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 provided 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. As a side note, multiples of the reward function would give the same value functions and optimal policy since the proportionality remains the same.

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

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

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 as represented by the different colors in the figure below. More specifically, they are grouped based on the term $r + \gamma v(s')$; therefore, each group has the same pattern in the combination of r and s' for given action.

Since the goal is maximizing reward, optimal policy always looks for A or B to gain +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 would direct the agent 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'.

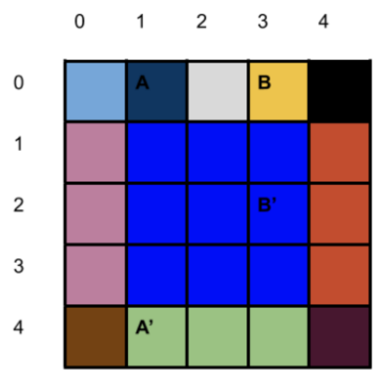


Figure 1: Groups of Bellman equations

The following shows the Bellman equation, while equation (8) to (18) shows the different equations grouped by the colors presented in the above figure.

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

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 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
```

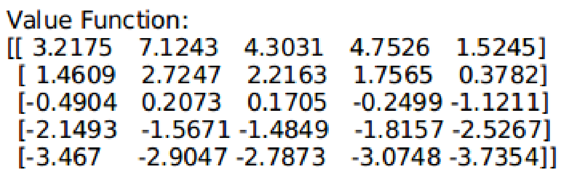
Listing 4: Returns the next state, s', given a in state s

```
    def p_transition(self, current_pos, action):
        self.new_pos = np.array(current_pos) + np.array(action)
        # if taking an action crosses the border = agent stays in same position
        if -1 in self.new_pos or self.size in self.new_pos:
            self.new_pos = current_pos
        # if in state A, transition to state A'
        if current_pos == [0, 1]:
            self.new_pos = [4, 1]
        # if in state B, transition to state B'
        if current_pos == [0, 3]:
            self.new_pos = [2, 3]
        return self.new_pos
```

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.

Using an iterative method, the following figure shows the value function achieved with a policy that features even probability for all actions. From this, it is seen that state A and B featured the highest values at 7.1243 and 4.7526, respectively, which makese sense as actions from these two states would yield a return of +10 and +5. The states towards the bottom of the grid featured a negative value as they are far from the high rewarding states mentioned previously. Additionally, states (0,0) and (0,2) have different values despite being both one cell away from state A because state (0,0) has a higher chance of going off-grid than state (0,2).



```
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

The following shows a graph of the change in max delta over iterations. From this, it is seen that the policy evaluation algorithm was able to converge in roughly 900 iterations with a threshold of 10^{-6}.

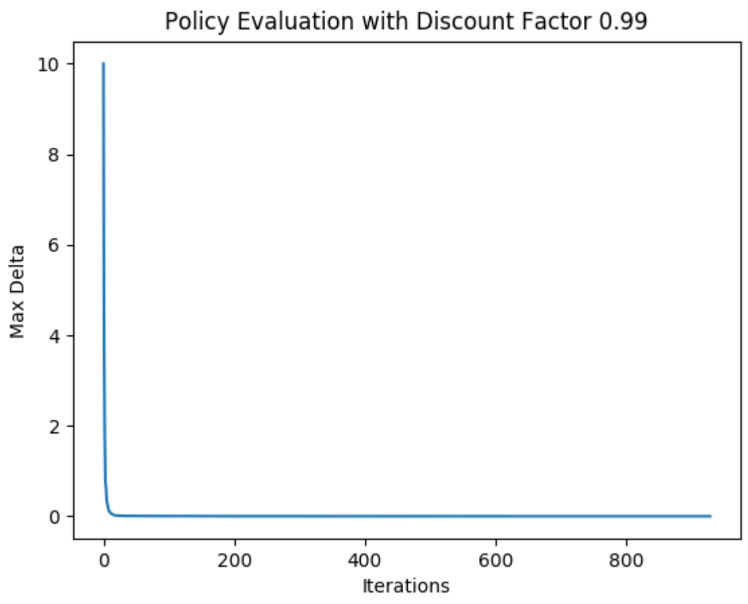


Figure 3: Convergence of policy evaluation

3 Value Iteration

In a separate file called value_iteration.py, provide a complete implementation of the value iteration.

- Did your algorithm converge at all?

In this work, the value iteration algorithm was used to find the optimal policy at 3 different discount factors; namely, 0.8, 0.95 and 0.99. In our case, the value iteration algorithm converged at all discount factors with a threshold of 10^{-6} as described in the problem.

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

As seen in figure 4, different discount factors resulted in different numbers of total iterations necessary to converge. In our case, a discount factor of 0.8, 0.95 and 0.99 have a total iterations of 74, 316, and 1605, respectively. In the left column, the convergence graph shows that the max delta decreases exponentially over the aforementioned interations to a value that is ultimately within the acceptable threshold.

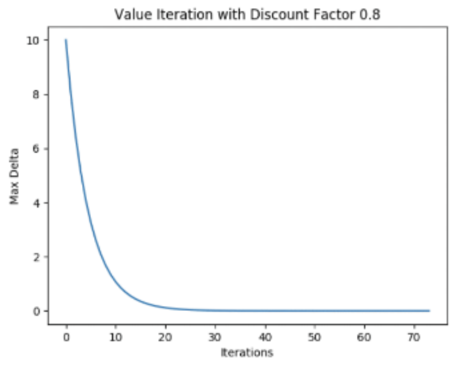
- What is the best value of γ? Also, What was the final policy you obtained for each value of γ?

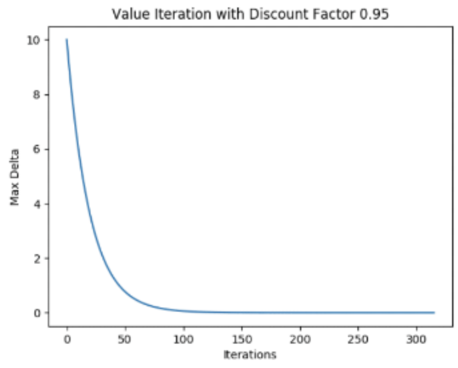
In terms of iterations, the best discount factor is 0.8 as it converged the fastest. In terms of policy that favors long term rewards, 0.95 is better than 0.99 as both achieved the same optimal policy while 0.95 took only 20% of the time it took 0.99 to converge. While the policy is the same in most states, the main difference between the optimal policy with 0.8 discount factor and 0.95/0.99 discount factor are states (1,3) and (1,4). In discount factor 0.8, the policy outputs "up" in these states because it is favoring short term rewards by reaching state B (0,3) for a +5 reward. Conversely, the agent favors long term rewards for discount factor 0.95/0.99; therefore, the policy outputs "left" in the aforementioned states, helping the agent to skip state B and reach state A. The full policy for each discount factor can be seen in figure 4.

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

Yes, it does.

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`

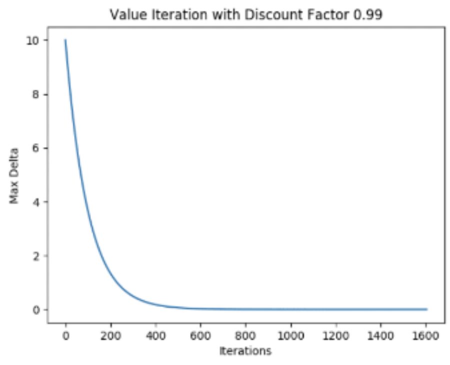
0.99		1605	`   0  1  2  3  4`
			`0 right up left up left`
			`1 up up up left left`
			`2 up up up up up`
			`3 up up up up up`
			`4 up up up up up`

Figure 4: Value iteration results: convergence graph, total iterations, optimal policy

4 Policy Iteration

Implement the policy iteration algorithm in a file called policy_iteration.py.

- Did your algorithm converge at all?

Similar to value iteration, policy iteration was implemented in this work and tested with the same 3 discount factors, 0.8, 0.95 and 0.99. In our case, all three discount factors converged. The convergence graph is seen in the left column of figure 5. Because policy iteration consists of 2 steps: policy evaluation and policy improvement, the evaluation part first converges on a value function and then the improvement step extracts a policy from the value function. The algorithm then takes this policy to create a new value function that is then used to extract a new policy. This process repeats until the policy becomes stable. Because of this two step process. the convergence graph shows different spikes which represents the iteration where a new policy is being evaluated.

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

Different discount factor values led to different numbers of iterations for convergence. As seen in figure 5, discount factors of 0.8, 0.95 and 0.99 have total iterations of 192, 1447 and 7349, respectively. It is also worth noting that the policy often converges faster than the values as shown in the brackets in the second column of figure 5. In this case, the policy converged in 118, 1131 and 5744 iterations for discount factors of 0.8, 0.95 and 0.99 respectively.

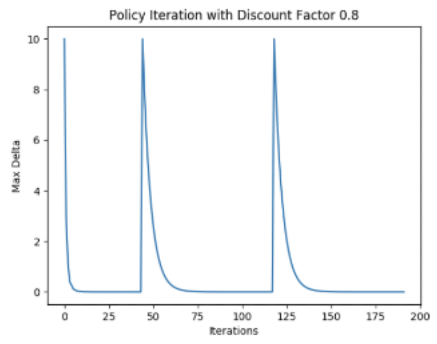
- What is the best value of γ? Also, What was the final policy you obtained for each value of γ?

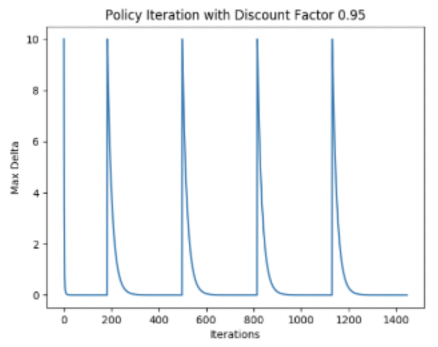
Similar to value iteration, the best discount factor value in terms of number of iterations is 0.8 as it converged the fastest. In terms of favoring future rewards, similar behaviour to value iteration is seen here as well where 0.95 converged much faster than 0.99 while the final optimal policy between the two remain identical. The final optimal policy for each discount factor is presented in figure 5.

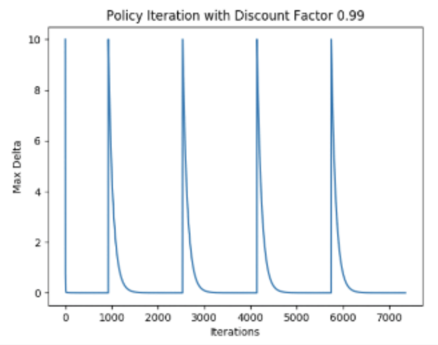
- Does the optimal policy you obtained correspond to the policy you conjectured earlier?

Yes, it does.

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`

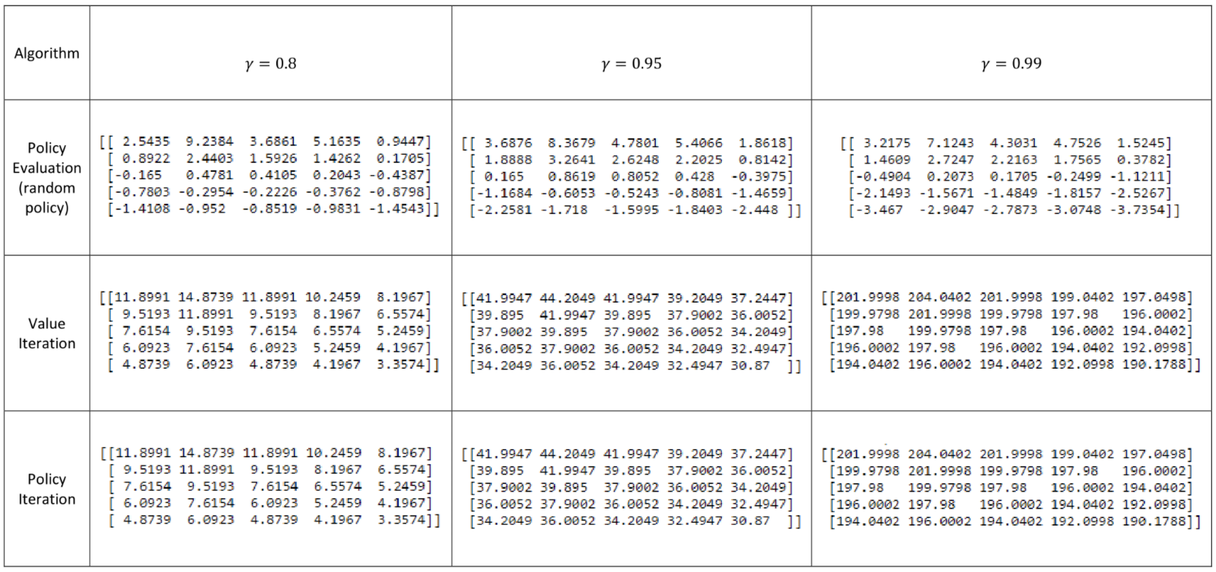
*Numbers in the bracket is the Number of Iterations that the policy has stopped changing.

Figure 5: Policy iteration results: convergence graph, total iterations, optimal policy

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:

The following shows a table of the different value functions achieved with each algorithm at 3 different discount factors.



Algorithm	$\gamma = 0.8$	$\gamma = 0.95$	$\gamma = 0.99$
Policy Evaluation (random policy)	`[[ 2.5435 9.2384 3.6861 5.1635 0.9447] [ 0.8922 2.4403 1.5926 1.4262 0.1705] [-0.165 0.4781 0.4105 0.2043 -0.4387] [-0.7803 -0.2954 -0.2226 -0.3762 -0.8798] [-1.4108 -0.952 -0.8519 -0.9831 -1.4543]]`	`[[ 3.6876 8.3679 4.7801 5.4066 1.8618] [ 1.8888 3.2641 2.6248 2.2025 0.8142] [ 0.165 0.8619 0.8052 0.428 -0.3975] [-1.1684 -0.6053 -0.5243 -0.8081 -1.4659] [-2.2581 -1.718 -1.5995 -1.8403 -2.448 ]]`	`[[ 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]]`
Value Iteration	`[[11.8991 14.8739 11.8991 10.2459 8.1967] [ 9.5193 11.8991 9.5193 8.1967 6.5574] [ 7.6154 9.5193 7.6154 6.5574 5.2459] [ 6.0923 7.6154 6.0923 5.2459 4.1967] [ 4.8739 6.0923 4.8739 4.1967 3.3574]]`	`[[41.9947 44.2049 41.9947 39.2049 37.2447] [39.895 41.9947 39.895 37.9002 36.0052] [37.9002 39.895 37.9002 36.0052 34.2049] [36.0052 37.9002 36.0052 34.2049 32.4947] [34.2049 36.0052 34.2049 32.4947 30.87 ]]`	`[[201.9998 204.0402 201.9998 199.0402 197.0498] [199.9798 201.9998 199.9798 197.98 196.0002] [197.98 199.9798 197.98 196.0002 194.0402] [196.0002 197.98 196.0002 194.0402 192.0998] [194.0402 196.0002 194.0402 192.0998 190.1788]]`
Policy Iteration	`[[11.8991 14.8739 11.8991 10.2459 8.1967] [ 9.5193 11.8991 9.5193 8.1967 6.5574] [ 7.6154 9.5193 7.6154 6.5574 5.2459] [ 6.0923 7.6154 6.0923 5.2459 4.1967] [ 4.8739 6.0923 4.8739 4.1967 3.3574]]`	`[[41.9947 44.2049 41.9947 39.2049 37.2447] [39.895 41.9947 39.895 37.9002 36.0052] [37.9002 39.895 37.9002 36.0052 34.2049] [36.0052 37.9002 36.0052 34.2049 32.4947] [34.2049 36.0052 34.2049 32.4947 30.87 ]]`	`[[201.9998 204.0402 201.9998 199.0402 197.0498] [199.9798 201.9998 199.9798 197.98 196.0002] [197.98 199.9798 197.98 196.0002 194.0402] [196.0002 197.98 196.0002 194.0402 192.0998] [194.0402 196.0002 194.0402 192.0998 190.1788]]`

Figure 6: Value functions of policy evaluation, value iteration and policy iteration

- 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.

The value functions obtained from policy and value iteration are identical for each state across all discount factors as seen in figure 6. Both algorithms provide larger values than the random policy evaluation, and the difference increases as discount factor increases. The reason for this is because during policy evaluation, the algoithm finishes when the value of each state converges according to the specific policy. There is no improvement step that would update the policy; therefore, the performance of the policy as demonstrated by the values in policy evaluation in figure 6 will always be smaller than those from policy and value iteration.

- 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)?

We obtained the same value functions and policies from both value and policy iteration. Our results showed that value iteration required less number of iterations than policy iteration. For example, when discount factor is 0.99, the number of iterations of policy iteration is 7349 and value iteration is 1605; when discount factor is 0.95, the number of iterations of policy iteration is 1447 and value iteration is 316; when discount factor is 0.80, the number of iterations of policy iteration is 192 and value iteration is 74. These results showed that within the same algorithm, the number of iterations to converge decreases as the discount factor decreases. This is because as the discount factor decreases, so does the agent's consideration of future rewards. This way, the value function is updated less at each time step which leads to faster convergence.

The extra iteration count during policy iteration is due to policy evaluation. Each policy evaluation starts with the value function obtained with the previous policy. As shown in the delta function graphs in figure 5, the iterations between two local max deltas represent the policy evaluation steps where the algorithm tries to converge on the value function while the local max delta appears when there is a new policy. Since the initial policy is randomly generated, we tried different runs and we found that starting from different initial policy will change the total number of iteration, but it won't change the optimal states values and the optimal policy.

Value iterations have less number of iterations because the algorithm truncates policy evaluation. The designed algorithm updates all values of cells based on the previous value functions, and progressively the value function converges. Once optimal value functions are found for all cells, one sweep of policy improvement with greedy search is performed to find the optimal policy.

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

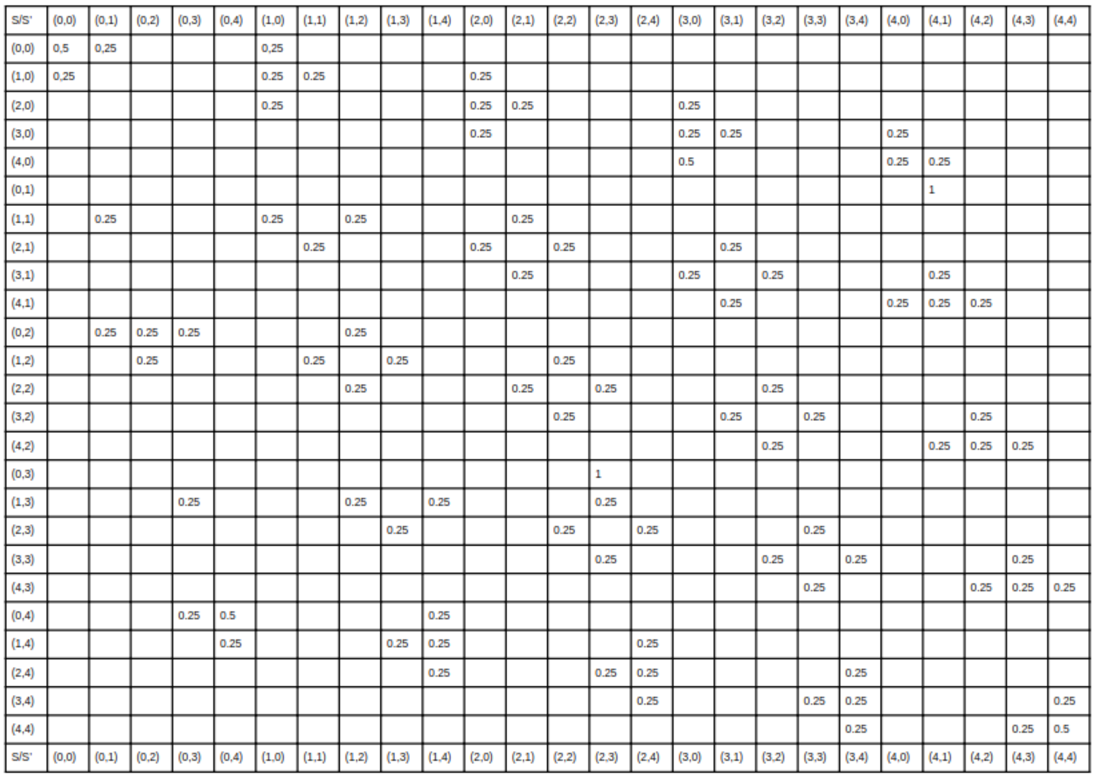
Between policy and value iteration, policy iteration was more difficult to implement because it required more steps (policy evaluation and improvement). More specifically, the policy evaluation must first sweep through every state to converge on a value for each state and then the improvement algorithm must sweep through every state again to find the best action to take in each state. Additionally, policy iteration also requires backing up more variables.

Value iterations has shown to have less iterations for each discount factor; however, policy iteration is able to find the optimal policy before the value converges. For example, for a discount factor of 0.99, optimal policy was found 1605 iterations before value converges. Because of this, policy iterations would work better when the problem is scaled up since optimal policy is always found before the value function converges.

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

It was found that there are two ways to update cells(states). First way is to update the cell with its new found value, and computation of values of the following cells is based on the new value. Second way is to update all the cells at the end, and they are updated all at once. The difference is whether you update the cells inside the for loop or outside the for loop. For policy iteration, both ways will lead to convergence. The second way will take more iterations than the first way. For value iteration, the first way will not converge, and the second way will converge. In the first way, we initiate cells value with 0, and we get new cells value with only A being 10, B being 5 and the rest of them being 0 . This leaves delta for some cells to be 0, which is less the threshold set (10^{-6}), then breaks the loop without maximizing the policy evaluation update.

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 7: Transition Matrix