Solving Markov Decision Processes with Dynamic Programming

Christoph Metzner (cmetzner)
ECE 517 - Reinforcement Learning
University of Tennessee, Knoxville

October 2021

1 Introduction

1.1 Late Student Problem

This project aims to solve a Markov Decision Process (MDP) using dynamic programming (DP) via the value-iteration-algorithm. A MDP = (S, A, P, R) is a stochastic control process in discrete time consisting out of states S, actions A, transition probabilities P, and rewards R (Puterman 1990). DP was introduced by Richard Bellman as a mathematical optimization method and a method to design programs in the 1950s (Sutton & Barto 2018). All programming in this work was completed using Python 3.
The goal of this work was to use DP to find the optimal policy of taking the best possible seat for a student (i.e., agent) arriving late to class (i.e.,environment) to maximize his or her learning outcome, while assessing the risks of getting sick with COVID-19. The student can maximize the learning outcome by sitting as far upfront as possible. The following conditions apply to the problem and define the way the agent behaves in the environment 1.

- There are N empty seats at the far right of the class; the student must seat in one of those.
- The student enters the class from the back of the room.
- When the student walks by a row, they can choose to either take the seat or move on to the next row.
- The student cannot go back to an earlier row.
- When at a row, the student can observe the next neighboring seat to the empty seat and asses if it is empty as well, has a student with a mask, or has a non masked student.
- When making a decision, the student is not able to see the situation in the next rows, i.e., the student cannot tell if anyone is sitting in the next row or not.
- The student would like to sit as far up as front as possible.

- If the student is not able to find a spot he will need to exit the class from the front door, which is undesirable.

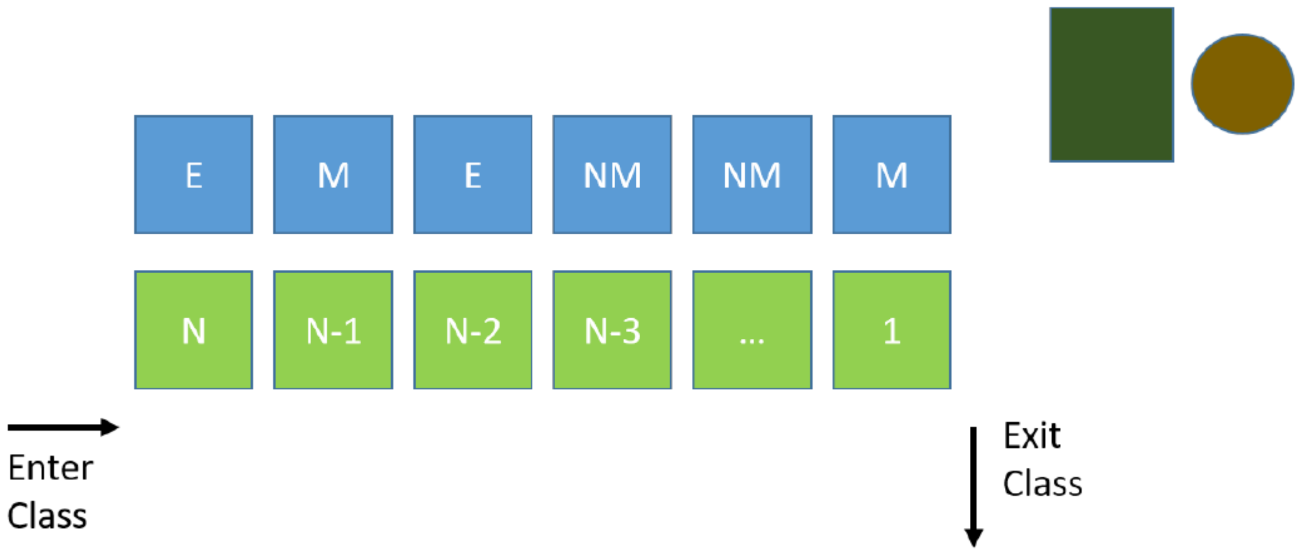


Figure 1: A diagram of the late student problem. The student enters the classroom in the back and they must decide for each row individually by looking at only the neighboring seat (blue) of the always empty seat (green) which action to take.

1.2 MDP for the Late Student Problem

The MDP = (S, A, P, R) for the late student problem is described below with a diagrammatic depiction of the problem in figure 2). This MDP is a discrete and episodic task and has a repetitive character. The MDP can terminate at either one of the following two points in a episode, that are a) the student decides to take the seat in the current row or b) the student does not find a seat and leaves the classroom. The environment has N rows and the condition of the neighboring seat can either be *empty*, *taken with student wearing a mask*, or *taken with student **not** wearing a mask*. The student can either choose to sit down at the seat in the current row or go to the next row. When choosing to go to the next row, the current state (row + condition of neighboring seat) has no effect on the probabilities getting to a certain state in the next row.

1.2.1 States - S:

The number of states depend on the number of rows N, the three conditions of the neighboring seat, and one terminal state. The total number of states is $(N * 3 + 1)$.

- Number of rows (1,...,N)
- Condition of neighbor seat (0, 1, 2)

  - Seat is empty, not taken
  - Seat is taken with masked student
  - Seat is taken with non-masked student

- One terminal state

1.2.2 Actions - A:

The agent can decide to either do one of the two actions.

- Take seat in current row - **stay**: 0
- Got to next row - **next**: 1

Action **stay** terminates the MDP at every given row. Action **next** only terminates the MDP if selected as the action in the final row, i.e., the row closest in the front.

1.2.3 Transition Probabilities - P:

The dynamics $p(s^{'}, r|s, a)$ of the MDP are known, provided as user input, and determine the action-environment interactions.

- Action: Stay
  - Probability of learning in any state: $p_{learn} = 1$
  - Probability of not getting sick depends on the condition of the neighboring seat
    - Empty: $p_{notsick-empty} = 1 - p_{sick-empty}$, where $p_{sick-empty}$ is given
    - Taken+Masked: $p_{notsick-masked} = 1 - p_{sick-masked}$, where $p_{sick-masked}$ is given
    - Taken+Nonmasked: $p_{notsick-nonmasked} = 1 - p_{sick-nonmasked}$, where $p_{sick-nonmasked}$ is given

- Action: Next
  - Probability to get from any state in row N to a seat with an empty neighboring seat in row N-1: $p_{empty} = 1 - p_{taken}$
  - Probability to get from any state in row N to a seat with a taken neighboring seat with a student wearing a mask in row N-1: $p_{taken+masked} = p_{taken} * p_{mask}$
  - Probability to get from any state in row N to a seat with a taken neighboring seat without a mask in row N-1: $p_{taken+nonmasked} = p_{taken} * (1 - p_{mask})$
  - Probability to exit the class from the first row: $p_{exit} = 1$
  - Probability to exit the class from any other row: $p_{exit} = 0$

1.2.4 Rewards - R:

- Action: Stay
  - Reward for learning outcome depends on: $R_{stay} = R_{learn} * row$, where R_{learn} is given and row is the row number the student decides to sit in
  - Reward of getting sick: R_{sick} given as input
  - Reward of not getting sick: $R_{not\_sick} = 0$
- Action: Next
  - Reward for going to next row, for all rows other than the first row: $R_{next} = 0$
  - Reward for leaving the class from the first row: $R_{next} = R_{exit}$ is given as input

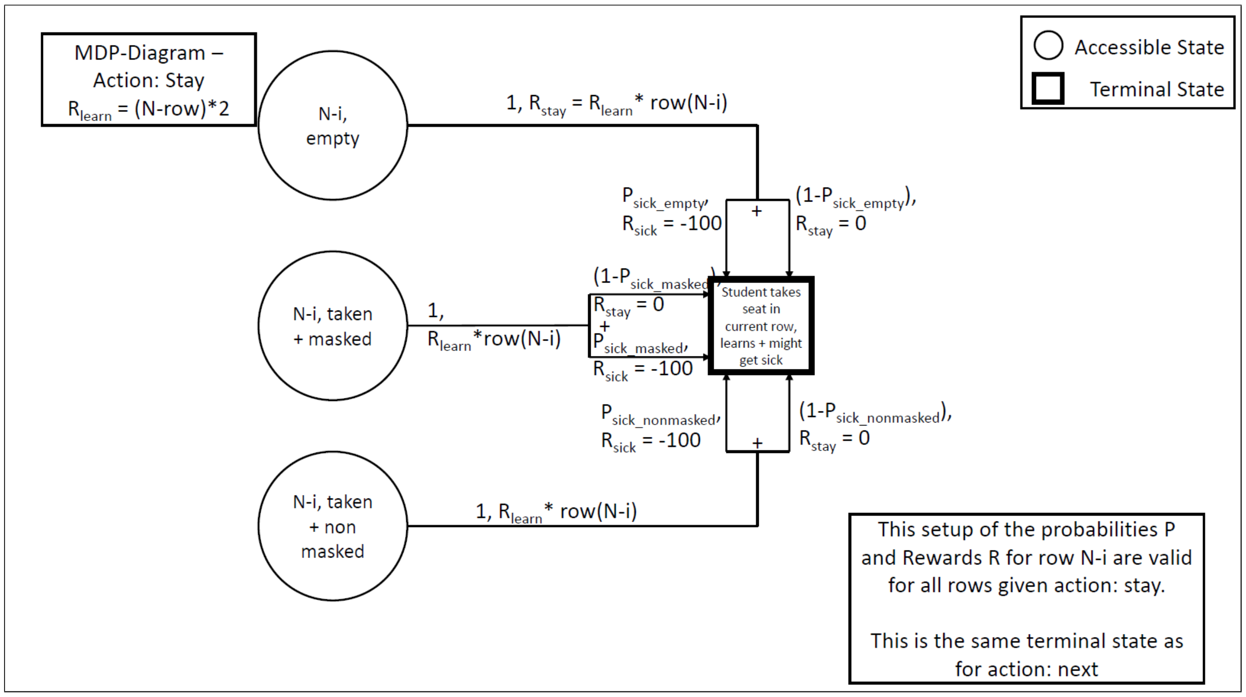
1.2.5 Example Setup:

The MDP for the late student problem was studied under the following example setup conditions:

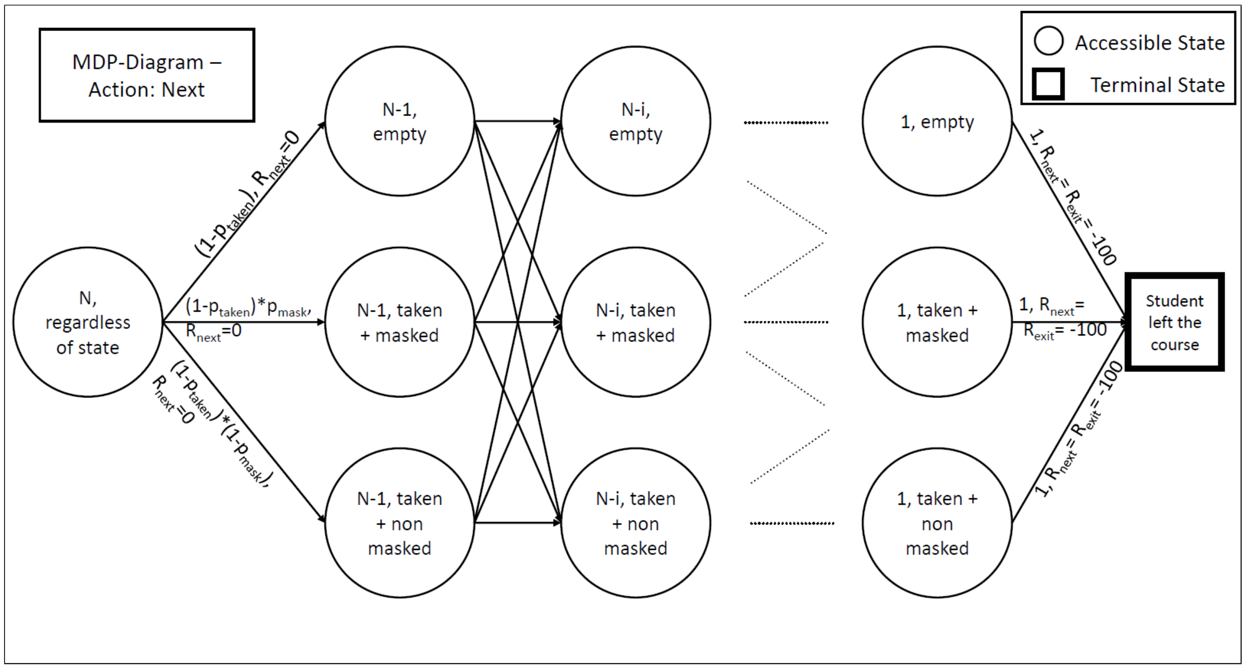
- N = 12, i.e., rows from 1 to 12, where 12 is the last row in the back of the classroom
- $p_{taken} = 0.5$, $p_{mask} = 0.5$, $p_{sick-empty} = 0.01$, $p_{sick-masked} = 0.1$, $p_{sick-nonmasked} = 0.5$
- The learning benefit gained from sitting in each row is $R_{learn} = (N - row) \times 2$
- Getting sick: $R_{sick} = -100$
- Exiting prematurely and missing class: $R_{exit} = -100$
- Discount factor of $gamma = 1$ was selected, since the MDP is episodic and the student wants to sit as far upfront as possible

2 Code Design and Data Structure

The main code is structured in three different functions. The first function `value_iteration` provides the mechanics to perform value iteration and loop through all states of the MDP to dynamically compute the state-value-function by calling the second function `get_state_value`. Once the optimal state-value-function V^* is found, the third function `policy_improvement` is called to do policy improvement by greedily selecting the best action for all states to determine the deterministic optimal policy π^*.
The data structure for the value-state-function is a 2-dimensional array with shape of $N * 3$, i.e., each row has three different conditions. The action-value-function is a 3-dimensional array with shape of $N * 3 * 2$, i.e., each row has 3 states and each state has two actions. The terminal state is not included in these arrays, but represented as a single variable with a state-value of 0.


MDP-Diagram – Action: Stay R_{learn} = (N-row)*2
Accessible State
Terminal State
N-i, empty
1, R_{stay} = R_{learn} * row(N-i)
P_{sick_empty}, R_{sick} = -100
$(1-P_{sick\_empty})$, R_{stay} = 0
N-i, taken + masked
1, R_{learn}*row(N-i)
$(1-P_{sick\_masked})$, R_{stay} = 0 + P_{sick_masked}, R_{sick} = -100
Student takes seat in current row, learns + might get sick
$P_{sick\_nonmasked}$, R_{sick} = -100
$(1-P_{sick\_nonmasked})$, R_{stay} = 0
N-i, taken + non masked
1, R_{learn}* row(N-i)
This setup of the probabilities P and Rewards R for row N-i are valid for all rows given action: stay.
This is the same terminal state as for action: next

(a) Action: Stay


MDP-Diagram – Action: Next
Accessible State
Terminal State
N, regardless of state
$(1-P_{taken})$, R_{next}=0
$(1-p_{taken})$*p_{mask}, R_{next}=0
$(1-p_{taken})$*$(1-p_{mask})$, R_{next}=0
N-1, empty
N-1, taken + masked
N-1, taken + non masked
N-i, empty
N-i, taken + masked
N-i, taken + non masked
1, empty
1, taken + masked
1, taken + non masked
1, R_{next} = R_{exit} = -100
1, R_{next}= R_{exit}= -100
1, R_{next} = R_{exit} = -100
Student left the course

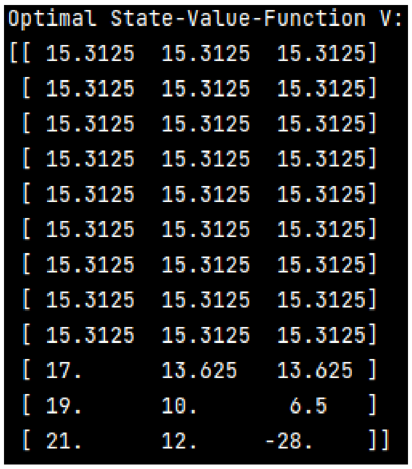
(b) Action: Next

Figure 2: Diagram of the MDP for the late student problem. The MDP has N*3 states plus one terminal state, two actions stay and next. Figure a) shows the MDP for action *stay* and figure b) shows the MDP for action *next*.

3 Results for Example Setup

3.1 Optimal State-Value-Function V^*

The optimal state-value-function for the MDP is shown in figure 3). The most desirable state with the highest state-value of $V(1, E) = 21$ for the agent is the seat in the front row with an empty neighboring seat. This makes sense, since the learning benefit is maximized while the risk of getting sick is minimal. The lowest state-value of $V(1, nonmasked) = -28$ was achieved in the first row as well despite receiving the highest learning outcome. This makes sense, since the learning benefit of +22 is summed up with a 50% chance of getting sick and receive a negative reward of -100, i.e., getting sick rewards -50. However, the best choice for the student is still to take a seat since -28 is greater than receiving -100 for leaving the classroom.
The state-values for the classroom rows 2 and 3 for the empty and masked case is always 2 less due to the calculation of the learning reward. The agent recognizes that the learning outcome decreases as he decides to stay in earlier rows.
The state-values of $V(12-4, E/M/NM) = 15.3125$ for all states in the earlier rows are the same. The reason for this is two-fold. First, the value-iteration algorithm always chooses the maximum action-value as the state-value and second, the dynamics of the MDP for the action *next* are the same and do not depend on the current state. Furthermore, the state-value depends on the expected return of the successor state. Also the action-value for the action *next* is larger than for the action *stay*, the state-values start to become different once the agent decides to take action *stay*. More about the optimal policy in the next chapter.

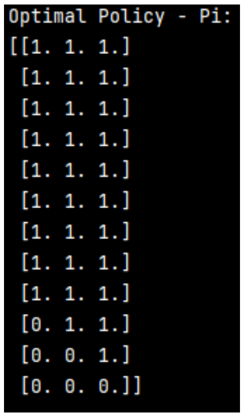


```
Optimal State-Value-Function V:
[[ 15.3125  15.3125  15.3125]
 [ 15.3125  15.3125  15.3125]
 [ 15.3125  15.3125  15.3125]
 [ 15.3125  15.3125  15.3125]
 [ 15.3125  15.3125  15.3125]
 [ 15.3125  15.3125  15.3125]
 [ 15.3125  15.3125  15.3125]
 [ 15.3125  15.3125  15.3125]
 [ 15.3125  15.3125  15.3125]
 [ 17.      13.625   13.625 ]
 [ 19.      10.       6.5   ]
 [ 21.      12.     -28.    ]]
```

Figure 3: Optimal state-value-function of the late student problem for the example setup. The state-values should be read as follows: each array represents the state-values for the condition of the neighbor seat - empty, taken+masked, and taken+nonmasked - for a row. The first array represents data for the first row the student can evaluate, i.e., that row is actually the row furthest away from the professor.

3.2 Optimal Policy π^*

The optimal policy π^* is shown in figure 4). The optimal policy for the agent acts according to the optimal state-value function. The agent starts to stay in rows where the neighboring seat is empty first if the learning outcome is high enough. This dynamic, the agent weighing the learning outcome vs. the risk of getting sick, can be also seen for states with the neighboring seat being taken by a masked student. Staying in a row where the neighboring seat is taken by a nonmasked student is almost never an option, i.e., the risk of getting sick is too high in comparison to the achieved learning outcome. For the first row, the agent decides to take action stay since the learning outcome plus the incorporated risk of getting sick is still larger than the negative consequences of leaving the classroom. This makes sense, since leaving the class is rewarded with a high negative number and thus, is never an option and studying reinforcement learning is always desirable. The best action for the student in the first nine rows in the back of the classroom is to go to next row regardless of the state of the neighboring seat. This means that the learning outcome in these rows are not high enough in general to justify the risk of getting sick.



```
Optimal Policy - Pi:
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [0. 1. 1.]
 [0. 0. 1.]
 [0. 0. 0.]]
```

Figure 4: Optimal policy for the optimal state-value-function of the late student problem for the example setup. The optimal policy should be read as follows: each array represents the optimal action for the condition of the neighbor seat - empty, taken+masked, and taken+nonmasked - for a row. The first array represents data for the first row, last row in the classroom, the student can evaluate.

4 Result Analysis

In the following section, the behavior of the value-iteration algorithm is analyzed. First, the optimal policy is determined for different discount factors γ, the number of epochs until convergence for different number of rows N, and different dynamics of the MDP.

4.1 Optimal Policy for different Discount Factors

The discount factor γ determines how far-sighted the agent acts in the environment. An agent with a discount value of $\gamma = 0$ acts short-sighted and only cares about the immediate reward it gets for doing a certain action. In contrast, an agent with a discount value of $\gamma = 1$ puts equal importance to all rewards, the current and all future rewards. Figure 5) shows the relationship between the optimal policy and the discount factor used in value-iteration. The less far-sighted an agent behaves (discount factor approaches 0) in the environment the agent decides to take action **stay** much earlier in the process. With $\gamma = 0$ it decides to stay in row 10 if neighbor seat is empty. On the flip-side, if the agent is far-sighted ($\gamma = 1$) it will wait to select a seat until the learning outcome becomes very good compared to the risk of getting sick and chance of leaving the classroom. For all discount factors, selecting a seat and staying if the neighbouring seat is taken by an unmasked student is never an option bur for the first row.

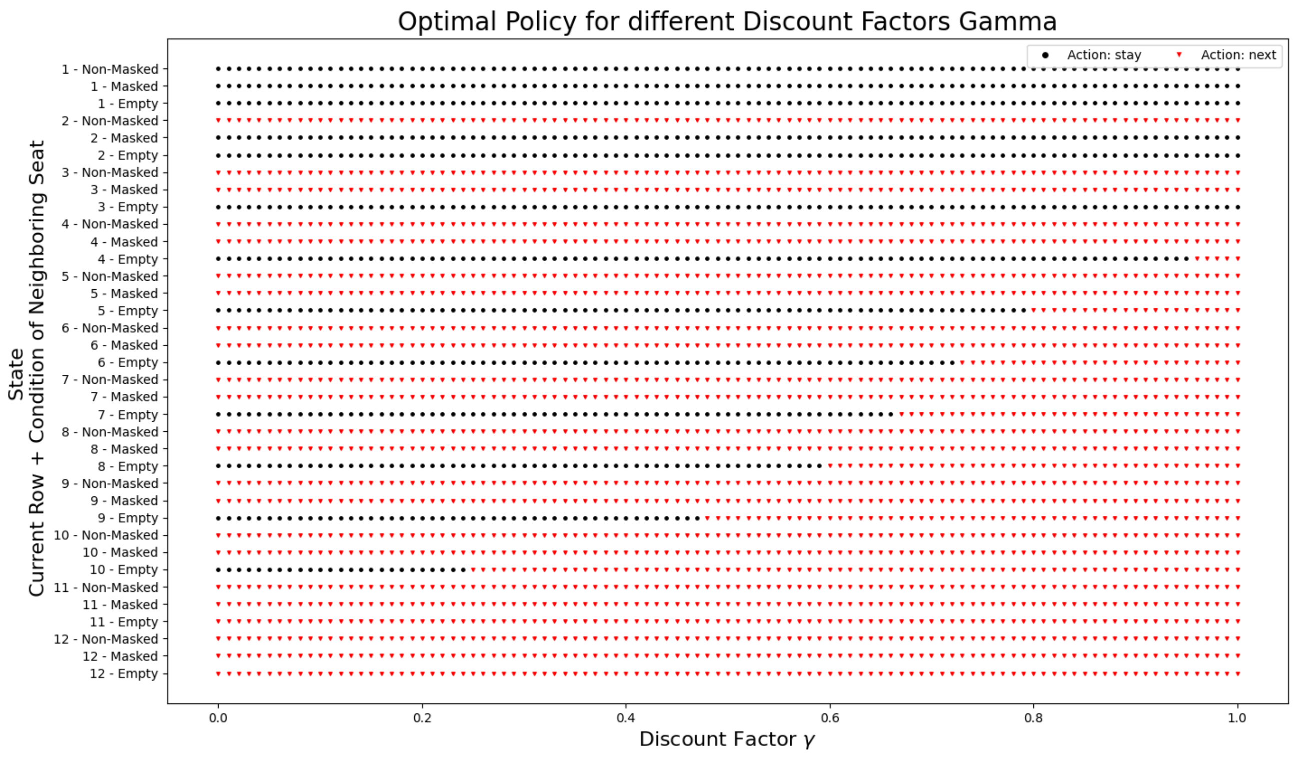


Figure 5: Optimal Policy for different discount factors γ. Action: stay is indicated as black dot and action next is indicated as a red dot.

4.2 Run time vs size of N rows

The following analysis indicates that the runtime, i.e., time until the algorithm converges, in terms of total number of rows N increases linearly. This is due to the linear nature of the learning outcome (reward for learning) depending on the row number.

4.3 Policy Changes due to Changes in Dynamics

In the following analysis, some dynamics of the MDP were changed to see the effects on the optimal state-value-function and the optimal policy. The basis of the changes is the example setup from chapter 1.2.5. Each change is documented for a different case. A discount factor of $\gamma = 1$ was used in all cases. The specifics of the changes are the following:

- Case a: A novel, more infectious variant of SARS-COV-2 emerged and the probabilities of getting sick for all conditions of the neighboring seat increased (Figure 7).
  - $p_{sick-empty} = 0.1$
  - $p_{sick-masked} = 0.3$

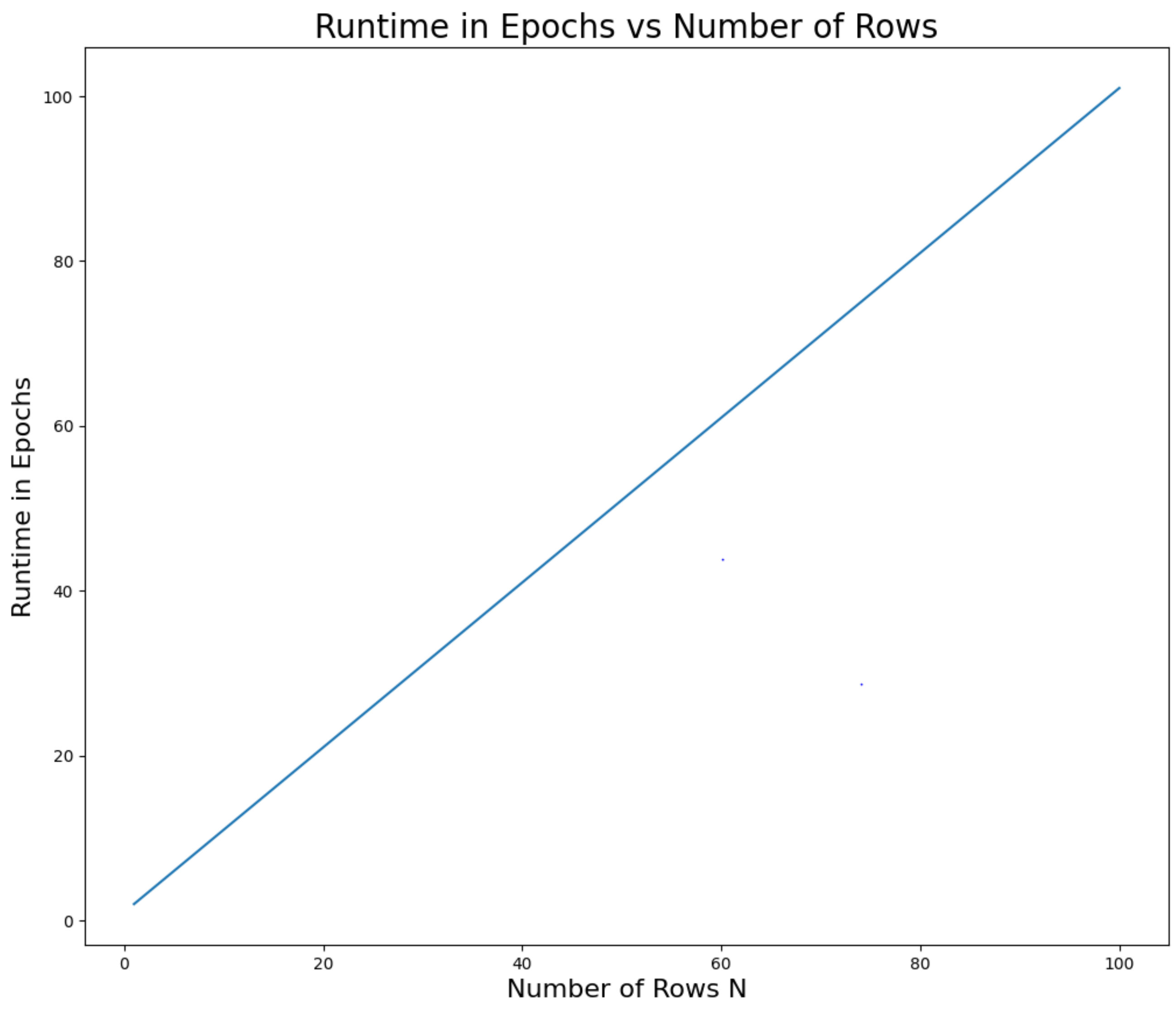


Figure 6: Optimal Policy under Optimal State-Value-Function for example setup.

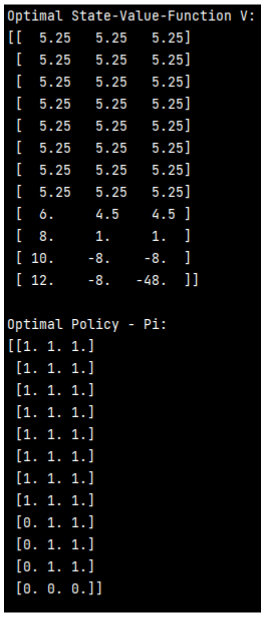
- $p_{sick-nonmasked} = 0.7$

- Case b: The reward for leaving the class is set to 0 instead of -100. The student is able to learn on its own at home (Figure 8).

- Case c: Getting sick has less impact on the health of a student, thus the negative reward was decreased from -100 to -30. (Figure 9).

4.3.1 Case A: Higher Risk of getting Sick

If the probabilities of getting sick are increased the state-values decrease for all states. Furthermore, the agent now takes getting sick more seriously than before (compare Figure 7 with Figure 4) and would select action next one more time instead of already taking a seat if the neighboring seat is taken by a masked student. Interestingly, the agent is more willing to stay in row 4 if the neighbor seat is empty. This could stem from the new expected returns for the successor states, i.e., choosing action next may be not as beneficial anymore since there is a chance to get into an undesirable state (masked or nonmasked student). This makes sense, since the negative reward of getting sick has a higher chance of occurring, thus the overall state-value functions drop.



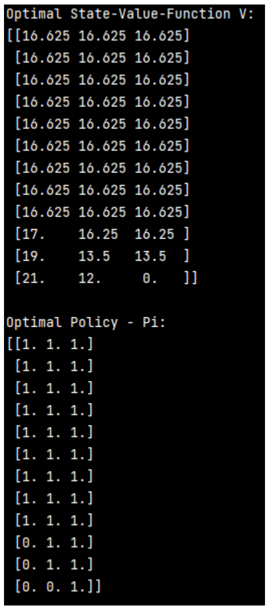
```
Optimal State-Value-Function V:
[[  5.25   5.25   5.25]
 [  5.25   5.25   5.25]
 [  5.25   5.25   5.25]
 [  5.25   5.25   5.25]
 [  5.25   5.25   5.25]
 [  5.25   5.25   5.25]
 [  5.25   5.25   5.25]
 [  5.25   5.25   5.25]
 [  6.     4.5    4.5 ]
 [  8.     1.     1.  ]
 [ 10.    -8.    -8.  ]
 [ 12.    -8.   -48.  ]]

Optimal Policy - Pi:
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [0. 1. 1.]
 [0. 1. 1.]
 [0. 1. 1.]
 [0. 0. 0.]]
```

Figure 7: Case: Higher probability of getting sick.

4.3.2 Case B: No Reward for Exiting the Classroom

The optimal state-value function and optimal policy slightly change for all states compared to the example setup. We can now see, that leaving the classroom actually becomes an option for the student. This happens since the student does not have to fear a negative reward when leaving. The state-values become overall a little larger since the there is no negative reward for leaving the classroom anymore and thus the expected return must increase.



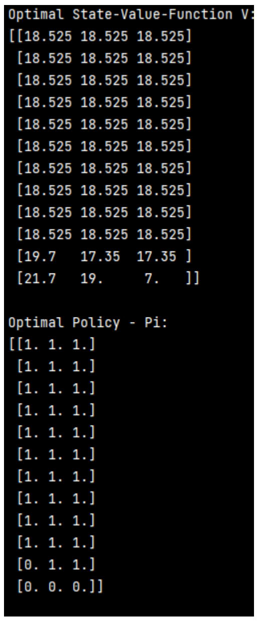
```
Optimal State-Value-Function V:
[[16.625 16.625 16.625]
 [16.625 16.625 16.625]
 [16.625 16.625 16.625]
 [16.625 16.625 16.625]
 [16.625 16.625 16.625]
 [16.625 16.625 16.625]
 [16.625 16.625 16.625]
 [16.625 16.625 16.625]
 [16.625 16.625 16.625]
 [17.    16.25  16.25 ]
 [19.    13.5   13.5  ]
 [21.    12.     0.   ]]

Optimal Policy - Pi:
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [0. 1. 1.]
 [0. 1. 1.]
 [0. 0. 1.]]
```

Figure 8: Case: No negative reward of leaving class.

4.3.3 Case C: Less negative Reward for getting Sick

The optimal state-value-function increases compared to the example setup case. This makes sense since the negative reward has been decreased, i.e., the negative reward has a lower contribution to the state values if the student decides to stay. Interestingly, despite a lower negative reward for potentially getting sick, the student decides to not sit in earlier rows. This could stem from the fact, that the state-values for the first row are much larger now and change the whole state-value function for the MDP.



```
Optimal State-Value-Function V:
[[18.525 18.525 18.525]
 [18.525 18.525 18.525]
 [18.525 18.525 18.525]
 [18.525 18.525 18.525]
 [18.525 18.525 18.525]
 [18.525 18.525 18.525]
 [18.525 18.525 18.525]
 [18.525 18.525 18.525]
 [18.525 18.525 18.525]
 [18.525 18.525 18.525]
 [19.7   17.35  17.35 ]
 [21.7   19.     7.   ]]

Optimal Policy - Pi:
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [0. 1. 1.]
 [0. 0. 0.]]
```

Figure 9: Case: Lower negative reward for getting sick.

5 Code

Please use the following command in your command line to run the example setup:

```
python ECE517_RL_MetznerChristoph_Project1.py 12 0.5 0.5 0.01 0.1 0.5 -100 -100 2
```

or,

```
python3 ECE517_RL_MetznerChristoph_Project1.py 12 0.5 0.5 0.01 0.1 0.5 -100 -100 2
```

6 Conclusion

Dynamic programming and value iteration is able to find the optimal policy for this MDP problem. However, this is only possible if the dynamics of the MDP are known. If the dynamics for more complex MDP's are unknown a different solution has to be deployed, e.g., an algorithm that approximates the dynamics of the MPD. Furthermore, even for small RL problems, such as the late student problem, the discount factor plays a vital role on how the optimal policy turns out to be. However, since this is an episodic task a discount factor of 1 is reasonable. Changing the dynamics of the MDP has an impact on the optimal state-value-function and the optimal policy. Therefore, for problems with more uncertain dynamics of the MPD the optimal policy may change over time and the agent needs to be able to adjust for these uncertainties, i.e., nonstationary RL problems. Specifically for this problem, the student now has a detailed policy what to do when arriving late to class and having to select a seat that maximizes his learning outcome while considering the risk of getting sick with COVID-19.
Furthermore, I realized, that despite commenting a convoluted function becomes difficult to follow. In my next works, I have to focus more on keeping the functions as small an flexible as possible.

References

Puterman, M. L. (1990), ‘Markov decision processes’, *Handbooks in operations research and management science* **2**, 331–434.

Sutton, R. S. & Barto, A. G. (2018), *Reinforcement learning: An introduction*, MIT press.