COL778
Principles of Autonomous System
Assignment - 2
Markov Decision Processes

Vaibhav Seth

3rd March, 2024

Contents

1 Environment Description

In this problem we are solving MDPs for car navigation. The car comes with a hybrid engine, allowing it to run on petrol or electricity. It is capable of moving in four directions: top, down, right, and left. Due to construction work, holes have been dug up in the streets of the grid world. Therefore, the task is to find an optimal policy (the action it needs to take in every grid cell) to avoid falling into the holes and drive safely to its destination.

The environment is represented as a 2D grid world with discrete grid cells. Specific cells are designated as 'H,' indicating the presence of a hole. The garage is denoted by 'S,' where the vehicle is initially parked, and the destination is marked as 'G.' Cells that are neither holes nor the goal are labelled as 'F,' indicating free cells.

1.1 Transtition Model

The transition model has following actions :

$$right(x, y) = (x + 1, y)$$

$$left(x, y) = (x - 1, y)$$

$$top(x, y) = (x, y + 1)$$

$$down(x, y) = (x, y - 1)$$

Because of rain, the roads in the grid world become slippery, introducing uncertainty to the effects of actions. In other words, the transitions are stochastic. Any action will result in the intended cell with a probability p and randomly in one of the other three neighbours with probability $(1 - p)$. Transitions outside the grid boundary are not allowed. That is, transitions resulting in a position outside the grid boundary will not change the state. Hence, the probabilities associated with any illegal or out of transition move are added to the probability of being in the state.

We have make a key assumption that moving into the hole state is considered legal, otherwise the robot would implicitly know that it can't move into hole and would have a low probability of moving into the goal and hence does not really "learn" that it should not enter a hole.

1.2 Goal Model

The car receives a reward after moving into a state, i.e, the rewards are of the form $R(s, a, s')$

$$R(s, a, s') = \begin{cases} living\_reward & s' \in F \\ hole\_reward & s' \in H \\ goal\_reward & s' = G \end{cases}$$

2 Solving for Optimal Policy

- The transition probability p is 0.8, meaning that 80% of the time, actions lead to the intended state, while there's a 20% chance of reaching a random neighbouring cell.
- The living_reward and hole_reward are 0. The goal_reward is 1.
- The discount factor, γ, is 0.9.

2.1 Vanilla Ice Cream Value Iteration

Table 1: epsilon vs metrics for small map

ϵ	**Iter.**	**Wall-Time(s)**	**Final-Delta**
1e-1	10	0.013	1e-2
1e-3	16	0.034	1e-4
1e-6	26	0.035	1e-8
1e-9	35	0.039	1e-11
1e-12	44	0.124	1e-13

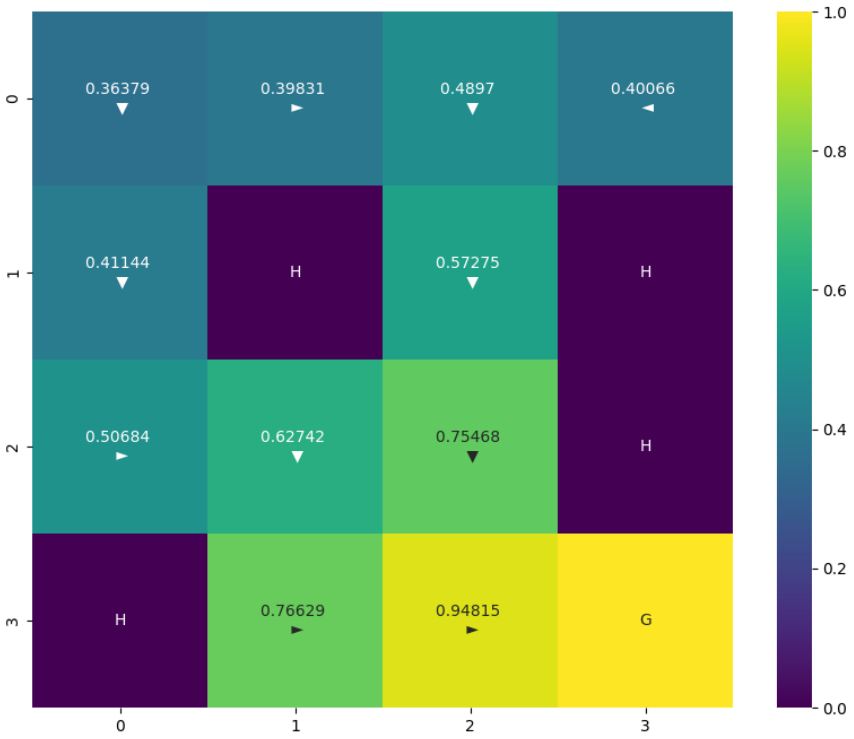

Figure 1: Policy Heatmap for Small Map Vanilla Value iteration ($\epsilon = 1e - 9$)

Table 2: epsilon vs metrics for large map

ϵ	**Iter.**	**Wall-Time(s)**	**Final-Delta**
1e-1	24	3.363	1e-2
1e-3	60	7.527	1e-4
1e-6	116	14.786	1e-7
1e-9	163	20.25	1e-10
1e-12	188	23.83	1e-13

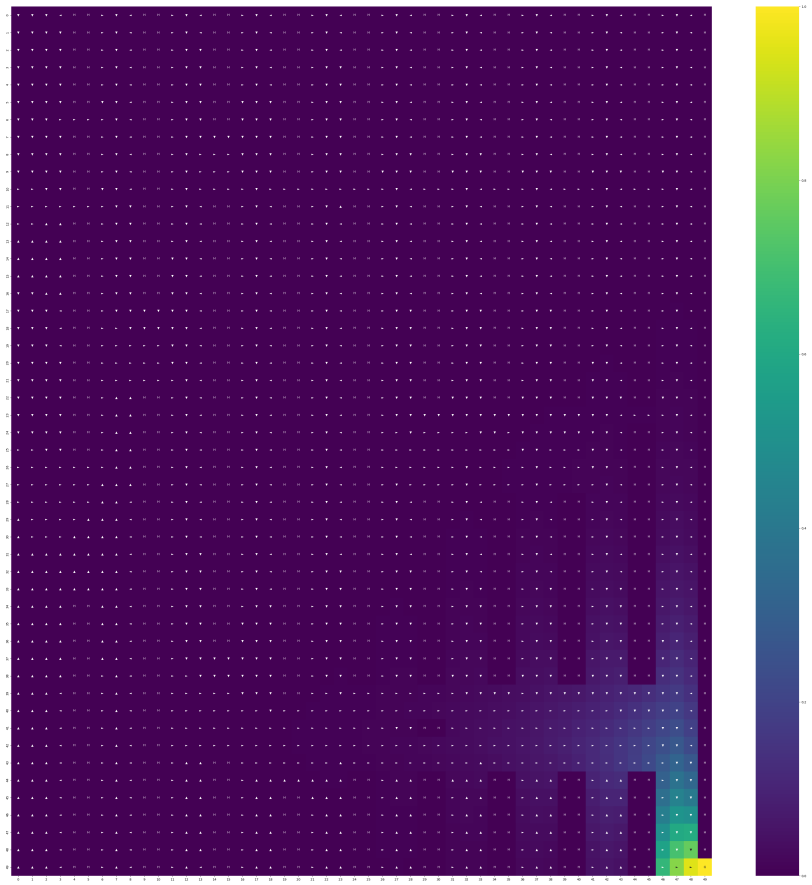

Figure 2: Policy Heatmap for Large Map Vanilla Value iteration ($\epsilon = 1e-9$)

We choose $\epsilon = 1e-9$ for subsequent experiments as it gives a good convergence time with decent number of iterations and a good delta as well.

2.2 Row-Major Sweep Value Iteration

The values are updated in the row major order. We update the states in the first row, then the next and so on till we reach the goal state. No update is performed for the terminal states.

The following values are obtained over the small map

$$\epsilon = 1e{-}9$$

$$Wall - Time = 0.022s$$

$$Iterations = 21$$

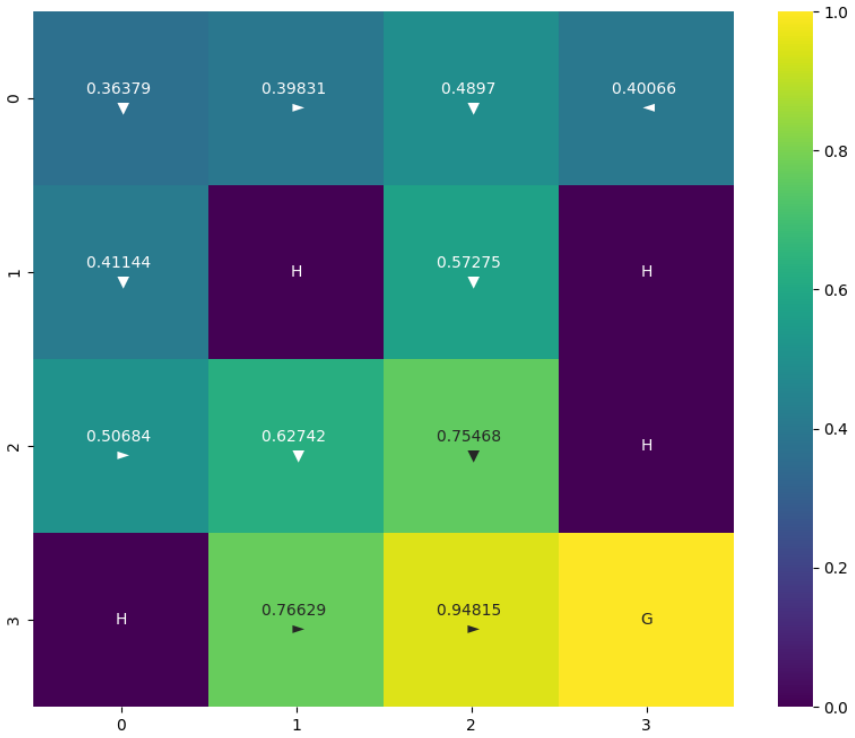

Figure 3: Policy Heatmap for Small Map Row Major Value iteration ($\epsilon = 1e-9$)

The following values are obtained over the large map

$$\epsilon = 1e{-}9$$

$$Wall - Time = 17.376s$$

$$Iterations = 124$$

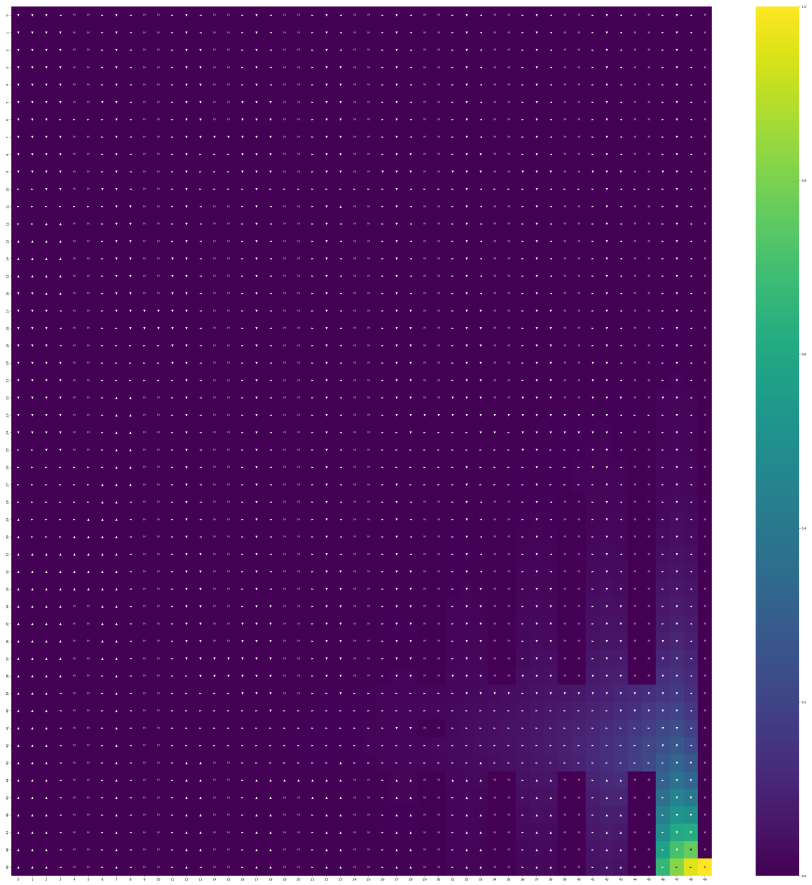

Figure 4: Policy Heatmap for Large Map Row Major Value iteration ($\epsilon = 1e - 9$)

In both cases we see that the row major sweep takes lesser time to converge and also take less number of updates over all state.

2.3 Prioritized Sweep Value Iteration

The values are updated in the order of their bellman errors (largest first). The heap is initialised with the state encoding values and then the prioritized updates are done. We have also kept a parameter of max updates, so if the number of state updates when the heap is empty is less than the max updates, we again enter the prioritized update loop. This results in a better update of values of states further away from goal. Convergence criteria is taken as the max-norm of difference between consecutive values and epsilon

The following values are obtained over the small map

$$\epsilon = 1e{-}9$$

$$\text{max updates} = 500$$

$$\text{Time for heap step } 1 = 0.014$$

$$\text{Wall-Time} = 0.014s$$

$$\text{Number of state updates} = 160$$

$$\text{Number of state updates for vanilla} = 560(35 * 16)$$

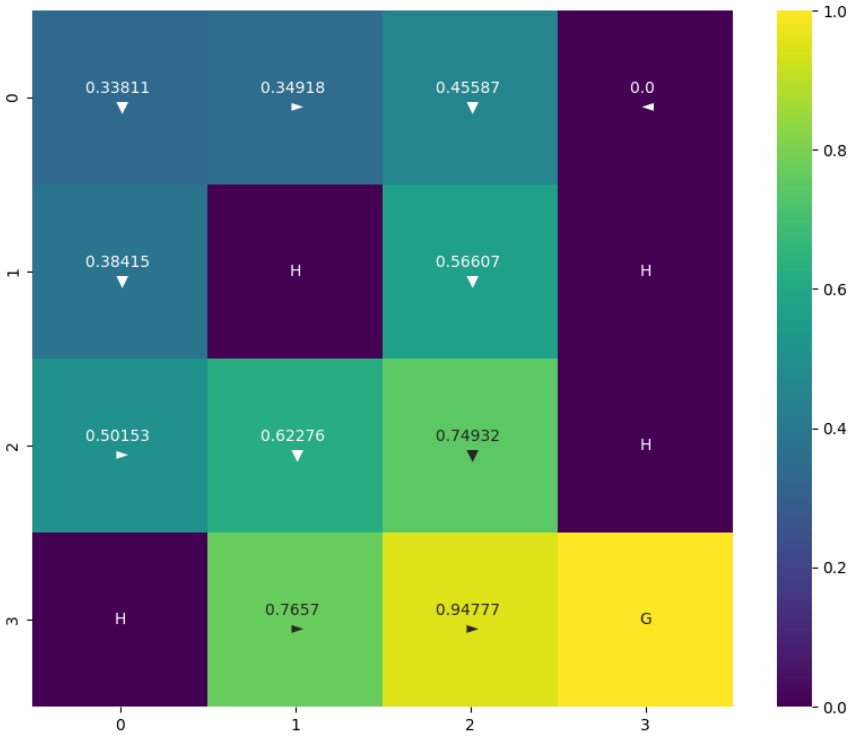

Figure 5: Policy Heatmap for Small Map Heap Value iteration ($\epsilon = 1e - 9$)

The following values are obtained over the large map

$$\epsilon = 1e{-}9$$

$$\text{max updates} = 5000$$

$$\text{Time for heap step } 1 = 0.374$$

$$\text{Wall-Time} = 0.374s$$

$$\text{Number of state updates} = 4443$$

Number of state updates for vanilla = 407500

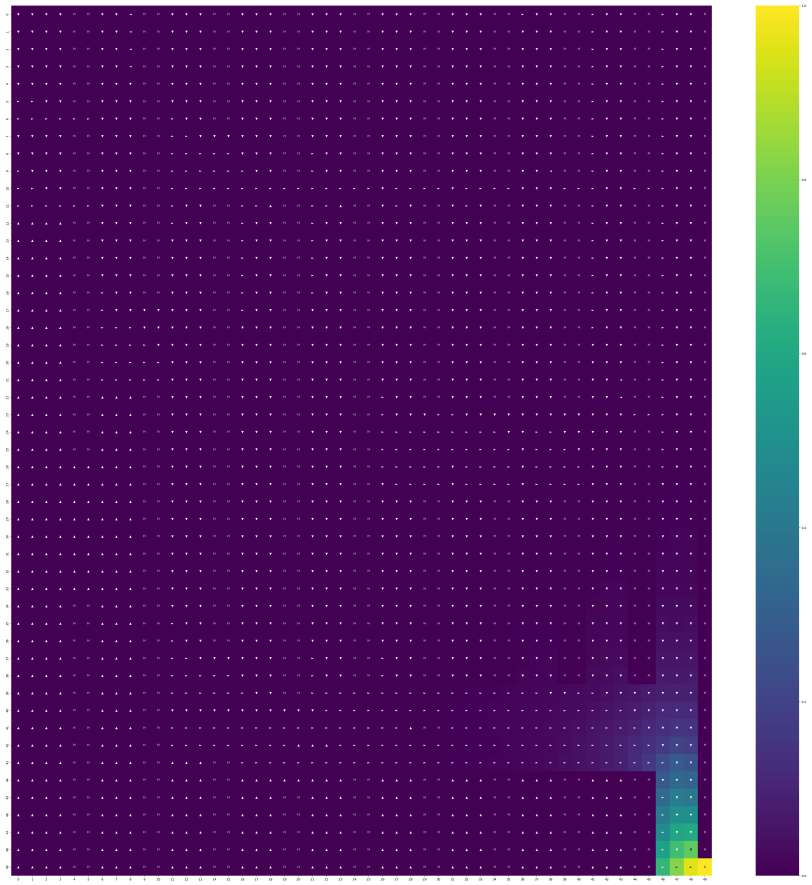

Figure 6: Policy Heatmap for Large Map Heap Value iteration ($\epsilon = 1e - 9$)

In both cases we see that the prioritized sweep takes lesser time to converge and also take less number of updates over all state. In the larger map the effect is much much more profound as the states far from the goal state do not get updated much and hence they do not require too many computations. The policy converges before the value does and that is why we get a good policy in such a short time.

2.4 Policy Iteration

We have implemented the standard policy iteration using the algorithm taught in class. The convergence criteria used is max-norm of change in value of the policy.

For the small map, all the epsilon mentioned in value iteration, policy iteration takes only 3 iterations in all of them (except 1e-1). The time taken is of the order 1e-3 to 1e-2. The policy obtained from all are identical.

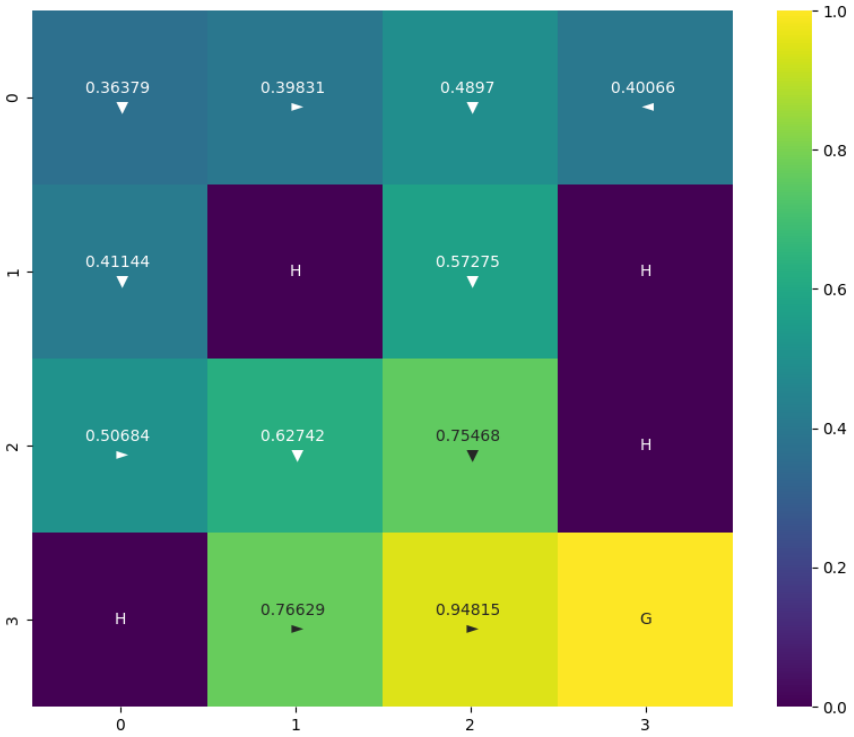

Figure 7: Policy Heatmap for Small Map Policy iteration ($\epsilon = 1e - 9$)

Table 3: epsilon vs metrics for large map

ϵ	**Iter.**	**Wall-Time(s)**	**Final-Delta**
1e-1	2	1.22	1e-1
1e-3	6	2.54	1e-4
1e-6	7	2.94	1e-8
1e-9	9	3.65	0.0
1e-12	10	4.38	0.0

For the large map, policy iteration run significantly faster than vanilla value iteration or row major, It still lags behind the prioritized value iteration, but because of the quality of result, policy iteration is a very good choice.

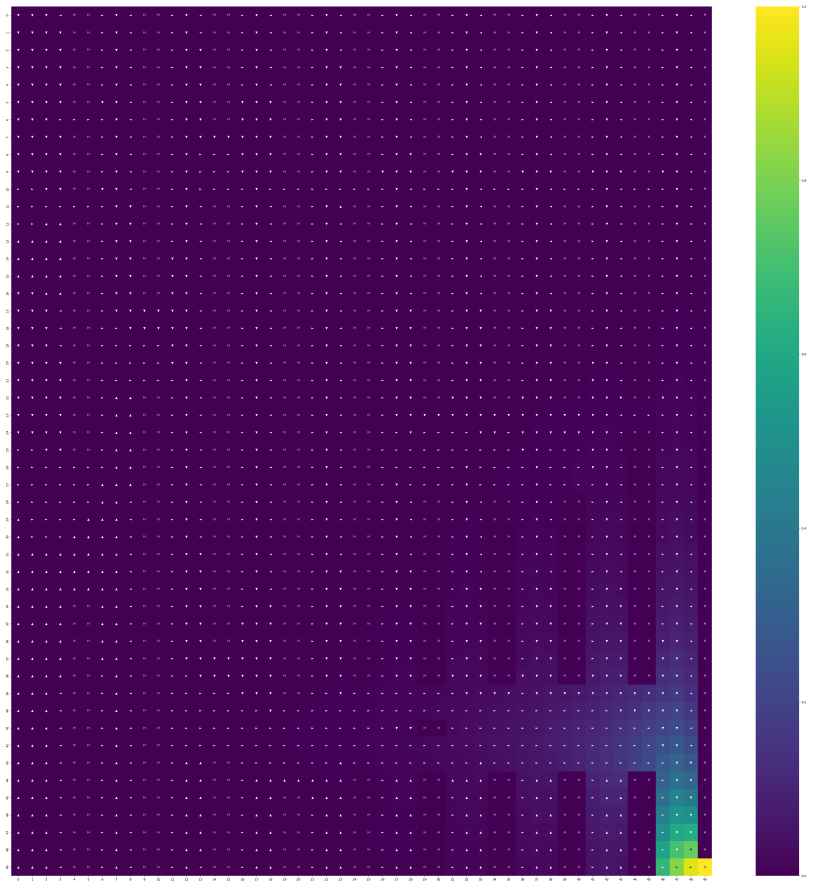

Figure 8: Policy Heatmap for Large Map Policy iteration ($\epsilon = 1e - 9$)

Policy Iteration and Prioritized are clearly scalable for larger environments. Even though policy iteration take fairly more time than Prioritized VI, it converges to the correct policy while heap based might not converge to the best policy but will offer a very good approximation. It comes down to the trade off between what is more important. If an approximate result works then PVI is a very good choic, otherwise PI.

The best optimal policy in all cases is provided by the policy iteration as it converges to the best policy. Value iteration also has a similar performance and so does row-major. The heap does not always converge to the best policy.

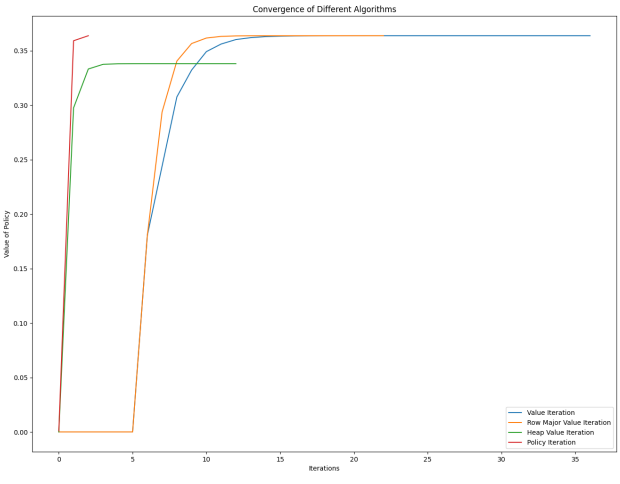

Figure 9: Line Plot of Value vs Iteration for start state for Small Map($\epsilon = 1e - 9$)

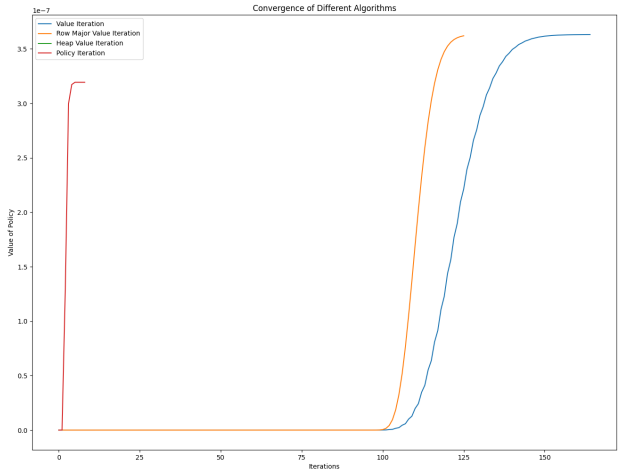

Figure 10: Line Plot of Value vs Iteration for start state for Large Map($\epsilon = 1e - 9$)

From the two plots we can clearly see that the Values for Heap and Policy converge much before the other methods. In the smaller map, policy iteration converges even before the heap updates, but in the larger map, heap method requires fewer updates of the start state, but the value is far off than the value found by PI or other VIs (as hypothesised above).

3 Analysis

Using Vanilla Value iteration for all analysis

3.1 Living Reward Analysis

$\gamma = 0.9$, $hole\_reward = 0$, $goal\_reward = 1$, $p = 0.8$, tol = 1e-9

3.1.1 Living Reward = -0.1

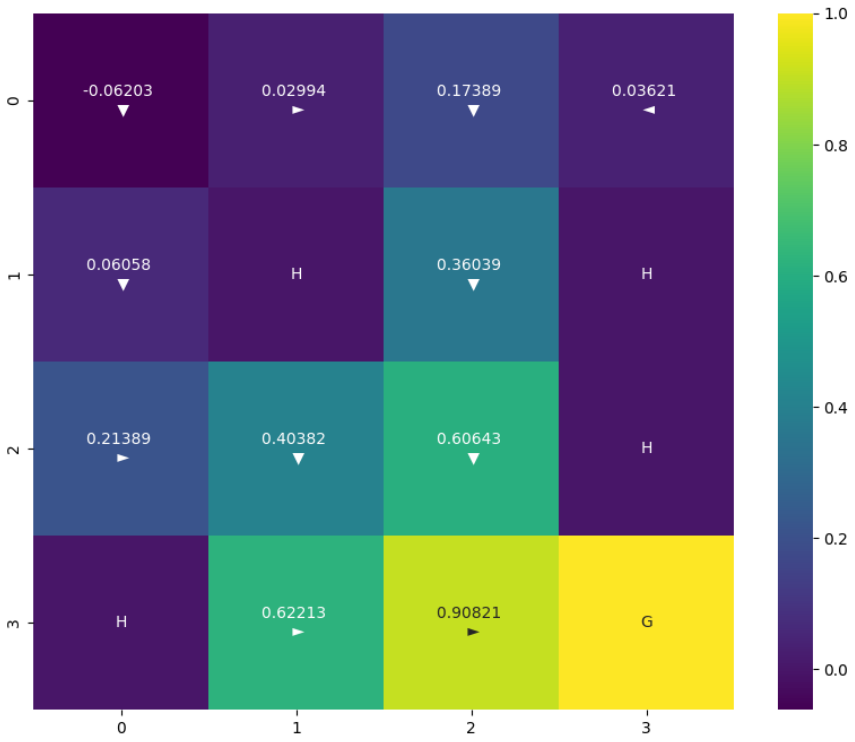

Figure 11: Heatmap for Living Reward = -0.1($\epsilon = 1e-9$)

We see that the value of initial state becomes negative as reaching from there to the goal state is very costly for it due to negative living rewards. The other states also have reduced values as living is penalized.

3.1.2 Living Reward = -0.9

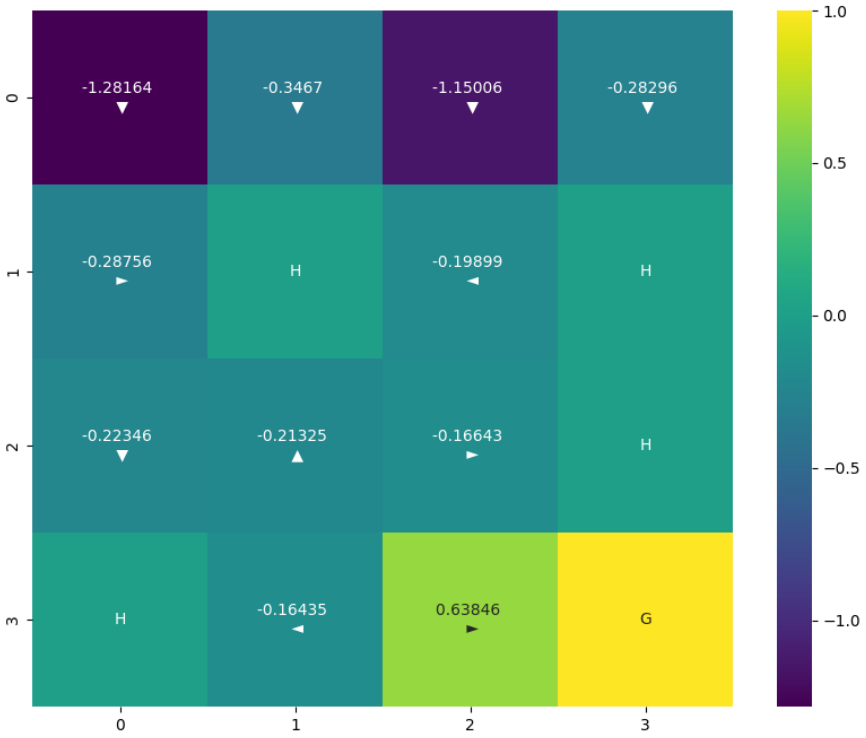

Figure 12: Heatmap for Living Reward = -0.9($\epsilon = 1e - 9$)

We see that the policy significantly changes. When the car is in a state that is adjacent to a hole, it prefers to go to the hole state rather than living and going to the goal state because, living for a longer time gives it tons of negative rewards which when accumulated become higher than the reward obtained by reaching the goal. Hence, by entering the hole which is a terminal state the car provides itself with a negative reward but of significantly smaller value than it would have received on exploring and finding the goal state.

3.2 $\gamma = 0.999$ and Living Reward = +0.001

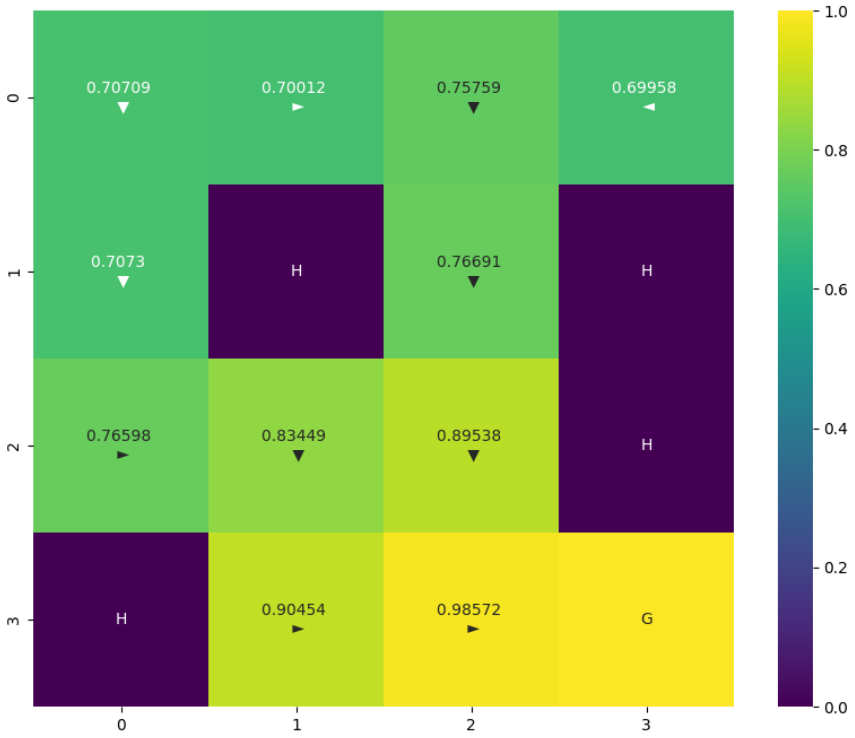

Figure 13: Heatmap for Living Reward = 0.001 and $\gamma = 0.999(\epsilon = 1e - 9)$

The values of the states have increased substantially as the car gets reward for living as well. Also, because of the increased gamma, the discount is less and as a result the discounted expected reward increases.

An interesting case is when we increase the living reward.

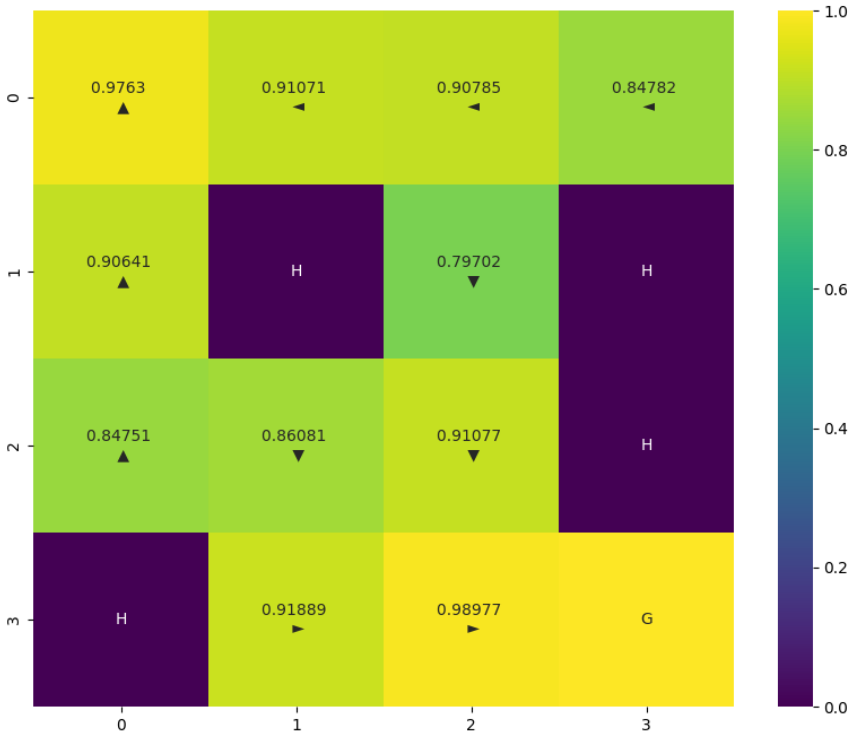

Figure 14: Heatmap for Living Reward = 0.01 and $\gamma = 0.999(\epsilon = 1e-9)$)

We can see that most of the states in the top two rows prefer to live indefinitely rather than enter a hole or reach the goal state. This is a combined effect of higher gamma and higher living reward. The robot increases it utility substantially by living forever. It gets more reward than it would have received on going to the goal state and ending it's journey in a finite amount of time.

3.3 $p = \frac{1}{3}$

$\gamma = 0.9$, $hole\_reward = 0$, $goal\_reward = 1$, tol = 1e-9

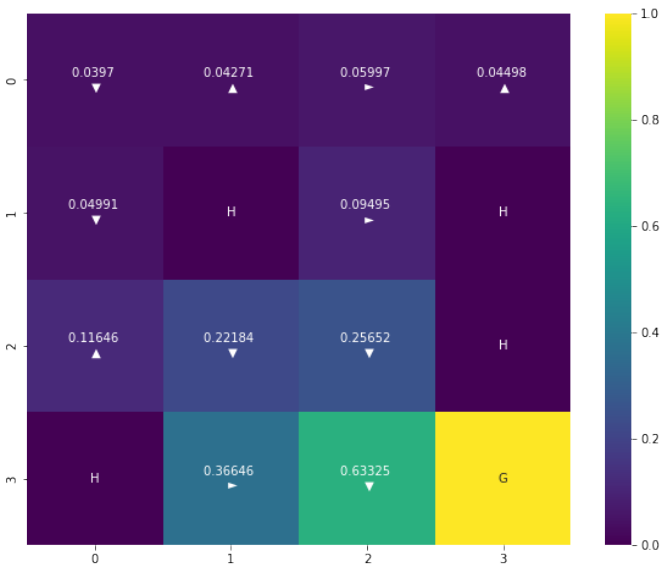

Figure 15: Heatmap using Value Iteration $p = \frac{1}{3}(\epsilon = 1e - 9)$

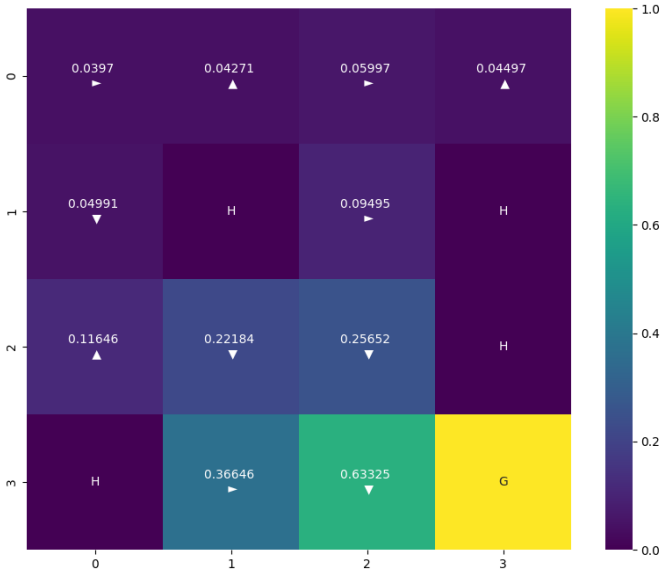

Figure 16: Heatmap using Policy Iteration $p = \frac{1}{3}(\epsilon = 1e - 9)$

The policy iteration and value iteration give almost identical solutions. The policy for initial state differs, but both of the policies will end up in the robot indefinitely moving in a region. For value iteration the robot moves down, down, up, down and oscillated between (1,0) and (2,0). For PI it goes right and up and stays in the up state.

The states near the goal have values that take the car towards the goal, but even those values are less, as the execution of policy is very random.

As we would expect, because of higher uncertainty the path of the car is also uncertain and in states far from the goal, where the information of goal does not reach effectively due to the said uncertainty, the motion is even more haphazard.