# Markov Decision Processes
# CS 7641 Machine Learning Assignment 4

# Siddharth Agarwal, GTID: sagarwal311


## Markov Decision Processes

Markov Decision Processes (MDP) are mathematical frameworks that helps to make decisions on a stochastic environment. Goal of solving a MDP problem is to find the optimal policy that maps each of the state in the environment to some optimal action. Most important parts of MDP processes are:

1. State, S: States are assumed to be complete or Markovian state because they represent all the necessary information that is required from the past to make good decision.
2. State Transition Model, T(s, a, s'): It represents the probability to go from state s to state s' when action a is performed.
3. Actions, A(s): Things that can be performed on a particular state s.
4. Reward, R(s, a, s'): Scalar value that the agent receives on transitioning from state s to s' on performing a given action a.
5. Policy, $\Pi(s) \rightarrow a$: Goal that maps states to optimal actions
6. Optimal Policy, $\Pi^*(s) \rightarrow a$: Policy that maximizes the expected rewards.

In order to solve MDP many planning algorithms can be used such as Value Iteration, Policy Iteration. Given that we know state transition model, and rewards, we need to find optimal policy that maximizes the discounted rewards.

**Value Iteration:**

Bellman's Equation: $U(s) = R(s) + \gamma \, max_a \sum_{s'} T(s, a, s')U(s')$

In Value Iteration we start with arbitrary utilities, update the utilities based on the utilities of the neighbouring states, and repeat the steps until convergence. Once we get the optimal utility function, we get the optimal policy from the optimal utility.

**Policy Iteration:**

In policy iteration, we start with random guess of the policy, then given the policy, Utility is calculated, and finally policy is updated by searching over all the actions using the equation shown below and then repeating. In each iteration of Policy Iteration utility is calculated using the equation

$$U_t(s) = R(s) + \gamma \sum_{s'} T(s, \Pi_t(s), s')U_t(s')$$

These are set of n linear equations in U. Thus, Utility for a state can either be calculated by solving linear equations or by iterating with random utilities and then converging. The key difference between Value Iteration and Utility Iteration is that in each step of Value Iteration both Policy and Utility are variable, while for each step of calculating utility for Policy Iteration, policy is fixed. So, the step for calculating utility in the value iteration is much more

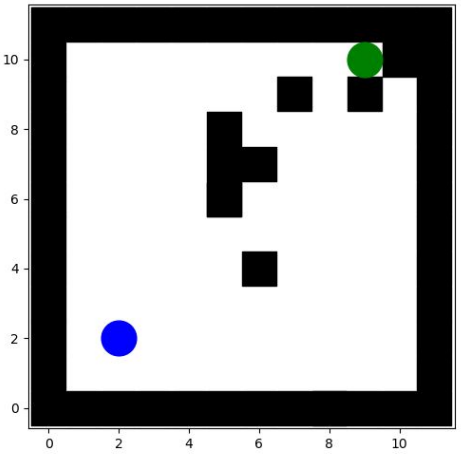
computationally expensive than the step for calculating utility in Policy Iteration. For improving the Policy following equation is used

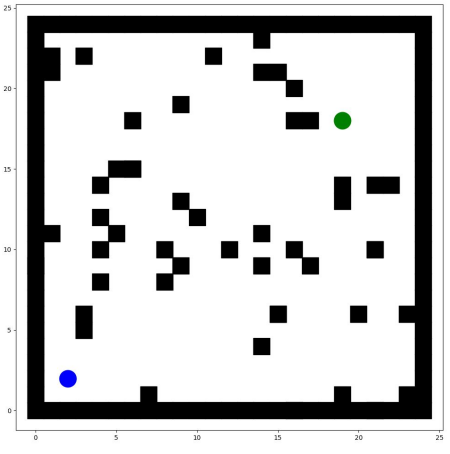
$$\Pi_{t+1} = argmax_a \sum T(s,a,s')U_t(s')$$

## Two MDP Problems:

Two MDP problems I have chosen are related to Robotics, where the task of the robot is to avoid the obstacle and the walls and reach the goal state from the start state. These are extremely interesting problems to me as I am interested in Path Planning in robotics and have used other path planning algorithms such as A* for finding optimal path of the robot. There can be many examples of the robot moving in the grid world. For example, idea of mobile robots serving in a restaurants, or a self driving car. But in all of these cases the environment is dynamic and it keeps changing with time. For purpose of this assignment I have taken grid world as the occupancy grid map with static obstacles and the robot should move from the start location to goal location avoiding these random obstacles. Map for both the problems are shown below. Map 1 has 144 states, while map 2 has 625 states. Start location of the robot is shown with blue state and goal location is shown with green state. Since the number of states in the map 1 are less in number, the size of each state is large as the map size is same. Configurations of the grid world are mentioned in the table below:

| Parameters | Map 1 | Map 2 |
|---|---|---|
| No. of rows | 12 | 25 |
| No. columns | 12 | 25 |
| No. of obstacle states | 8 | 44 |
| Correct Probability | 0.9 | 0.95 |
| Wrong Probability | 0.05 | 0.025 |
| No. of actions | 4 | 4 |
| Goal Reward | 100 | 1 |
| Obstacle Reward | -500 | -15 |
| Stay Reward | -3 | -0.03 |
| Discount | 0.98 | 0.98 |



Obstacle states are randomly chosen within the map. Correct probability is the probability of transitioning that the robot will move in the direction of the action. And wrong probability is the probability that robot move to the state which is not in the direction of the action. There are total 4 actions : Right, Top, left, and bottom. Other diagonal actions are ignored for the purpose of simplicity. Goal Reward is the reward that the robot will collect if it reaches the goal state, while obstacle reward is the reward that the robot will receive it hits any obstacle or the walls. Finally, stay reward is the reward that the robot will receive when it just moves to the next unoccupied state. Since, one of the side goal is also to absolutely make sure that the robot doesn't hit the obstacle or the wall. Therefore, reward when the robot hits the obstacle or the wall is much higher in magnitude.

To solve these problems Value Iteration, Policy Iteration and Q- learning algorithms were used. First, experiments for Value Iteration and Policy Iteration are presented in this paper.

Convergence table showing the iterations it takes to converge for Policy Iteration and Value Iteration algorithms is shown below. Convergence method is mentioned in the brackets in the table.

| | **Map 1** | **Map 2** |
|---|---|---|
| **Value Iteration (Epsilon optimal policy)** | 831 | 650 |
| **Policy Update - Policy Iteration (No change in Policy)** | 9 | 12 |
| **Value Update - Policy Iteration (Epsilon optimal policy)** | [704, 827, 828, 831, 831, 831, 831, 831, 831] | [461, 650, 650, 650, 650, 650, 650, 650, 650, 650, 650] |

Here, Epsilon optimal policy is obtained when variation (i.e. absolute of maximum policy change) is less than threshold (= epsilon*(1 - discount)/discount).

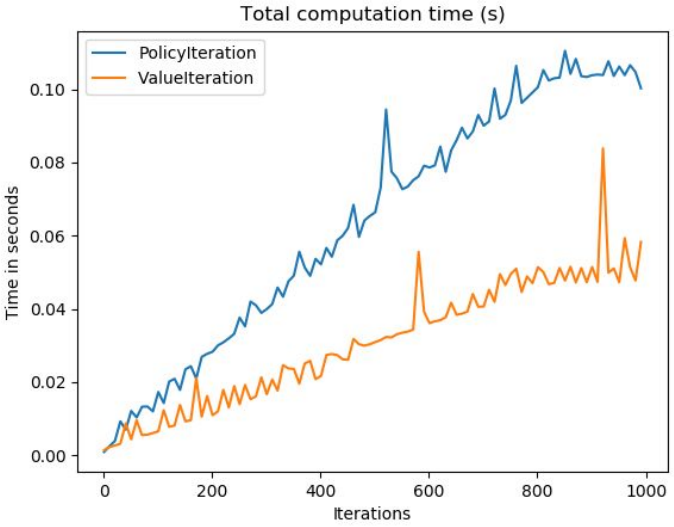
First row of the convergence table shows the total iterations it takes to converge for Value Iteration. While for policy iteration, second row provides the number of policy update iterations and third row shows number of value update iterations for each value calculation for every policy update. It can be seen for larger map, number of Policy update iterations increases while value iterations decreases. Since Value Iteration has a single update in the algorithm while Policy Iteration is nested update type algorithm where Value calculation for each policy update is itself an iterative algorithm, and the convergence method for used in MDP toolbox for both these algorithms are different these two algorithms cannot be compared in terms of iterations. Also, total convergence time for two algorithms is shown in the table below.

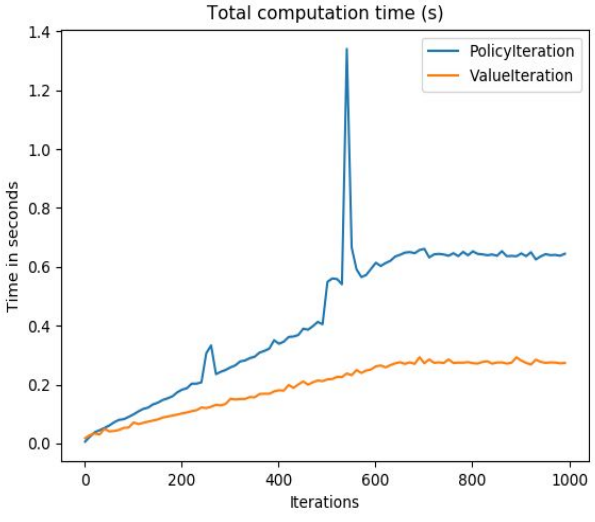
It can be seen that average value update time in Value Iteration algorithm is approximately 10 times higher than average value update time in Policy Iteration algorithm. It is obvious as

in Value Iteration for each iteration it searches over all policies and value together while policy is fixed in Policy Iteration value update. But overall convergence time for Value Iteration (when epsilon optimal policy is obtained) is less than overall convergence time for Policy Iteration (when Policy is not changing). This is because it takes more time to update policy in Policy Iteration and iterative process for value is calculated every time policy is updated while this happens only once in Value Iteration.

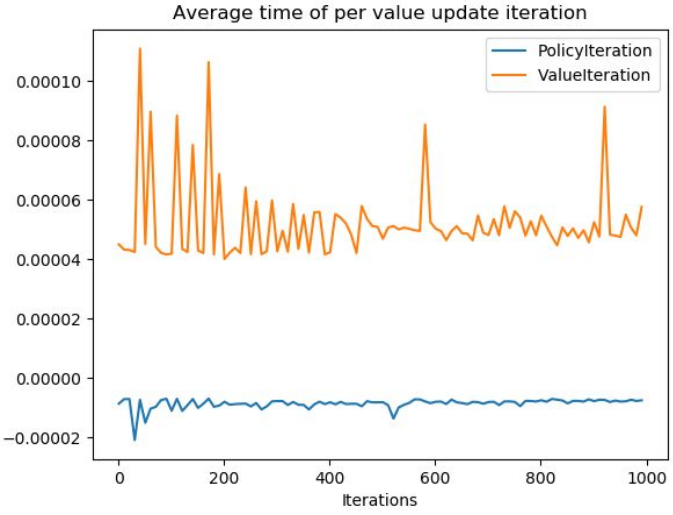
| | **Map 1** | **Map 2** |
|---|---|---|
| **Total Time for Value Iteration** | 0.05156 | 0.2735 |
| **Total Time for Policy Iteration** | 0.10377 | 0.644 |
| **Average Value Update time for Value Iteration** | 0.00005276 | 0.0003671 |
| **Average Value Update time for Policy Iteration** | 0.000007109 | 0.0000733 |

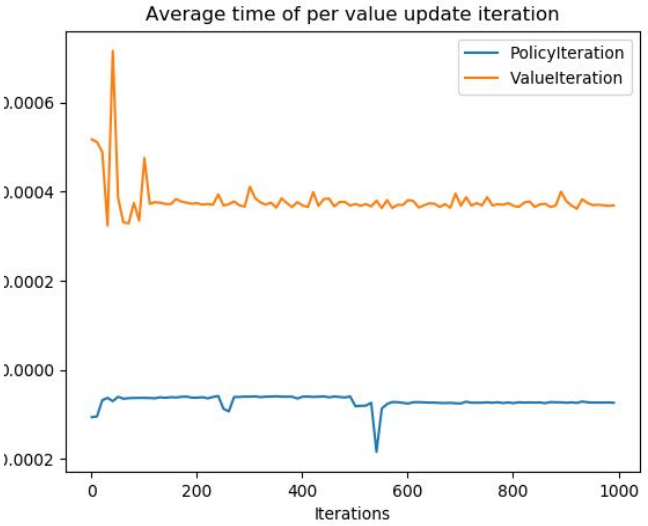
The two plots below shows the total computation time for Policy Iteration and Value Iteration with varying iterations (iterations for value update) for both the Map 1 and Map 2. The plot on the left hand side is for Map 1 and plot on the right hand side is for Map 2. As in the convergence case, policy iteration takes more overall time. Also, as the number of states are increased total computation time increases.



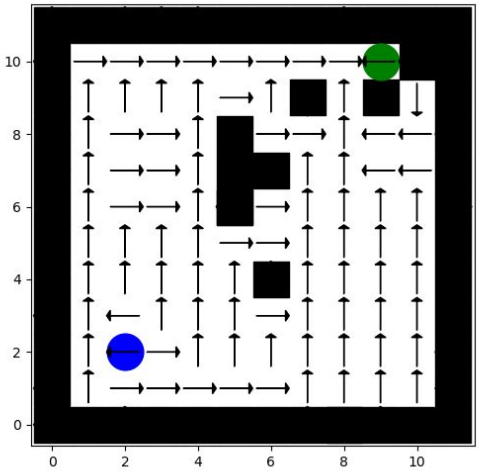

The two graphs below shows the average time per value update iteration varying with maximum iteration for Map 1 and Map 2 respectively. As explained above it can be seen that average time for Policy Iteration is way less than average time for Value Iteration. But the

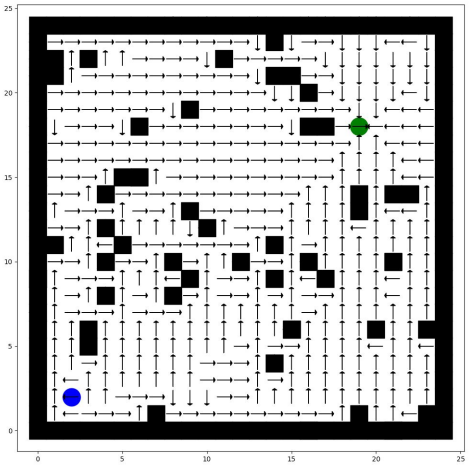


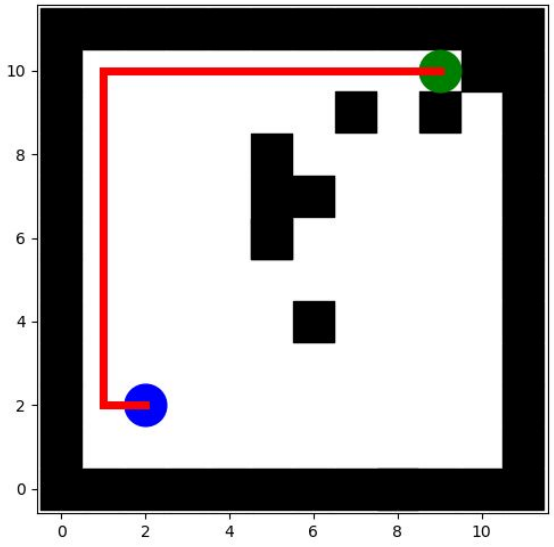

time increases with number of states in the map, since it takes more computation to arrive at optimal values. This average time almost remains constant with increasing iterations.

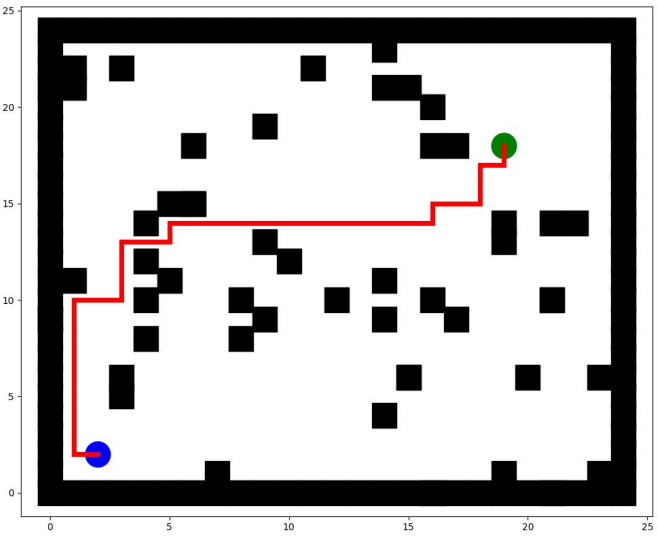
Following two plots are the plots of final converged policies for Map 1 and Map 2 respectively. Arrows in the plot shows the optimal action obtained for each state. It can be seen that policy directs the robot to move from start location to goal location. Same policy is obtained from both Value Iteration and Policy Iteration algorithm. Thus, these two algorithms are converging to same answer though they take different time to converge.



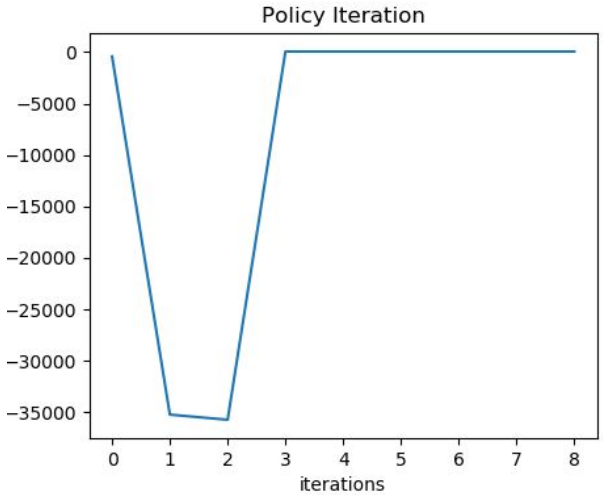

Plot below shows the final path traced by the robot to reach the goal location using optimal policy for both the maps. It is same for both the algorithms. It shows that robot tries to avoid obstacle as much as possible because the obstacle reward is much higher in negative direction.

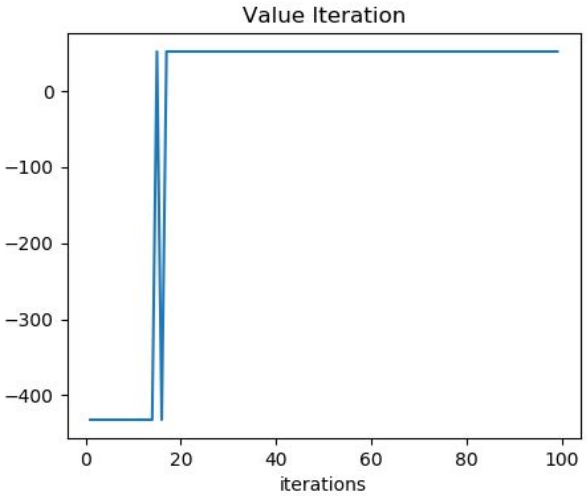


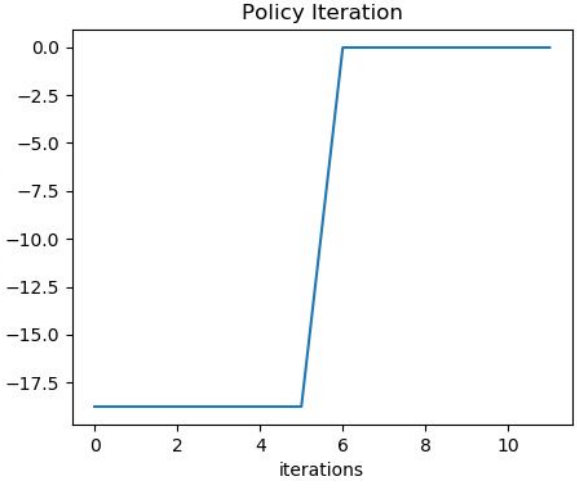


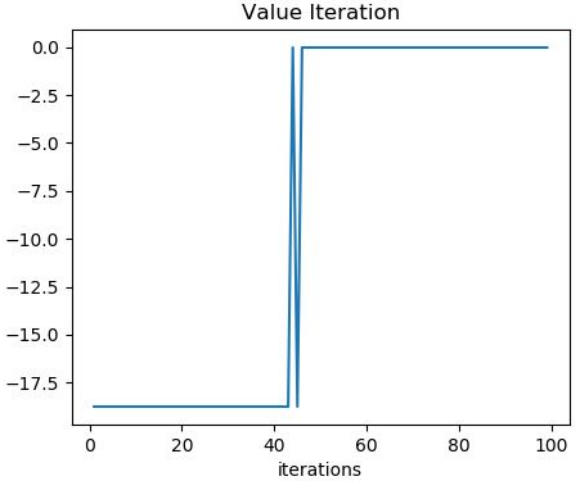
Finally, total reward that the robot collects while moving from start state to goal state is plotted with iterations. For plotting this, policy at every step of value update in Value Iteration and policy update in Policy Iteration is stored. Each stored policy may not be optimal but the last policy stored will be optimal.

Reward collected for Map 1 and Map 2 for Policy Iteration and Value Iteration with varying iterations is shown below. First two plots are for Map 1 and next two plots are for map 2.



For both the maps,in Value Iteration it is visible that reward doesn't change after few iterations (18 for map 1 and 45 for map 2) even though value hasn't converged yet below the epsilon threshold. Whereas, for policy iteration reward remains constant after 3 policy update iteration for map 1 and 6 policy update iterations for map 2. **Since, reward is a function of value, it is evident that policy converges very fast for both the algorithms even though value is changing because the output of the operator 'argmax' doesn't change**.

## Reinforcement Learning Algorithm: Q-Learning

Q-Learning is a reinforcement learning algorithm used for calculating optimal policy in the sense that the expected value of the total reward return over all successive steps, starting from the current state, is the maximum achievable. Q-learning is model free algorithm hence it does not require state transition model. It only requires transitions <s,a,s',r>. Q-learning stands for 'quality' of action taken in a given state.

Before the learning begins, Q is initialized to some arbitrary values. Then, at each time step agent selects an action $a_t$, observes a reward $r_t$, enters a new state $s_{t+1}$ (depending on the previous state and action) and updates Q. Q update is similar to Value Iteration update using the weighted average of old value and new information:

$$Q(s_t, a_t) \leftarrow (1 - \alpha).Q(s_t, a_t) + \alpha(r_t + \gamma . max_a Q(s_{t+1}, a))$$

An episode/iteration ends when state $s_{t+1}$ is the terminal state.

Following are the hyperparameters used for finding optimal policy using Q-learning algorithm. Parameters are same for both map 1 and map 2.

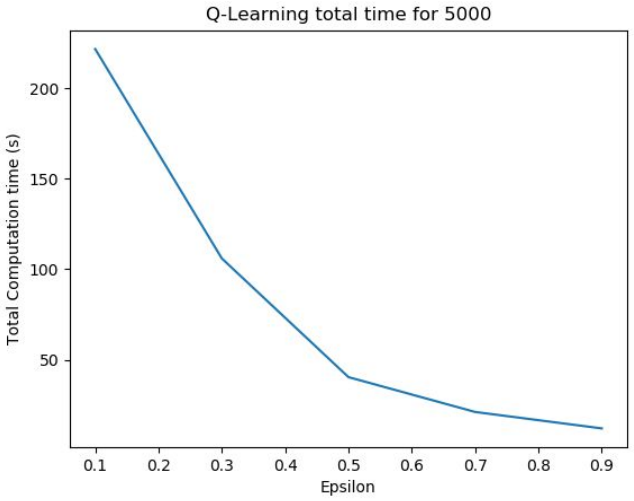
| Parameters | Values |
|---|---|
| **Iterations/ Episodes** | 5000 |
| **alpha** | 0.3 |
| **epsilon** | [0.1, 0.3, 0.5, 0.7, 0.9] |
| **discount** | 0.98 |

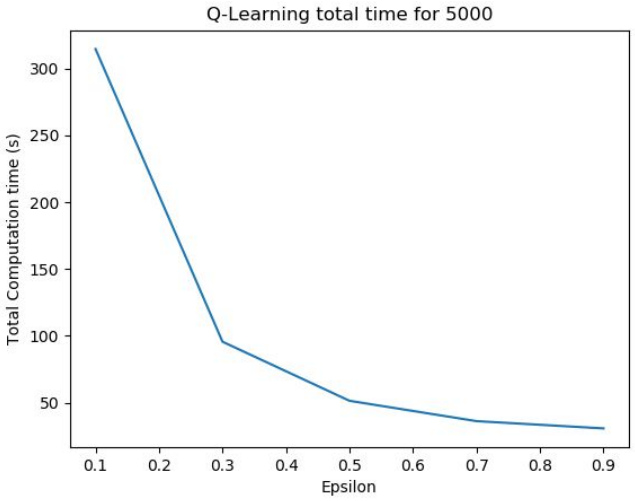
In each iteration/ episode random state is chosen and then action is chosen to give the next state. Within each episode in every iteration action is selected until the next state is the terminal or the final state. Q learning is class of algorithms and it can be subdivided into different algorithms based on how action is selected. One of the algorithm is epsilon-greedy algorithm. In this algorithm for selecting an action, a random number between 0 and 1 is generated and a random action is selected if the random number generated is greater than or equal to epsilon value and other wise it is selected based on maximum Q value. This is called exploitation- exploration dilemma in Q learning algorithms. It is exploitation when it selects the action based on maximum Q value, while it is exploration based on random

number. This parameter epsilon can be varied with iterations/ episodes as well. Intuitively, early in the life, agent should select more random selection to encourage initial exploration, while as time progresses it should act more greedily.
For purpose of this assignment epsilon-greedy Q learning algorithm is used and epsilon is varied.

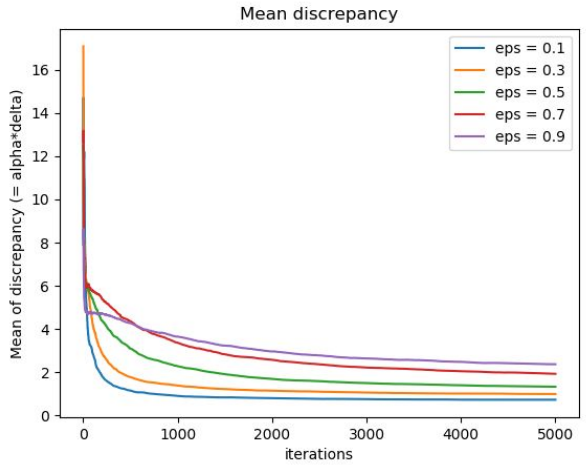
Alpha is the learning rate as shown in the Q update equation.

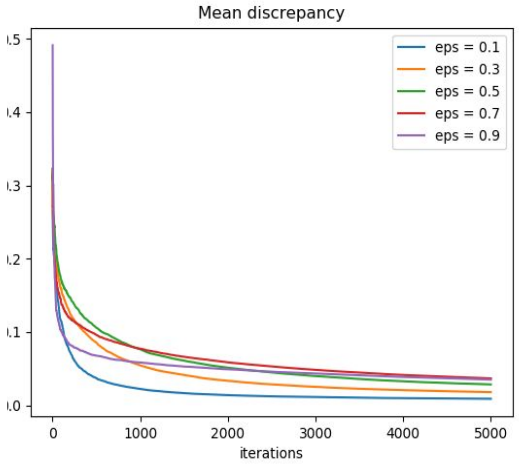
Total computation time for Q learning algorithm is plotted with varying epsilon for both map 1 and map 2 and the plots are shown below. It can be seen that with increasing epsilon, total computation time decreases rapidly. The reason for this is that when epsilon is low, agent takes much more iterations to reach the goal state within each episode because only random actions are being selected and no learning is happening. While when epsilon is high, random action is selected at only few instances and more learning is happening and agent reaches the goal state fast.
If we compare the total computation time for Q learning with Value Iteration and Policy Iteration, it is much slower.



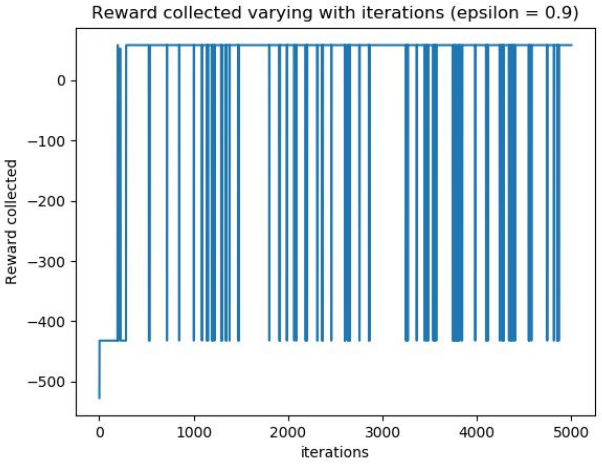



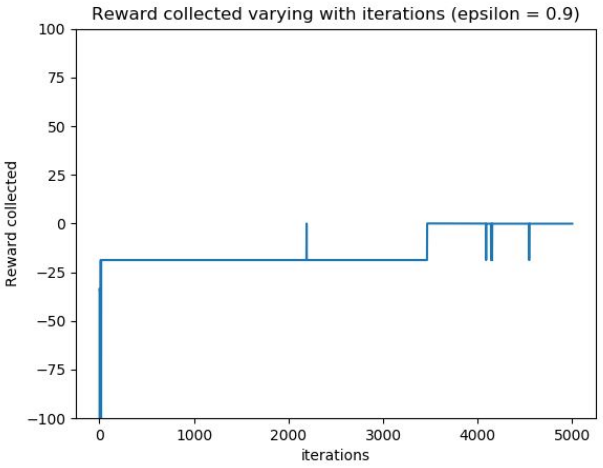

Next, mean discrepancy with varying episodes/ iterations is plotted for both the maps. Mean discrepancy is the mean of alpha*delta for all the iterations in one episode, where delta is the $r_t + \gamma . max_a Q(s_{t+1}, a) - Q(s_t, a_t)$ which represents the new information that is added to the Q. With increase in epsilon, mean discrepancy increases because agent learns and more useful information is added every time. While, mean discrepancy decreases with number of iterations/episodes as the scope for adding new information decreases as agent converges towards the optimal Q.



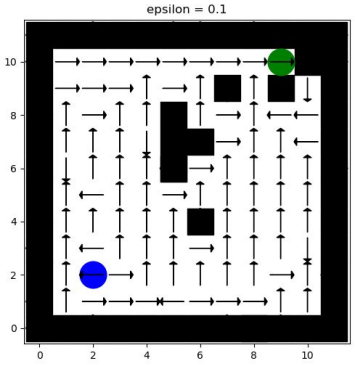



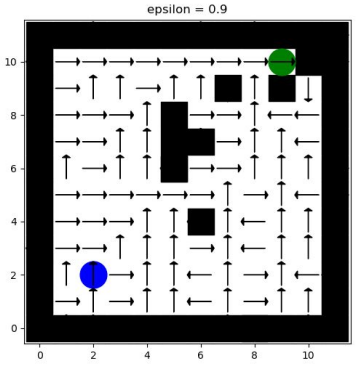

Next total reward collected is plotted for each episode for eps = 0.9 for both map 1 and map 2. It can be seen that agent collects more reward by following the policy that is obtained after higher number of episodes/iterations. This is because when iterations are more, policy obtained is better learned than the policy obtained earlier.

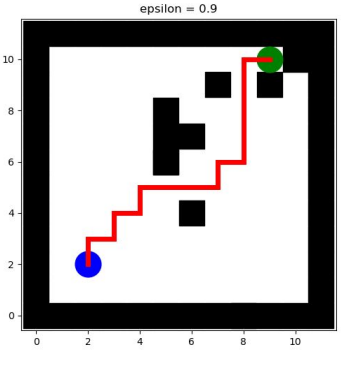


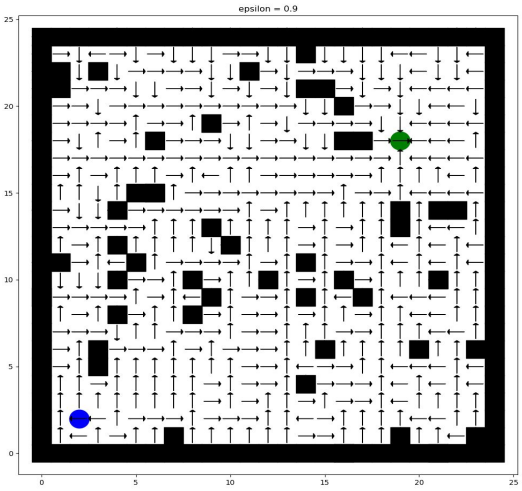

Finally, policy converges for epsilon value of 0.3, 0.5, 0.7 and 0.9 for map 1 while it does not converge for epsilon value of 0.1. This is because using epsilon 1 agent will take random action most of the times. And for map 2, policy only converges for epsilon value of 0.7 and 0.9. Since, map 2 has greater number of states more iterations are needed to reach to a convergence point. Policy map and path traced is shown below for map 1 (epsilon = 0.1, and 0.9). It can be seen that there's no path from the start location to goal location for epsilon = 0.1.

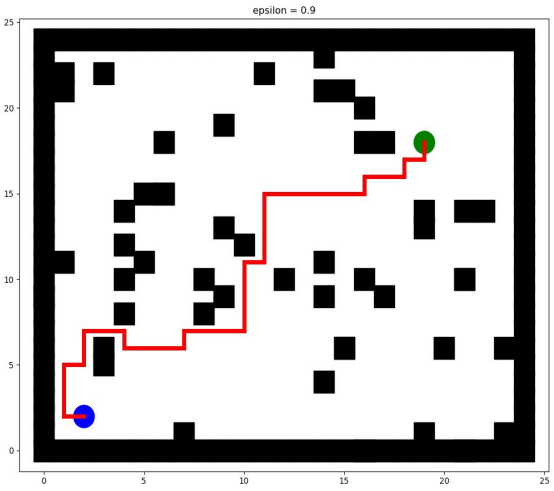


Next, policy map and path traced is shown for the map 2 for epsilon = 0.9.




## Conclusions:

1. I have used two grid world path planning MDP where purpose of the robot is to avoid obstacle and reach the goal state. One with smaller number of states and one with larger number of states. These MDPs are interesting to me as I am interested in path planning in robotics.
2. When these problems are solved using Policy Iteration and Value Iteration, each value update in policy iteration takes less time than each value update in Value Iteration because policy is constant for value update in Policy Iteration, while both policy and value is varying for value update in Value Iteration.
3. Policy Iteration takes less iterations to converge (when policy is not changing) as compared to value Iteration (epsilon optimal policy), because policy often converges faster as compared to value even when value is changing. This is because policy is the output of the operator ‘argmax’ which doesn’t change.
4. They converge to same answer in terms of policy in my experiments.
5. If we increase the number of states that is go from map 1 to map 2 total computation time increases, number of policy iterations increases, number of value iterations decreases, reward converges after more number of iterations.
6. I have used Q-learning (epsilon-greedy) algorithm as the reinforcement learning algorithm. I have varied epsilon (exploitation - exploration variable) to measure the performance.
7. With increasing epsilon total computation time decreases becauses agent learns the Q and reaches the goal state quickly when epsilon is large as very few action chosen would be random.
8. Q learning is much slower than Value Iteration and Policy Iteration.
9. Mean discrepancy decreases decreases with more episodes or iterations because scope of learning new information by Q decreases as agent converges towards optimal Q.
10. With increase in epsilon, mean discrepancy decreases as randomness is reduced and agent learns more information. Thus, more new information in Q is added.