

Hamza Wahhas

Gridworld Project

6/21/2023

Markov Decision Processes

For this project, I chose two MDPs based on the grid world example. There is one small grid world and one large grid world, both containing 2 terminal states. The small grid world has 9 states which is smaller than the typical 12 states in the example while the large grid world contains 20 states. I used the mdptoolbox library in Python to utilize the policy and value iteration methods.

Small Grid World

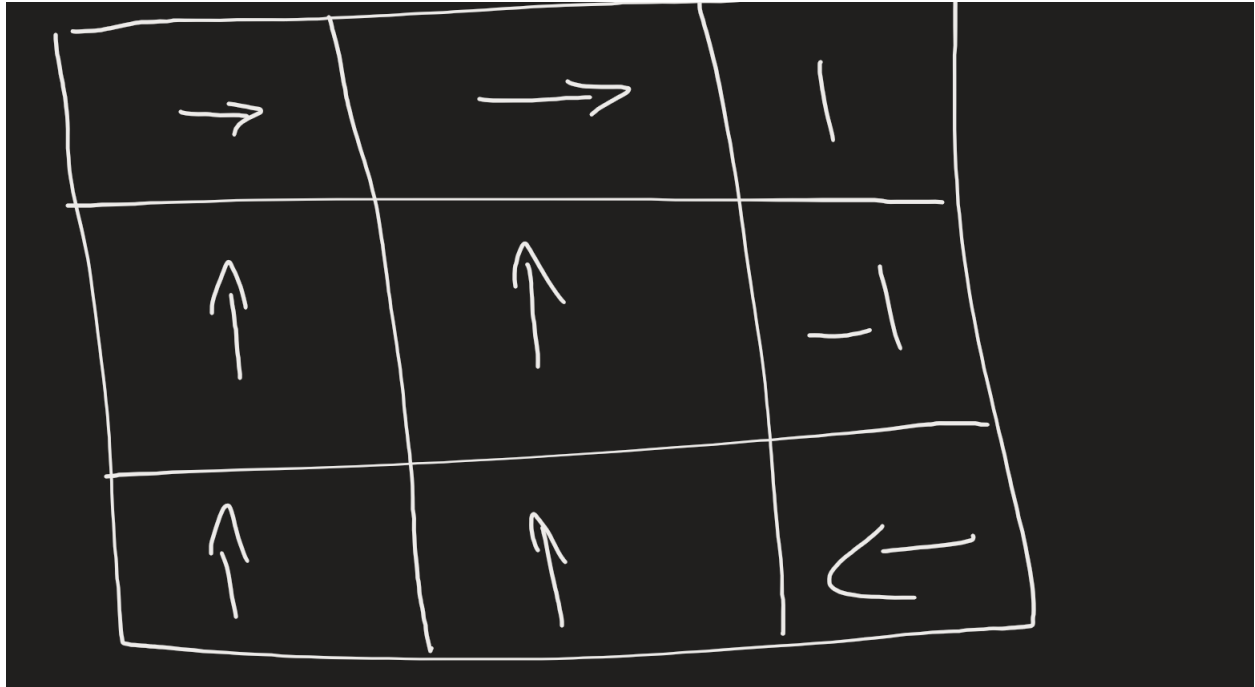
The actions for the small grid world follow the original grid world with every state except the terminal ones being able to move up, down, left, and right. The agent is locked within the grid world and will stay in the same state if it hits the wall. The grid worlds are also completely deterministic and each move has a 100 percent chance of being completed. The discount factor was set to .9, which is usually what most MDPs base their value around to focus on the best reward and ultimately the best policy rather than a low discount factor which would focus on getting rewards immediately, basically being greedy and can result in a lower reward at the end. Further details are located in the analysis at the end.

$-.04$	$-.04$	1
$-.04$	$-.04$	-1
$-.04$	$-.04$	$-.04$

It should be noted that in both of these diagrams, there is no empty state or a state that can't be reached or left from, typically in the middle. Every state has a value of $-.04$ to encourage reaching the terminal states faster.

Policy Iteration

When running the program, I got an optimal policy of $(3, 3, 0, 0, 0, 0, 0, 0, 2)$ with it taking 4 iterations to converge. The diagram for the policy is as follows.



The agent typically wants to get to the top row to get closer to the reward of 1. In the bottom right state, it chooses to go left rather than going up due to the punishment of -0.04 being better than the punishment of -1 .

The time to run this was ~ 1 ms, making it very fast.

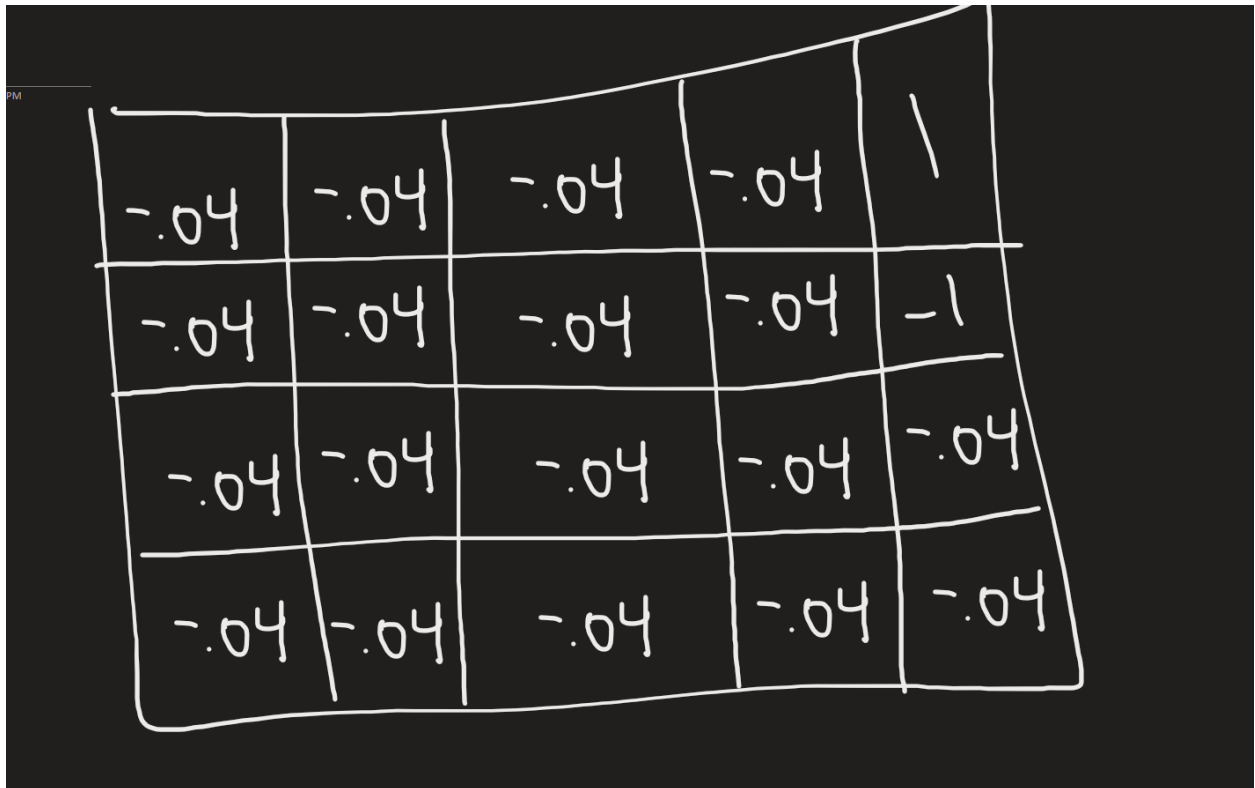
Value Iteration

When running the program, the output is $(3, 3, 0, 0, 0, 0, 0, 0, 2)$ and the number of iterations is 72. The policy is the same but the number of iterations increased substantially under value iteration. I expected this due to value iteration converging on an optimal policy slower than policy iteration. The only downfall of policy iteration is that it is more complex to compute, but because the library we imported did the hard work, we can use policy iteration to find the optimal policy quickly.

The time to run this was .001, which was slower than the time to converge the policy iteration, further supporting our analysis.

Large Grid World

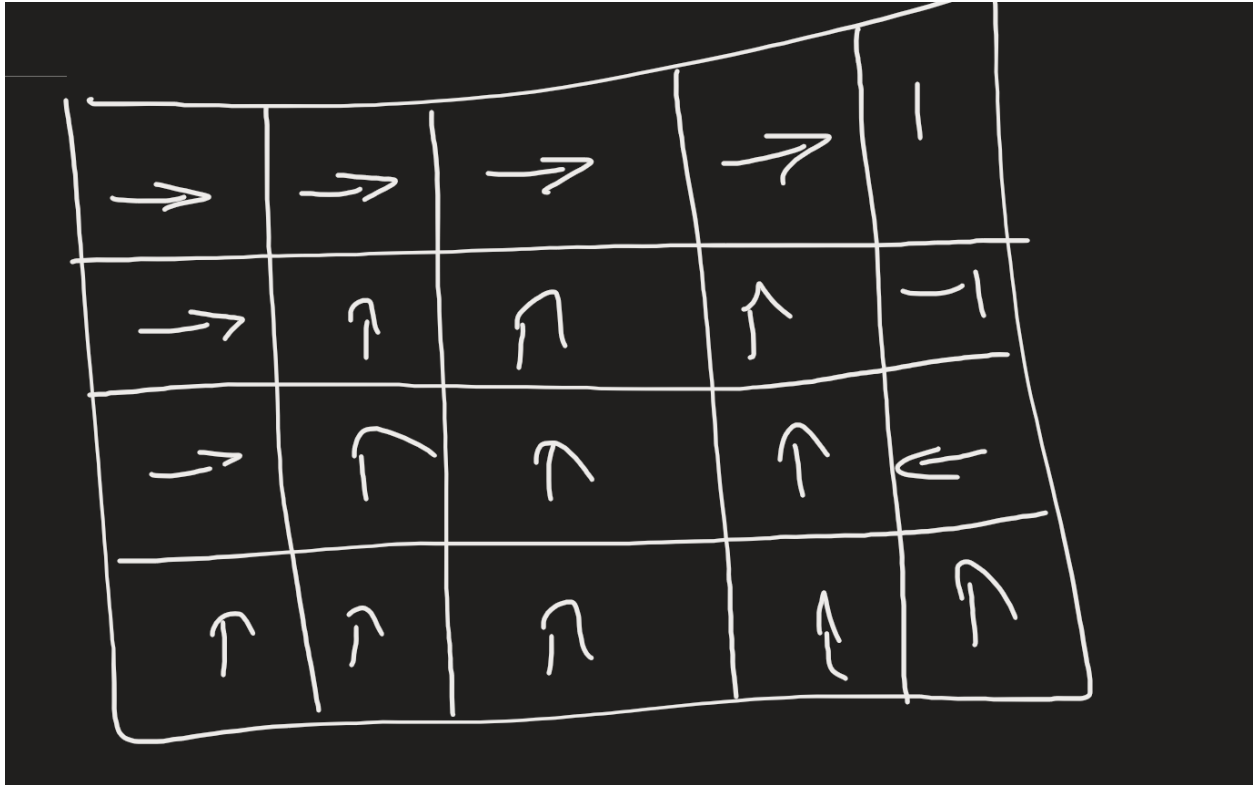
This is the same as the small grid world except it has 20 states instead of 9. It follows the same format with the top right holding the terminal state with a reward of 1, the state below with a punishment of -1, and every other state with a punishment of -.04 to increase convergence speed on an optimal policy.



-.04	-.04	-.04	-.04	1
-.04	-.04	-.04	-.04	-1
-.04	-.04	-.04	-.04	-.04
-.04	-.04	-.04	-.04	-.04

Policy Iteration

When running the program this is the optimal policy using policy iteration (3, 3, 3, 3, 0, 3, 0, 0, 0, 0, 3, 0, 0, 0, 2, 0, 0, 0, 0, 0). In diagram form, it looks like this.



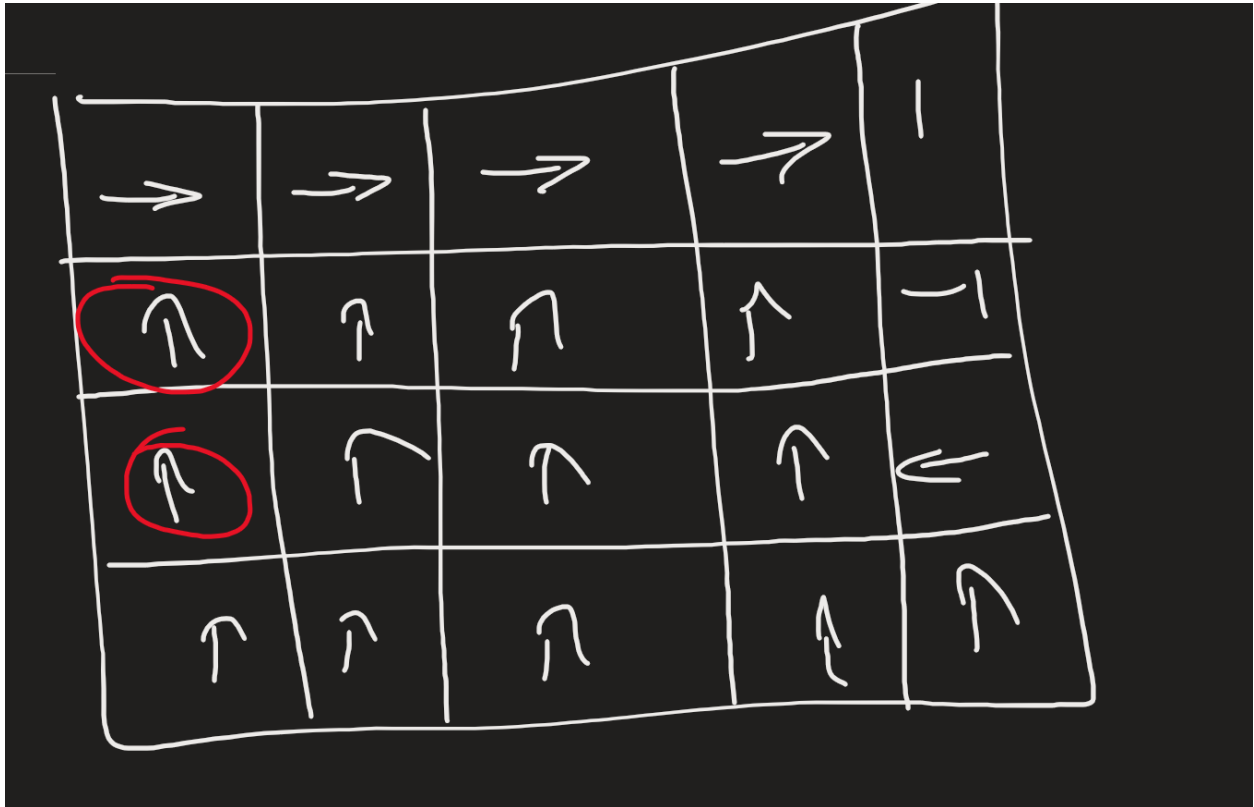
Just like before the best move is to go up and if you are already in the top row keep going right.

It also continues the pattern where the agent avoids -1 due to it resulting in the worst result. The number of iterations is larger than the smaller grid world with 8 instead of 4.

The time to run this was .001ms which is slower than the small grid world, proving that more states result in a slower convergence time.

Value Iteration

When running the program the optimal policy I get is (3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0). This is slightly different from the first policy I got when running policy iteration with the 6th and 11th state instead choosing to go up rather than going right.



Either or wouldn't change the speed as it was about the same as the policy iteration. Oddly enough, however, it had 72 iterations which mirror the number of iterations from the small grid world value iteration. I assume this is because the problems are similar and the number of states isn't that much larger to change the amount needed to converge.

Analysis

I found these MDPs interesting due to them being similar to a game in a sense. You have obstacles that push you in the direction of an optimal path. For example, imagine if, in a video

game, you had two tunnels, one that ranks you up and another that demotes you. The best policy would be to go to the one that promotes you, but you don't have to so you can wait or move around until you choose to enter one of the terminal states. This is where the discount factor comes in, if it's 0, you wouldn't feel as inclined to go through either gate but if the discount factor was high, let's say .9, you would know that the future policy for the highest reward would probably be taking that tunnel. Now let's say you wanted this to converge faster, you can add punishment if you don't enter the tunnels, something small like .04 like we did in the grid world. The agent would want to quickly leave the punishment for the reward. This type of format is in all types of games which is what makes this example fun.

The policy iteration and value iteration function didn't have to be implemented thankfully due to the library `mdptoolbox` which handled it for us. All I had to do was implement the transitions and the rewards. I used the discount of .9 to focus on rewards in the future rather than in the present, resulting in a better optimal policy. For the transitions, I chose to keep them deterministic to make things easier to interpret. From each state, it can go up down left, or right and in cases where there is no state to move to from that action it will stay in that same state. I created a transition from all states using each action. In the case of terminal states, I kept it within the same state no matter what action, to signify the termination of the program. When it came to rewards, I set the top-right state with a reward of 1 and the state below that with a punishment of -1, making the target the top-right state. I set the values within the array for the 4th row as 1 off any action and the 9th row as -1 off any action. The rest of the states as specified earlier were -.04 on all actions to increase convergence speed. The speed might've decreased with the increase of states but I didn't see a large difference in convergence time, most likely due to the efficiency of the value iteration and policy iteration functions created in the `mdptoolbox` library. Due to it being a

small problem, I think the discount factor could be decreased and the optimal policy would still be found, as well as a lower number of iterations. A change to this program when it comes to actions wouldn't be that difficult when it comes to creating the rewards transitions but when implementing the transition function from each state, you would have to implement an entire row of new transitions in this case I would have to check the result of each action from every state in the grid world, the time of which depends on the number of states and actions.