouped together states with similar value functions. The different groups are shown in color in the following figure.

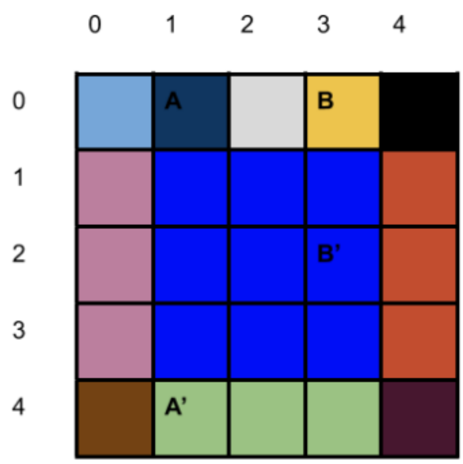


Figure 1: Groups of Bellman Equations

The following shows the Bellman equation.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p\left(s', r|s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right] \tag{7}$$

The following represents the value function for state (0, 0)

$$\begin{gathered} P([0,-1]|(i,j))(-1+\gamma v(i,j)) + P([-1,0]|(i,j))(-1+\gamma v(i,j)) \\ +P([0,+1](i,j))(0+\gamma v(i,j+1)) + P([+1,0]|(i,j))(0+\gamma v(i+1,j)) \\ \text{for } i \in \{0\}, j \in \{0\} \end{gathered} \tag{8}$$

The following represents the value function for state (0, 1)

$$\begin{gathered} P([0,-1]|(i,j))(10+\gamma v(i+4,j)) + P([-1,0]|(i,j))(10+\gamma v(i+4,j)) \\ +P([0,+1]|(i,j))(10+\gamma v(i+4,j)) + P([+1,0]|(i,j))(10+\gamma v(i+4,j)) \\ \text{for } i \in \{0\}, j \in \{1\} \end{gathered} \tag{9}$$

The following represents the value function for state (0, 2)

$$
\begin{gathered}
P([0,-1]|(i,j))(0+\gamma v(i,j-1)) + P([-1,0]|(i,j))(-1+\gamma v(i,j)) \\
+P([0,+1]|(i,j))(0+\gamma v(i,j+1)) + P([+1,0]|(i,j))(0+\gamma v(i+1,j)) \\
\text{for } i \in \{0\}, j \in \{2\}
\end{gathered} \tag{10}
$$

The following represents the value function for state (0, 3)

$$
\begin{gathered}
P([0,-1]|(i,j))(5+\gamma v(i+2,j)) + P([-1,0]|(i,j))(5+\gamma v(i+2,j)) \\
+P([0,+1]|(i,j))(5+\gamma v(i+2,j)) + P([+1,0]|(i,j))(5+\gamma v(i+2,j)) \\
\text{for } i \in \{0\}, j \in \{3\}
\end{gathered} \tag{11}
$$

The following represents the value function for state (0, 4)

$$
\begin{gathered}
P([0,-1]|(i,j))(0+\gamma v(i,j-1)) + P([-1,0]|(i,j))(-1+\gamma v(i,j)) \\
+P([0,+1]|(i,j))(-1+\gamma v(i,j)) + P([+1,0]|(i,j))(0+\gamma v(i+1,j)) \\
\text{for } i \in \{0\}, j \in \{4\}
\end{gathered} \tag{12}
$$

The following represents the value function for state (1,0), (2,0), (3,0).

$$
\begin{gathered}
P([0,-1]|(i,j))(-1+\gamma v(i,j)) + P([-1,0]|(i,j))(0+\gamma v(i-1,j)) \\
+P([0,+1]|(i,j))(0+\gamma(i,j+1)) + P([+1,0]|(i,j))(0+\gamma v(i+1,j)) \\
\text{for } i \in \{1,2,3\}, j \in \{0\}
\end{gathered} \tag{13}
$$

The following represents the value function for state (1,1), (1,2), (1,3), (2,1),(2,2),(2,3), (3,1),(3,2),(3,3).

$$
\begin{gathered}
P([0,-1]|(i,j))(0+\gamma v(i,j-1)) + P([-1,0]|(i,j))(0+\gamma v(i-1,j)) \\
+P([0,+1]|(i,j))(0+\gamma v(i,j+1)) + P([+1,0]|(i,j))(0+\gamma v(i+1,j)) \\
\text{for } i \in \{1,2,3\}, j \in \{1,2,3\}
\end{gathered} \tag{14}
$$

The following represents the value function for state (1,4),(2,4),(3,4).

$$
\begin{gathered}
P([0,-1]|(i,j))(0+\gamma v(i,j-1)) + P([-1,0]|(i,j))(0+\gamma v(i-1,j)) \\
+P([0,+1]|(i,j))(-1+\gamma v(i,j)) + P([+1,0]|(i,j))(0+\gamma v(i+1,j)) \\
\text{for } i \in \{1,2,3\}, j \in \{4\}
\end{gathered} \tag{15}
$$

The following represents the value function for state (4,0).

$$
\begin{gathered}
P([0,-1]|(i,j))(-1+\gamma v(i,j)) + P([-1,0]|(i,j))(0+\gamma v(i-1,j)) \\
+P([0,+1]|(i,j))(0+\gamma v(i,j+1)) + P([+1,0]|(i,j)(-1+\gamma v(i,j)) \\
\text{for } i \in \{4\}, j \in \{0\}
\end{gathered} \tag{16}
$$

The following represents the value function for state (4,1), (4,2), (4,3).

$$
\begin{gathered}
P([0,-1]|(i,j))(0+\gamma v(i,j-1)) + P([-1,0]|(i,j))(0+\gamma v(i-1,j)) \\
+P([0,+1]|(i,j))(0+\gamma v(i,j+1)) + P([+1,0]|(i,j))(-1+\gamma v(i,j)) \\
\text{for } i \in \{4\}, j \in \{1,2,3\}
\end{gathered} \tag{17}
$$

The following represents the value function for state (4,4).

$$
\begin{gathered}
P([0,-1]|(i,j))(0+\gamma v(i,j-1)) + P([-1,0]|(i,j))(0+\gamma v(i-1,j)) \\
+P([0,+1]|(i,j))(1+\gamma v(i,j)) + P([+1,0]|(i,j))(-1+\gamma v(i,j)) \\
\text{for } i \in \{4\}, j \in \{4\}
\end{gathered} \tag{18}
$$

2 Now, in a Python file called Gridworld.py, create a class that replicates the behaviour of the MDP you formulated in the previous question. Your class should contain four functions: one to return the initial state of the MDP, one to return a view of all possible states, and two to return, respectively, the reward and probability of a transition (s; a; s') from state s to state s' when taking action a.

Listing 1: Returns a random initial state

```
def initial_state(self):
    # randomly generate an initial state
    i = random.randint(0, len(self.states)-1)
    rand_state = self.states[i]
    return rand_state
```

Listing 2: Returns a view of all possile states

```
def possible_states(self):
    # return the possible states
    return self.states
```

Listing 3: Returns probability of transition from s to s' given a

```
def transition_probability(self, current_pos, new_pos):
    # deterministic environment = s + a has a 100% probability of ending up in s'
    return 1
```

Listing 4: Returns a reward given current position and action

```
def reward(self, current_pos, action):
    # take action in current pos
    self.new_pos = np.array(current_pos) + np.array(action)
    # normally, reward = 0
    reward = 0
    # if new pos results in off the grid, return reward -1
    if -1 in self.new_pos or self.size in self.new_pos:
        reward = -1
    # if in state A, transition to state A'
    if current_pos == [0, 1]:
        reward = 10
    # if in state B, transition to state B'
    if current_pos == [0, 3]:
        reward = 5
    return reward
```

2 Policy Evaluation

Now, suppose the agent selects all four actions with equal probability. Use your answers to questions 1 and 2 to write a python function, in a new file policy evaluation.py to find the value function for this policy. You may use either an iterative method or solve the system of equations. Show the value function you obtained to at least four decimals.

Following value function is generated based on equal action probability, and = 0.99. The value function found using an iterative method is as follow:

```
Value Function:
[[ 3.2175   7.1243   4.3031   4.7526   1.5245]
 [ 1.4609   2.7247   2.2163   1.7565   0.3782]
 [-0.4904   0.2073   0.1705  -0.2499 -1.1211]
 [-2.1493  -1.5671 -1.4849  -1.8157 -2.5267]
 [-3.467   -2.9047 -2.7873  -3.0748 -3.7354]]
```

Figure 2: Value Function using Policy Evaluation

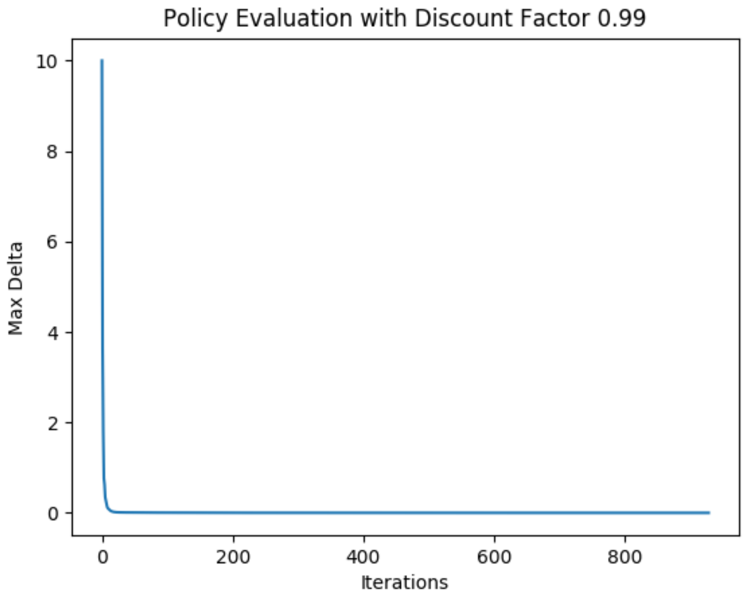


Figure 3: Convergence of Policy Evaluation

3 Value Iteration

The goal now is to solve the MDP above using dynamic programming. In a separate file called value iteration.py, provide a complete implementation of the value iteration algorithm you learned in class for solving the Bellman equations you derived earlier.

- Did your algorithm converge at all?

- How many iterations did this take for each value of γ ?

- What is the best value of γ? Also, What was the final policy you obtained for each value of γ?

- Does the optimal policy you obtained correspond to the policy you conjectured earlier?

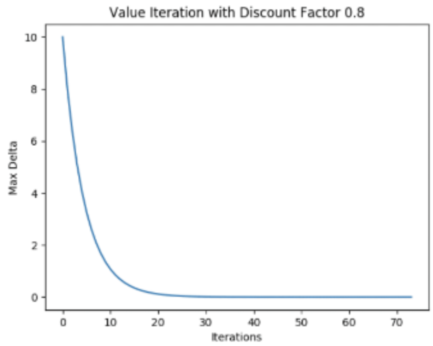
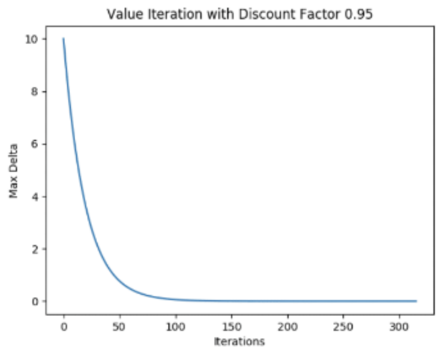
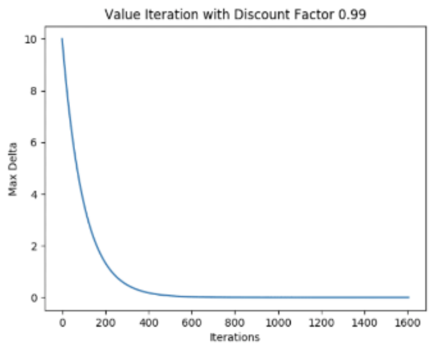
γ	Delta Function as number of iterations increase	Number of Iterations	Optimal Policy
0.8		74	0 1 2 3 4 0 right up left up left 1 up up up up up 2 up up up up up 3 up up up up up 4 up up up up up
0.95		316	0 1 2 3 4 0 right up left up left 1 up up up left left 2 up up up up up 3 up up up up up 4 up up up up up
0.99		1605	0 1 2 3 4 0 right up left up left 1 up up up left left 2 up up up up up 3 up up up up up 4 up up up up up

Figure 4: Value Iteration results

4 Policy Iteration

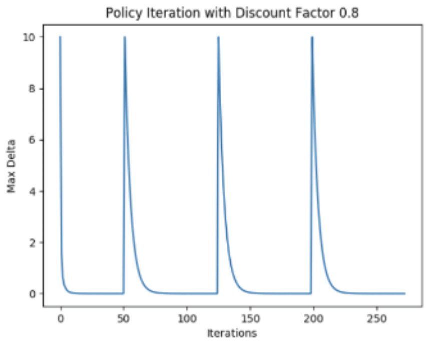
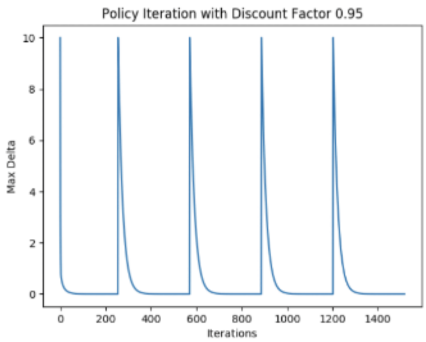
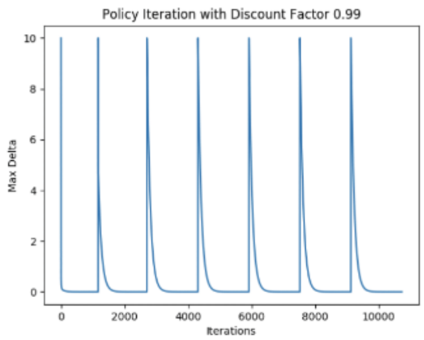
Repeat the previous part of the assignment, but now implement the policy iteration algorithm from class in a file called policy iteration.py. Report your results and answer all questions as in the previous part.

- Did your algorithm converge at all?

- How many iterations did this take for each value of γ ?

- What is the best value of γ? Also, What was the final policy you obtained for each value of γ?

- Does the optimal policy you obtained correspond to the policy you conjectured earlier?

γ	Delta Function as number of iterations increase	Number of Iterations	Optimal Policy
0.8		192 (118)	0 1 2 3 4 0 right up left up left 1 up up up up up 2 up up up up up 3 up up up up up 4 up up up up up
0.95		1447 (1131)	0 1 2 3 4 0 right up left up left 1 up up up left left 2 up up up up up 3 up up up up up 4 up up up up up
0.99		7349 (5744)	0 1 2 3 4 0 right up left up left 1 up up up left left 2 up up up up up 3 up up up up up 4 up up up up up

*Numbers in the bracket is the Number of Iterations that the policy has stopped changing.

Figure 5: Policy Iteration results

5 Comparison of Algorithms

In the final section of your report, you must compare the two algorithms you implemented and their performance on the Gridworld domain, and comment on any differences you observed. In particular, please answer at least the following questions in your report:

- Compare the performance (e.g. values) of the optimal policies obtained using value and policy iteration to the performance of the policy that chooses an action at random (as you analyzed earlier), and comment on the difference.

- Were the value functions and policies you obtained, and the number of iterations required to obtain these policies, similar between algorithms? How do your results differ for different values of the parameter(s) (e.g. γ)?

- Which algorithm was more difficult to implement and why? Which algorithm do you think would work better if the problem was scaled up?

- Include any additional insights, challenges, or important observations that you discovered while building or running your experiments.

6 Appendix

S/S'	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)
(0,0)	0,5	0,25				0,25																			
(1,0)	0,25					0.25	0.25				0.25														
(2,0)						0.25					0.25	0.25				0.25									
(3,0)											0.25					0.25	0.25				0.25				
(4,0)																0.5					0.25	0.25			
(0,1)																						1			
(1,1)		0.25				0.25		0.25				0.25													
(2,1)							0.25				0.25		0.25				0.25								
(3,1)												0.25				0.25		0.25				0.25			
(4,1)																	0.25				0.25	0.25	0.25		
(0,2)		0.25	0.25	0.25				0.25																	
(1,2)			0.25				0.25		0.25				0.25												
(2,2)								0.25				0.25		0.25				0.25							
(3,2)													0.25				0.25		0.25				0.25		
(4,2)																		0.25				0.25	0.25	0.25	
(0,3)														1											
(1,3)				0.25				0.25		0.25				0.25											
(2,3)									0.25				0.25		0.25				0.25						
(3,3)														0.25				0.25		0.25				0.25	
(4,3)																			0.25				0.25	0.25	0.25
(0,4)				0.25	0.5					0.25															
(1,4)					0.25				0.25	0.25					0.25										
(2,4)										0.25				0.25	0.25					0.25					
(3,4)															0.25				0.25	0.25					0.25
(4,4)																				0.25				0.25	0.5
S/S'	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

Figure 6: Transition Matrix