Dynamic Distributed Decision Making
Project 1
MIE567

Hao Tan 999735728
Xiali Wu 999011322
David Molina

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 modeled as an MDP because it has a discrete state space (cells) and a set of actions (north, east, west, south)/(up, right, down, left), where the transition from one state to another is based on a given action.

We can model this problem in terms of components such as a reward function,actions, transition probability, states, time steps, discount factor. All these components describe a MDP type of question. This problem is infinite horizon MDP since we can take unlimited amounts of steps between cells, and possibly revisit the same cell. Unlimited amount steps are definitely not optimal , but can potentially happen to max reward. Therefore, for infinite horizon MDP, a discount factor accounts for future reward.

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, since the agent can only move one cell for one action, the next cell is only dependent on the current cell; the next cell is not depending on all previous cell 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 (i,j), where i= 0,..,4, and j=0,...,4. Then, the set of states is State = [(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)	(0,1)	(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 1: States represented in coordinate format

The available set of actions are north, east, west, south, that can be represented as adding or subtracting to the respective coordinate, i.e Action = [0, 1], [0, -1], [1, 0], [-1, 0]. [0,-1] means going west; [-1,0] means going north; [0,1] means going east; [0,-1] means going south.

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. This type of action representation is difficult to implement in mathematical calculations and representation. We can't identify the situation of off-the-grid when moving one action.

We decided to represent it as a coordinate to facilitate coding, mathematical calculations and visual representation of the outputs for this project.

Diagonal action = [1, 1], [-1, -1], [1,-1], [-1,1] Diagonal action requires both coordinates to move. Therefore with the constraint of moving one cell in one direction at a time, diagonal actions require moving two directions in order to complete one action at a time, given only compass direction of north, east, west, and south. This conflicts with the problem description, and therefore not used in the model.

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

Assuming a policy where each four actions has the same probability to be chosen, then the transition probability matrix is 0.25 for all four cells surrounding the current cell. For example, the current cell is (0,0); there is 50 % chances that it will go off the grid; there is 25 % chances of moving to cell(0,1); 25% chances of moving to cell (1,0).

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 = 0,-1,+5,+10 is the only possible one given the problem description. This is the only reward function that leads to the optimal behaviour solved below. To satisfy this reward forward, we are assuming that when the current state is at the edge, taking East or West direction meaning the next state is off the grid. For example, the current state is (0,4), taking East direction leads to off-the-grid. However, without the state assumption, taking East or West direction can mean that the next state is the last cell in the above row, or the first cell in the next row. Take the same current state (0,4) as an example, then taking East direction leads to (1,0).

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 2: Transition Matrix

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)?

Since the goal is maximizing reward, optimal policy always looks for A or B to get +10 or +5 reward. If the initial state is away from row 0 where A and B are located, such as row 4, the optimal policy requires more actions to get to the vicinity of A or B. If the initial state is in the vicinity of A and B, there is a tradeoff between more actions towards A and earning discounted +5 more reward from A->A' compared to B->B'.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p\left(s', r|s, a\right) \left[r + \gamma v_\pi\left(s'\right)\right] \tag{1}$$

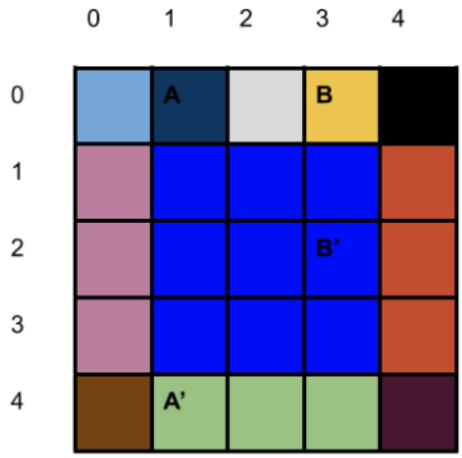

Figure 3: Groups of Bellman Equations

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: Function 1:

```
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: Function 2:

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

Listing 3: Function 3:

```
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 4: Value Function using Policy Evaluation

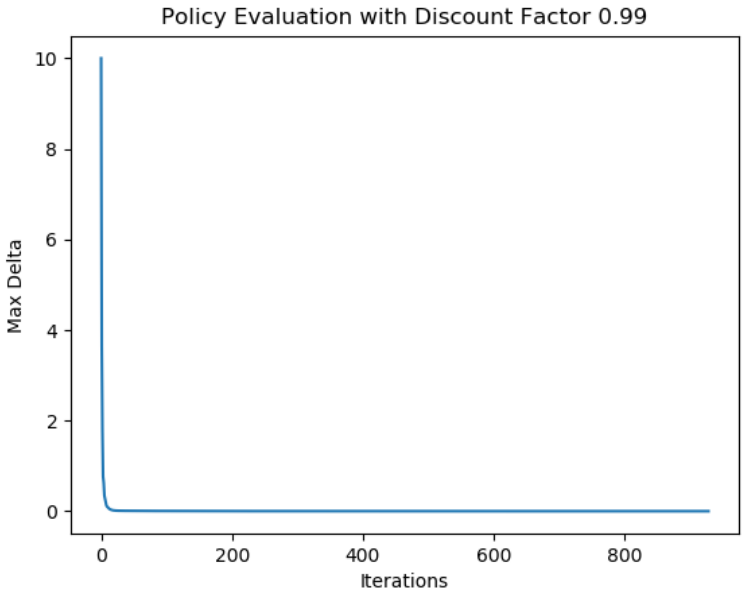

Figure 5: Convergence of Policy Evaluation

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 ?

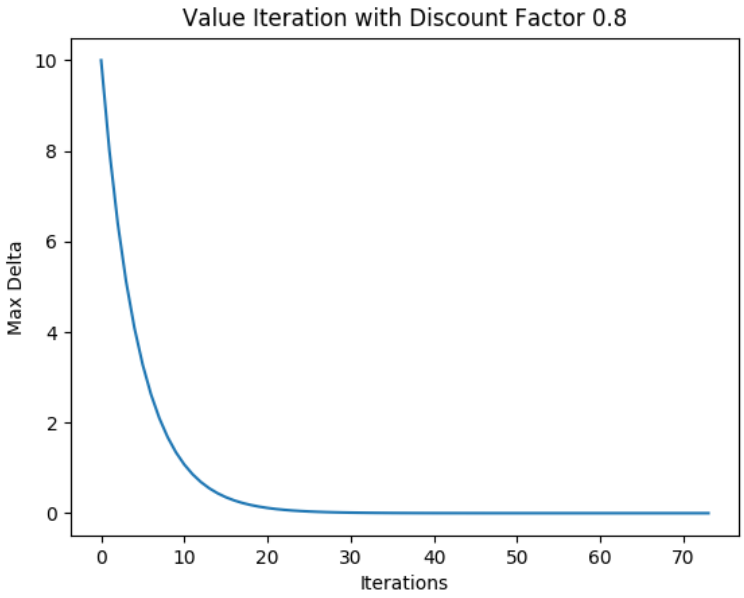

Figure 6: Convergence of Policy Evaluation

- 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?

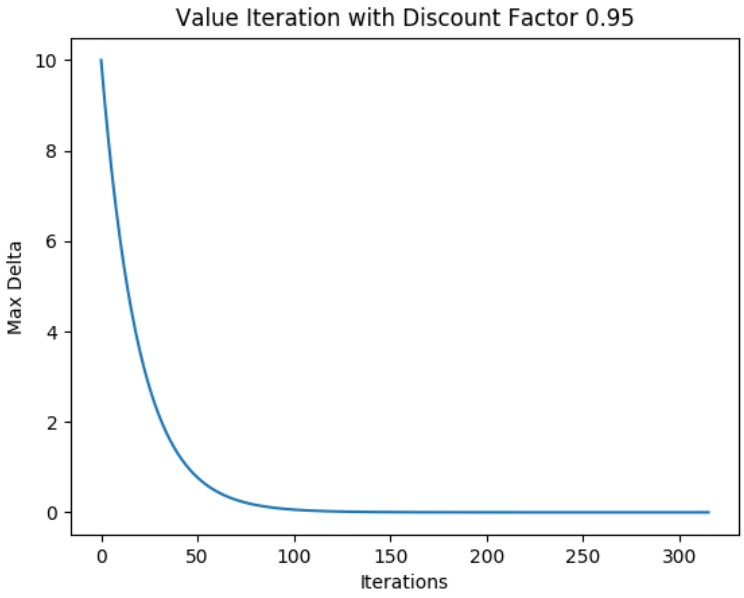

Figure 7: Convergence of Policy Evaluation

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?

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)?

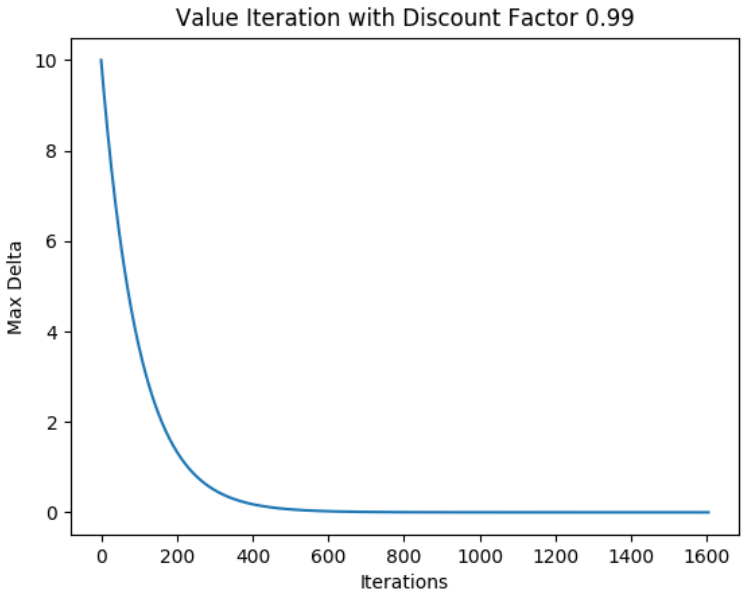

Figure 8: Convergence of Policy Evaluation

- 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.