Markov Decision Processes(MDP)

Anusha Challa (achalla8)

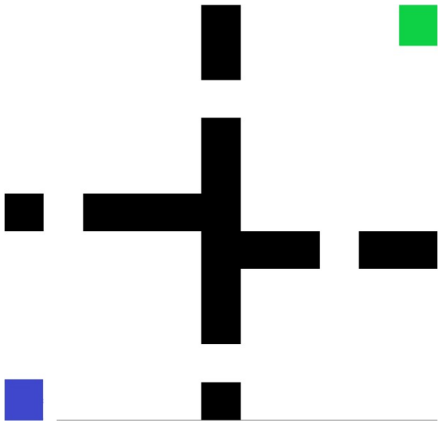
1. **Introduction**: In this assignment, Value Iteration (VI), Policy Iteration (PI) are used to solve two interesting Markov Decision process(MDP) Problems. In addition to this, the Reinforcement Learning (RL) algorithm, Q-Learning is used to is used to solve the same set of MDP problems and the results are analyzed. BURLAP library, which is provided as a coding resource has been used to code the solutions
2. **Markov Decision Processes(MDPs) Chosen**: Markov decisions processes describe an environment for Reinforcement learning. Markov decision processes (MDP) are a mathematical framework to model tasks or problems that need a series of sequential decisions under uncertainty. In an MDP, agent interacts with an environment that has a set of possible states. The agent is supposed to decide the best action to take based on the current state. MDP contains the below components. Markov's principle states that "The future is independent of the past given the present"

States:	S
Transition Model:	T(S, a, S') ~ P(S' | S,a)
Actions: A(S), A	
Reward Model:	R(S), R(S,a), R(S,a,S')

Policy:	$\pi(S)$ -> a
Optimal Policy:	π^*

States comprise of set of tokens that represent all the states that the agent can be in. Model is a Set of rules that define the possible transitions and their corresponding probabilities. Action is a set of possible actions that the agent can take. Reward is a real-valued number that indicates the reward that the agent would receive for being in a state. Rewards can be both positive and negative. Policy is a solution to MDP. Policy provides mapping from States to Actions. It defines the action that can be taken at each possible state. There can be multiple policies that solve an MDP. Of these multiple policies, an optimal policy is a policy that always chooses the action that maximizes the utility of the current state. In this assignment, 2 MDPs have been chosen and solved using Policy Iteration, Value Iteration and Q-Learning algorithms. Below is a description of the MDPs chosen

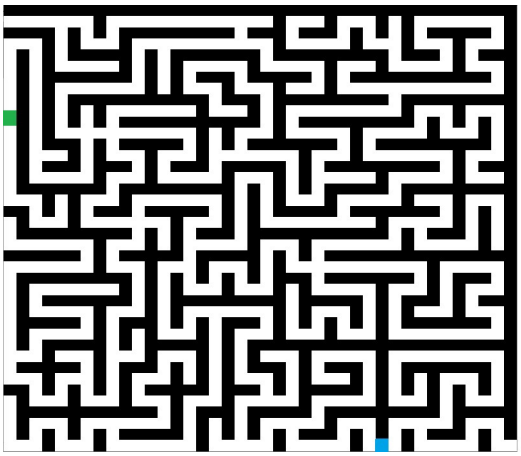
2.1. Four-Room Problem:



The four-room problem has a 11x11 grid, which is divided into 4 sections. Four-Room problem have 121 possible states. This has been chosen as an easy grid world example. The Each cell in the grid corresponds to a state that the environment can take. Blue square indicates the starting state and Green square indicates the goal state. Black squares are walls, indicate the states that the environment cannot take. Four stochastic movement actions available: Left, Right, Up and Down. The probability that the agent would move in the direction in which it is intended to move is 0.8. The probability that the agent would move in any of the other three directions is 0.2; The probability for each direction being 1/15. When the next state of the agent, through an action is a wall, agent remains in the same state.

Why is this problem interesting? This problem is interesting because, when an agent is in a wrong room, there are only 2 states that can get the agent into another room. It would be interesting to see how the policy directs the agent towards the goal sate. There are multiple states in the environment in which any action would result in equal reward and Utilization. It would be interesting to see how the target policy actions in these states. Example: 4 squares in each of the

corners have the same reward for moving in any of the 2 directions, up and right



2.2. Maze Problem: Maze is a hard grid world example. The maze that was chosen for analysis is of the size 35x35. It has 1225 states. The blue ball in the adjacent picture indicates the starting state and the green ball indicates the goal state. Black squares represent walls. Four stochastic movement options available: Left, Right, Up and Down. The probability that the agent would move in the direction in which it is intended to move is 0.8. The probability that the agent would move in any of the other three directions is 0.2; The probability for each direction being 1/15. When the next state of the agent, through an action is a wall, agent remains in the same stat

Why is this problem interesting? This problem is interesting because, of the high number of walls in the maze. Because of the high number of walls, agent's path from start state to end state becomes highly restrictive. Also, there is a high chance that the agent gets stuck in a non-terminating state forever.

3. Solving the Markov Decision Processes (MDPs) using Value Iteration and Policy Iteration

The solution of an MDP is a policy that maps states to the best action that can be taken at that state to maximize the reward. Optimal policy is the one that returns the maximum utility given that the optimal policy is followed. Utility of a state is defined using Bellman Equation:

$$U(S) = R(S) + \gamma \max_{a} \sum_{S'} T(S, a, S')U(S')$$

If there are n states that the agent can assume, then here will be n Bellman equations to solve and there will be n unknowns. However, Bellman since the equation has a max operation, it is not a linear equation. Hence, we reinforcement learning algorithms are used to find the solution for optimum policy.

3.1. Value Iteration: The first algorithm that was explored is value Iteration algorithm. In Value iteration algorithm, the goal is to find the true Utility for each state. If the Utility of each state is known, it would be simple to choose an action that maximizes the utility. Immediate reward of each state is known. The true value or Utility of each state is the immediate reward for the state plus the expected discounted reward, if the agent has chosen the optimal action for each state from that point on. The value for each state can depend on its neighbors' values. Both the MDPs that we have considered are acyclic. Hence, we can use value iteration to solve. Value iteration algorithm has the following steps:

i. Assign each state a random value
ii. Calculate the new utility of each state based on the neighbor's utility
iii. Utility for each state is updated using the below equation: Reward for the state plus the discounted utilities expected given the original estimates for the utility

$$U'(S) = R(S) + \gamma \max_{a} \sum_{S'} T(S, a, S')U'(S'^{T})$$

iv. Repeat steps until convergence. This algorithm is guaranteed to converge within finite number of iterations

3.2. Policy Iteration: In Policy Iteration, instead of finding the value for each state, the algorithm searches the policy space for the policy directly. This can be achieved by modifying the value iteration to iterate over policies. The algorithms start with a random policy, computes utility for each state given the policy, and then revise the policy until it becomes the optimal policy. Given below are steps in Policy Iteration algorithm

i. Create a random Policy π_0 by choosing a random action for each state
ii. For each time step t, Calculate Utility of each state given π_t as $U_t = U^{\pi_t}$

$$U_t(S) = R(S) + \gamma \sum_{S} T(S, \pi_t(S), S')U_t(S')$$

iii. Given these utilities compute the optimal action for each state. This is nothing but computing the new policy π_{t+1}
iv. Repeat the steps until convergence i.e.., until there are no changes in actions chosen for each step

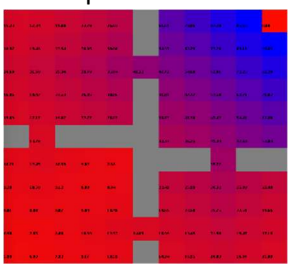
3.3. Parameters chosen for Value Iteration(VI) and Policy Iteration(PI): For both policy iteration and value iteration, the parameters Discount factor (γ), and Rewards R(S) for Terminating and all other steps have been altered to find the best

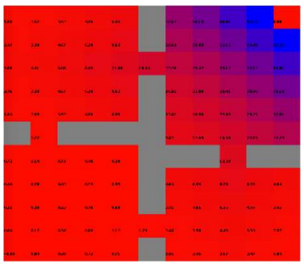
set of parameters. Reward at a time steps is exponentially affected by a factor of γ. TR is the reward that agent gets for terminating states and SR is the reward that agent gets for non-terminating states. In most cases SR is a cost that the agent must pay to move to a state. There is a high level of correlation between TR and SR. The more the value of TR is with respect to SR, the agent gets more reward to move towards the terminating states. Since SR is a cost that the agent must pay, sometimes it de-motivates the agent to move to a state because the reward at terminating state would be lesser than the cost that the agent must pay to take steps to enter a state. Below set of parameter variations have been considered for both Policy and Value Iterations

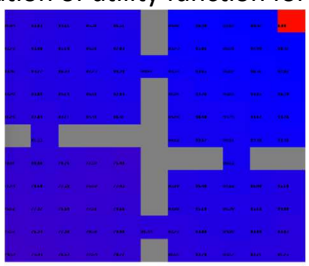
Name of the Parameter	Values on which experiments are run
Discount Factor γ	0.99, 0.95, 0.9, 0.85, 0.8
Terminating Reward (TR)	100, 50, 20, 10, 5
Step Reward (SR)	-1, -0.1

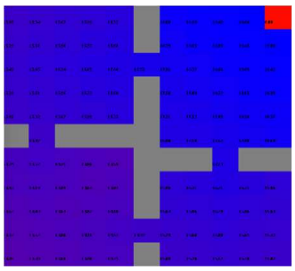
3.4. Four Rooms Problem Solution Analysis for Policy Iteration and Value Iteration Algorithms:

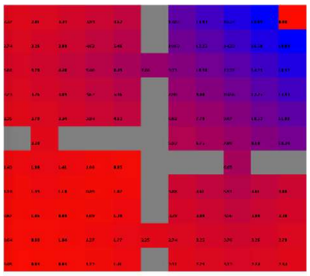
Four rooms problem presents in section 2 is solved using value iteration and policy iteration algorithm and the results are presented in this section. Below are the evaluation of utility function for all states for various combinations of parameters γ, TR and SR. From these ColorBlend Interfaces, it can be observed that that a learning rate of 0.99, TR = 20 and SR = -0.1 are the optimal parameters for are the policy iteration algorithm as they produced the highest reward for each state in the four-room grid world graph. Adjacent plots are the ColorBlend interfaces for policy iteration. The value of a cell of the grid world has been rendered with a color that blends from red to blue in the interface to create an instance of the ColorBlend interface. The numeric values to which the colors are assigned are normalized. 0 is the minimum value and 1 is the maximum. The number assigned to each cell is the output of utility function computed for that state. It can be noticed that the utility function has high value for all states for the parameter combination that has been chosen. For the other combination, the utility function has resulted in negative values in many cases which would discourage the agent from entering a state.

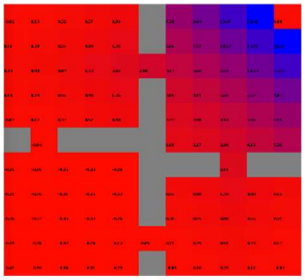

PI: γ=0.8, TR=20, SR=-0.1


PI: γ=0.9, TR=20, SR=-0.1

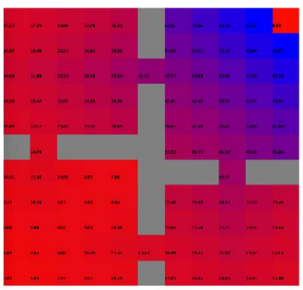

PI: γ=0.99, TR=20, SR=-0.1

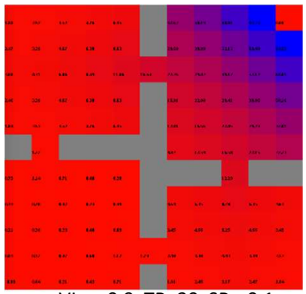

PI: γ=0.99, TR=100, SR=-0.1

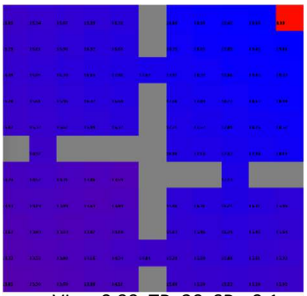

PI: γ=0.8, TR=100, SR=-0.1

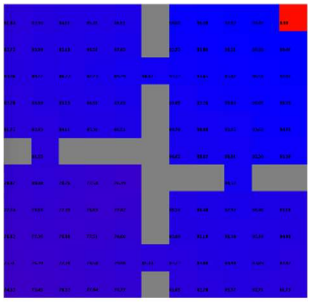

PI: γ=0.9, TR=20, SR=-0.1

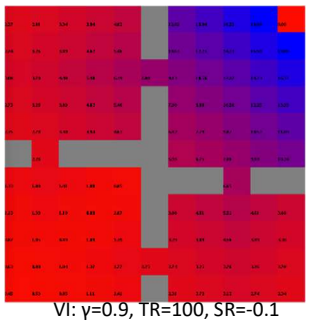
Adjacent graphs are the Color blend interfaces for Value iteration for different combinations of parameters. It can be observed from these color blend interfaces that the best reward for value iteration algorithm can be observed when the parameters values are: learning rate of 0.99, TR = 100 and SR = -0.1. Similar to that of policy Iteration, the value iteration also showcases highest value for reward for the chosen parameter combination. The other parameter combination has less utility value for each state and in some cases the utility is negative as well.

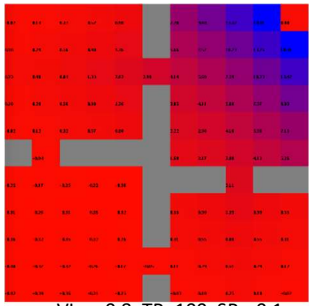

VI: γ=0.8, TR=20, SR=-0.1


VI: γ=0.9, TR=20, SR=-0.1

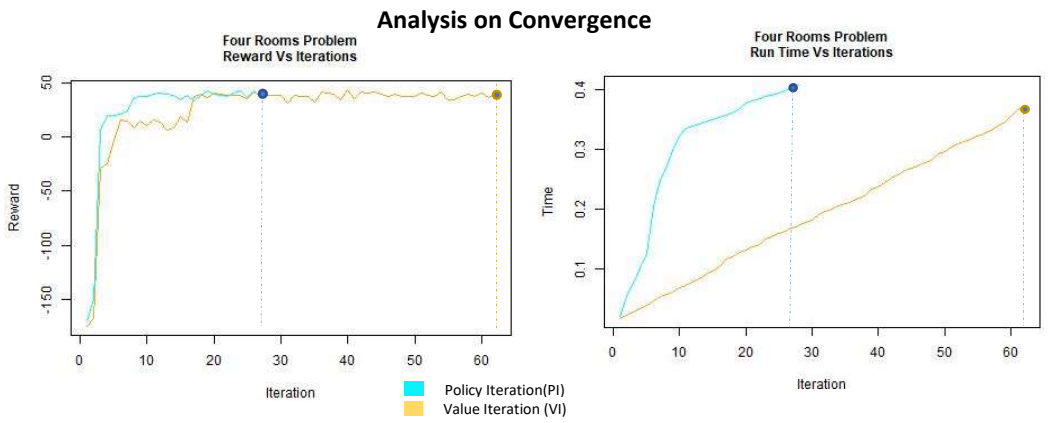

VI: γ=0.99, TR=20, SR=-0.1

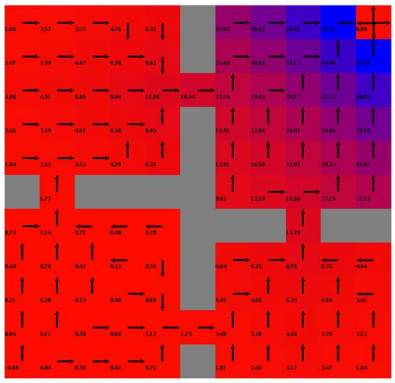

VI: γ=0.99, TR=100, SR=-0.1


VI: γ=0.9, TR=100, SR=-0.1

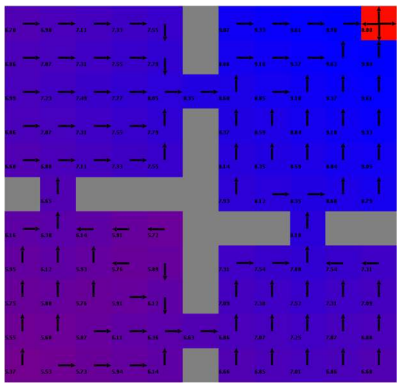

VI: γ=0.8, TR=100, SR=-0.1

Using the above optimal parameter combinations, both the Policy Iteration(PI) and Value Iteration(VI) algorithms are run on Four rooms problem. Number of iterations vs reward is plotted for both Policy and Value Iteration algorithms. From the graphs, it can be observed that it takes more number of iterations for value iteration to converge than it takes for policy iteration. Policy Iteration algorithm has converged at iteration 28 and Value iteration has converged at iteration 62. This outcome can be explained by the fact that it takes less time to scan across policy space to find the optimal policy than it takes to compute utility for each state. On the other hand, the time taken for Policy iteration for each iteration on average is more than the time taken by VI algorithm for each iteration. This can be explained by the fact that policy iteration computes the utility of each state at all iteration based on the new policy. This computation of utility can be time consuming. The reward for both VI and PI starts as negative and reached to approximately 50 for both the algorithms at convergence. Time taken for PI at for 10 iterations is approximately 3 times as that of VI and the gap has reduced as the number of iterations increased. Policy maps obtained for both Policy and Value iteration algorithms for at various iterations until convergence is shown below.

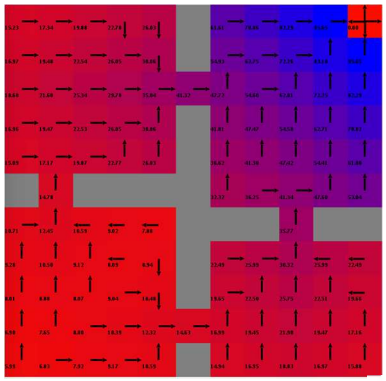




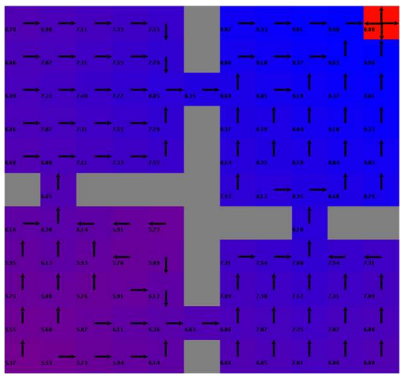
Policy Iteration: Policy map at Iteration 5



PI: At Iteration 28
Policy map after convergence



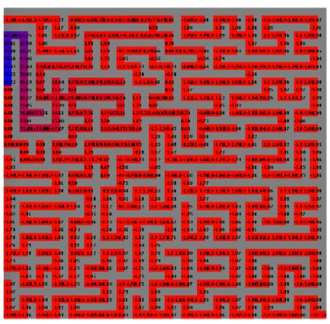
Value Iteration: Policy Map at Iteration 5



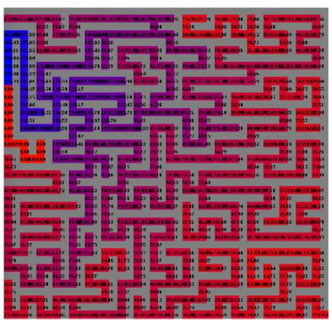
VI: At Iteration 6
Policy map after convergence

From the Color-Blend interfaces above it can be observed that the final optimal policy obtained by both Policy and value iteration algorithms is the same. Value iteration has started with high utility values than policy iteration. However, as noted earlier policy iteration converged faster than value iteration as it searches for the policy directly.

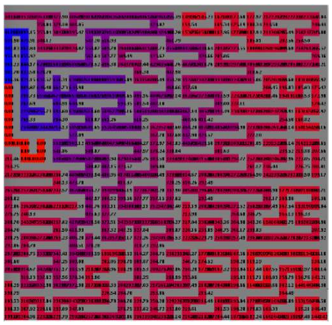
3.5. Maze Problem Solution Analysis for Policy Iteration and Value Iteration Algorithms:



VI: γ=0.8, TR=100, SR=-0.



VI: γ=0.99, TR=20, SR=-0.

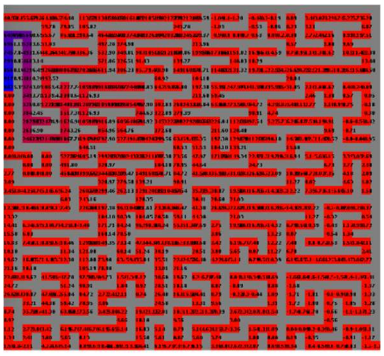


VI: γ=0.99 TR=100, SR=-0.

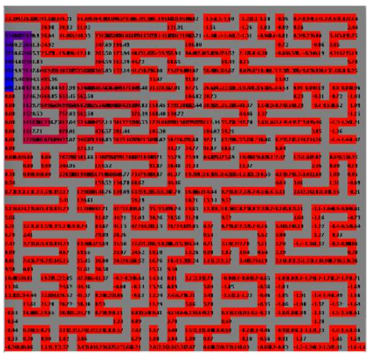
Maze problem that is defined in section 2 is solved using policy iteration and value iteration algorithms. Below are representations of the evaluations of utilizations of all the states in maze problem with different values of the parameters γ, TR and SR. From these ColorBlend Interfaces, it can be observed that that a learning rate of 0.99, TR = 100 and SR = -0.1 are the optimal parameters for are the value iteration algorithm as they produced the highest reward for each state in the maze grid world graph. Adjacent plots are

the ColorBlend interfaces for value iteration. The value of a cell of the grid world has been rendered with a color that blends from red to blue in the interface to create an instance of the ColorBlend interface. The numeric values to which the colors are assigned are normalized. 0 is the minimum value and 1 is the maximum. The number assigned to each cell is the output of utility function computed for that state. It can be noticed that the utility function has high value for all states for the parameter combination that has been chosen. For the other combination, the utility function has resulted in negative values in many cases which would discourage the agent from entering a state.

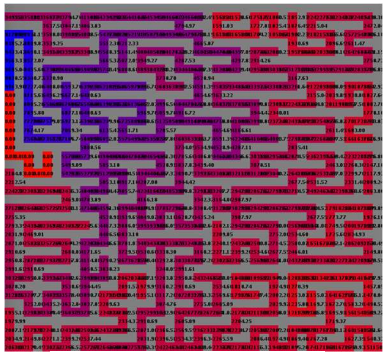
The below graphs are the Color blend interfaces for Policy iteration for different combinations of parameters. It can be observed from these color blend interfaces that the best reward for policy iteration algorithm can be observed when the parameters values are: learning rate of 0.99, TR = 100 and SR = -0.1. Similar to that of policy Iteration, the value iteration also showcases highest value for reward for the chosen parameter combination. The other parameter combination has less utility value for each state and in some cases the utility is negative as well. It can be observed that unlike four room problem, the maze problem chooses the same set of parameters for both policy iteration and value iteration algorithms as they produced the same high value as a result.



PI: γ=0.8, TR=100, SR=-0.1

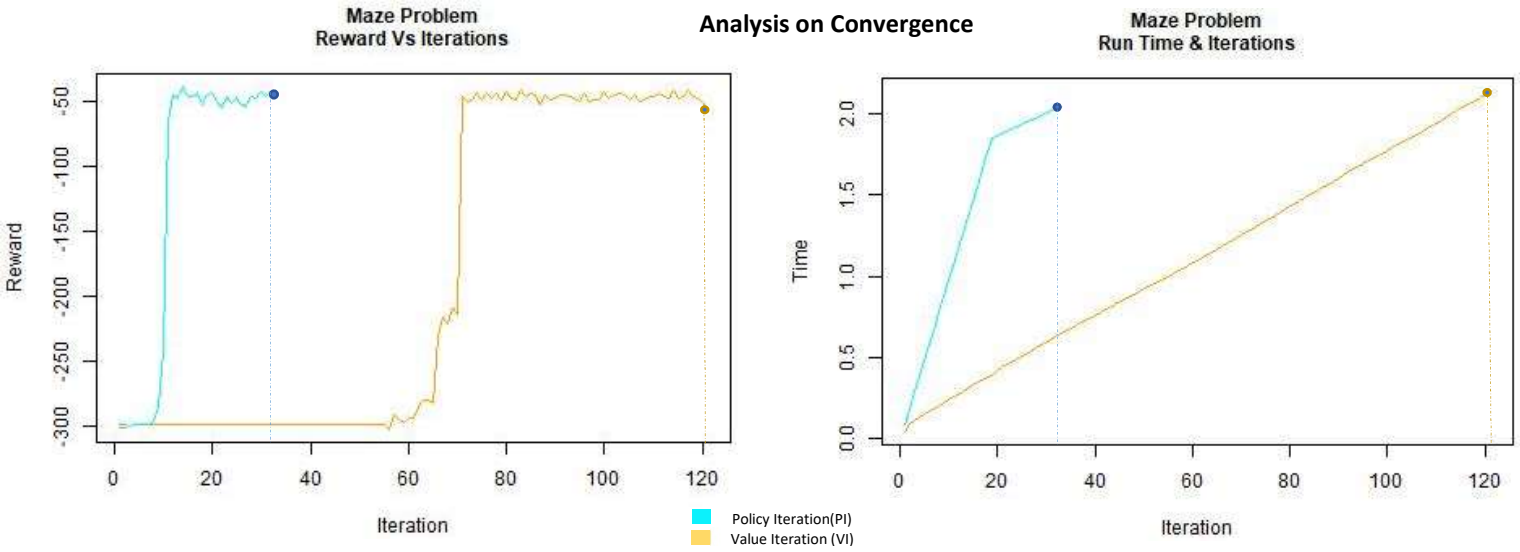


PI: γ=0.99, TR=20, SR=-0.1



PI: γ=0.99, TR=100, SR=-0.1

Using the above optimal parameter combinations, both the Policy Iteration(PI) and Value Iteration(VI) algorithms are run on Four rooms problem. Number of iterations vs reward is plotted for both Policy and Value Iteration algorithms. From the graphs, it can be observed that it takes more number of iterations for value iteration to converge than it takes for policy iteration. Policy Iteration algorithm has converged at iteration 32 and Value iteration has converged at iteration 122. This outcome can be explained by the fact that it takes less time to scan across policy space to find the optimal policy than it takes to compute utility for each state. On the other hand, the time taken for Policy iteration for each iteration on average is more than the time taken by VI algorithm for each iteration. This can be explained by the fact that policy iteration computes the utility of each state at all iteration based on the new policy. This computation of utility can be time consuming. The reward for both VI and PI starts as negative and reached to approximately -50 for both the algorithms at



convergence. Time taken for PI at for 10 iterations is approximately 3 times as that of VI and the gap has reduced as the number of iterations increased. From the plots, it can be observed that maze problem has taken significantly higher number of iterations both for policy iteration and value iteration algorithms to converge. For the value iteration, until 50

iterations the reward remained at -300. At 60 iterations, the reward has started to climb from -300 towards -50. Interestingly the reward at convergence is still negative which is -50 for the maze problem.

4. Solving the Markov Decision Processes (MDPs) using Q-Learning

Value Iteration and policy iteration algorithms do a great job at determining the optimal policy with an assumption that the agent has a great deal of domain knowledge. These algorithms assume that agent has prior knowledge on the transition function and reward for all states in the environment. In some cases, however, the agent might not have prior domain knowledge. Q-Learning algorithm presents a way to learn the domain knowledge i.e., a priori knowledge and learning time can be traded with each other.

Q-Learning is presents model-free learning. In this setup, the agent doesn't need to have any model of the environment. The only knowledge that the agent need to have is number of possible states and the permitted actions that can be taken. In Q-Learning, each state is assigned an initial value called Q-Value. When a state is visited and a reward is received, the reward value is used to update the Q-Value of the state. Note that since the rewards may be stochastic, each state must be visited more than just once. Q-Values are updated using the below formula

$$Q(s,a) = R(s,a) + \gamma\, max_{a'} Q(s',a')$$

This equation indicates that the Q-value of taking an action, a in the state, s is the sum of immediate reward for the State s on action a plus the value of best possible state, action combination for the successor state.

In Q-Learning, since the gent is learning in an online mode, learning is costly. The time spent by the Q-Learner making mistakes early on improves the overall performance. In the initial iterations, when the knowledge of the world in minimal, Q-Learner explores the environment, try unknown actions and learn about the world. In the later iterations, the agent would almost always choose the best action for each state. Q Learning equation is given as below:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t))$$

α is the learning rate that controls the different between the previous and newly computed Q Value. α ensures that the Q-Value doesn't grow exponentially. Q-Learning is performed using probability models. A probabilistic function is modelled that assigns a probability of being chosen to each action given the agent is in a state. This function should tend to choose those actions which result in higher Q Value. Also, the probability of choosing the highest Q-Value should increase over time. To achieve this, Boltzmann distribution is used. The parameter k, called as temperature, controls the probability of selecting actions that are not optimal.

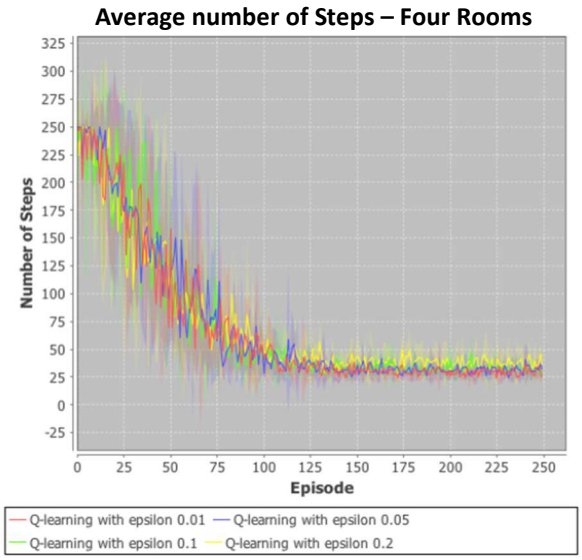
$$P(a|s) = \frac{e^{Q(s,a)/k}}{\sum j^{e^{Q(s,a_j)/k}}}$$

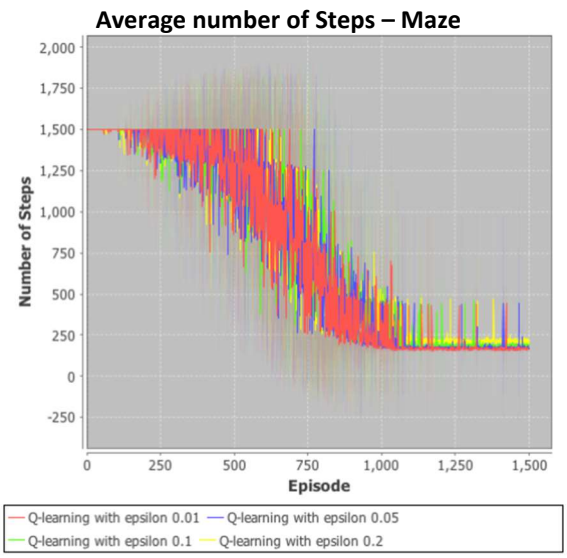
When k value is large, there will be uniform selection between all actions. When k is almost zero, best action is selected. During implementation of Q-Learning, K is initiated to a large value and it decreases over time. This concept is similar to simulated annealing in Randomized optimization. When the temperature, k value is large, large steps are taken in the start. Gradually as the temperature reduces, the algorithm converges to the local maximum.

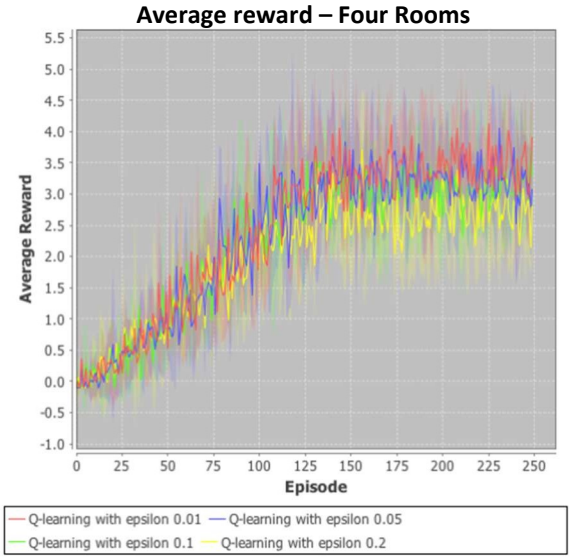
4.1. Implementing Q-Learning on Four Rooms and Maze Problems: Q-Learning has the parameters γ, TR and SR similar to that of VI and PI. It has 3 additional parameters. There are the Learning Rate, α; Initial Q Value Q_{init} and epsilon ε. High values of Q_{init} are advisable as it enables exploration. ε is a measure of randomness. In this implementation ε-greedy policy is used to maintain a balance between exploitation and exploration. parameters used to explore Q-Learning are:

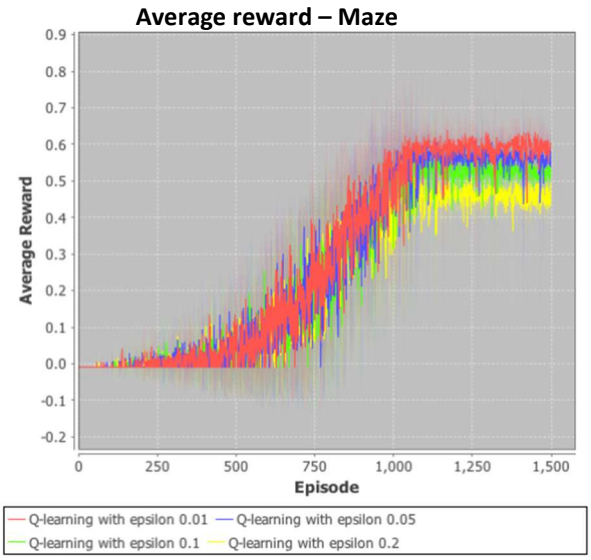
Name of the Parameter	Values on which experiments are run
Discount Factor γ	0.99, 0.8, 0.85, 0.9, 0.95
Terminating Reward (TR)	5, 10, 20, 50, 100
Step Reward (SR)	-0.1, 1
Learning Rate (α)	0.01, 0.1, 02, 0.5
Initial value for Q (Q_{init})	0.1, 1, 5, 10
Epsilon (ε)	0.01, 0.05, 0.1, 0.2

Q-Learning is assesses based on the average number of steps and average rewards per episode. The below graphs show the impact of epsilon (ε) on the 2 grid world problems Four Rooms and Maze. The average number of steps graph plots the episode number to the number of steps taken within that episode. It can be observed that the number of steps decrease gradually per episode. The decrease in number of steps is immediate for the four-world problem. The number of steps started at 250 and reduced gradually to 25 steps from 0 to 100 episodes. From 100 to 150 iterations, the number of steps remained fairly constant at 25. For the maze problem, the number of steps remained fairly constant for the first 500 episodes and then reduced gradually from 1500 to 250 in between 250 and 750 iterations. From 750 60 1500, the number of steps remained constant at 250.





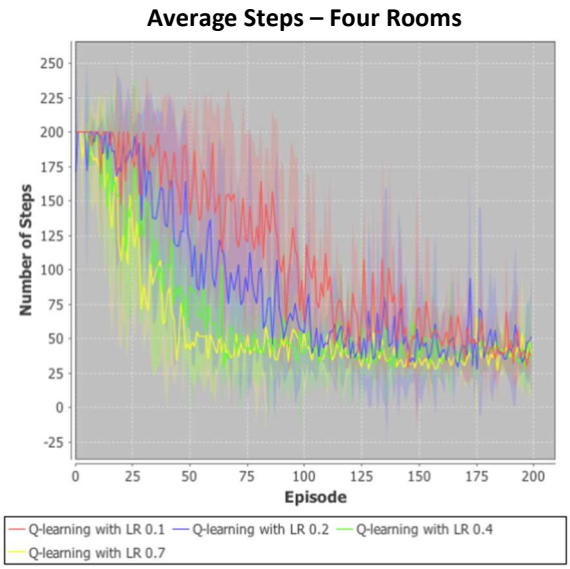


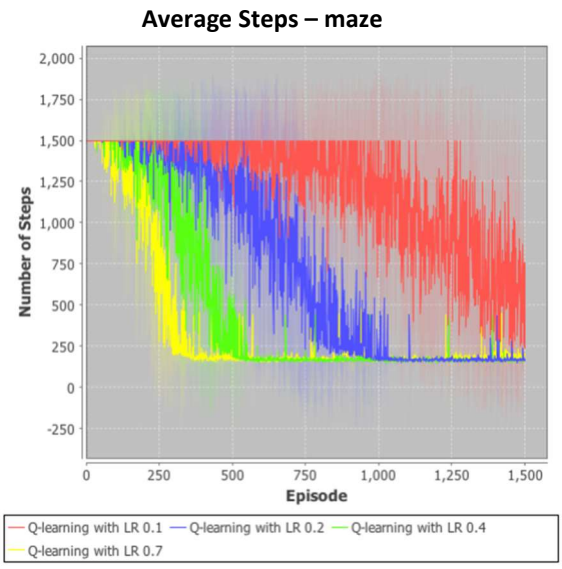


Both four-rooms and maze problems show a consistent behavior wrt ε. A smaller value of ε led to lesser number of steps per episode. Maze problem showcases a larger difference between steps because of the larger number of steps.

The average reward increases per episode for both four rooms and maze problems. Lower value of ε showcases lower average reward for both four world and maze problems. The difference between rewards for epsilon values is significant in maze. Also, the variance in average reward value for epsilon values is higher for four rooms problem compared to that of the maze problem. E value being small implies that agent wouldn't take a random action. This works well for the maze because as the maze is large, taking random steps might result in not reaching the target state at all. However, in the four-world problem as the space is smaller, agent can perform well by exploring the space rather than exploiting. It can also be notices that maze made use of exploitation in order to converge faster.

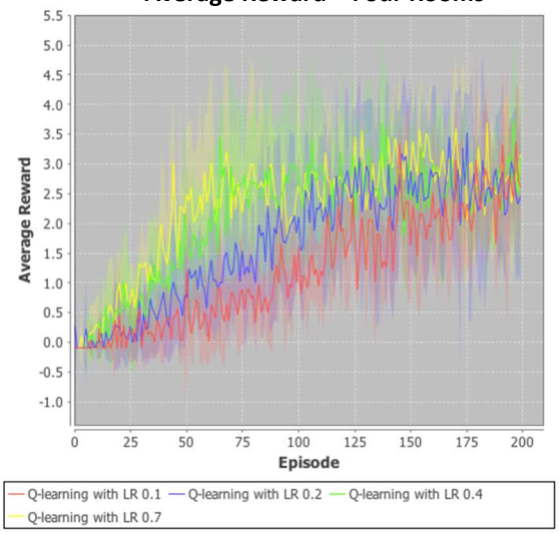
Next the results of varying values of learning rate (α) are analyzed. As shown in the table earlier, Q-Learning is until for varying values of learning rate α: 0.01, 0.1, 02, 0.5. For each of the various α values, Q-learning is run until convergence. For each value α, the number of steps and average reward values are analyzed and the results are plotted in graphs in the next page.

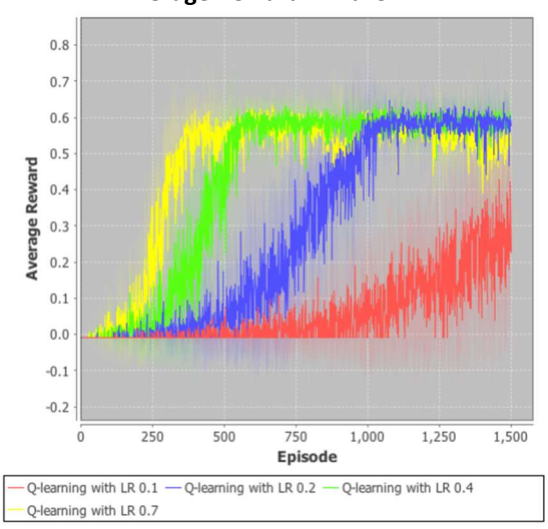




From the adjacent graphs, it can be observed that the average number of steps decreases consistently for all values of α. The decrease is significant for higher values of α than it is for lower values of α. For lower values of α, the number of steps shows a steep decrease from 200 to 40 steps within 25 episodes. However, for higher values of α, it took 125 episodes for the number of steps to be reduced from 200 to 125, The difference in number of steps plots for various values of α is more significant for the maze problem. This can be explained by the fact that maze has more number of episodes. Like for four rooms, the decrease in number of steps is significant for lower values of α than it is for higher values of α. There is a steep decrease in number of steps from 1500 to 175 from episodes 0 to 25 and after that the number of steps remained constant at 175 for further episodes. The decrease in number of steps is not as significant for higher α.

The variation in average reward for different values of α is shown in the graphs below. It can be observed that the average reward is higher for lower values of α and lower for higher values of α for both Four rooms and maze problems. The difference in average reward for different values of α is not very prominent in four rooms problem as it is in maze problem.

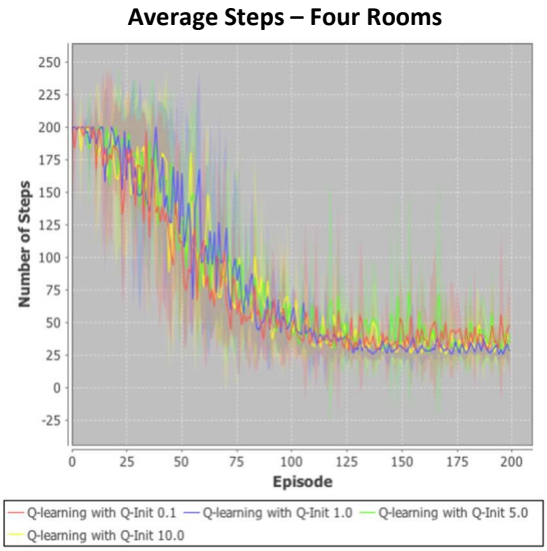


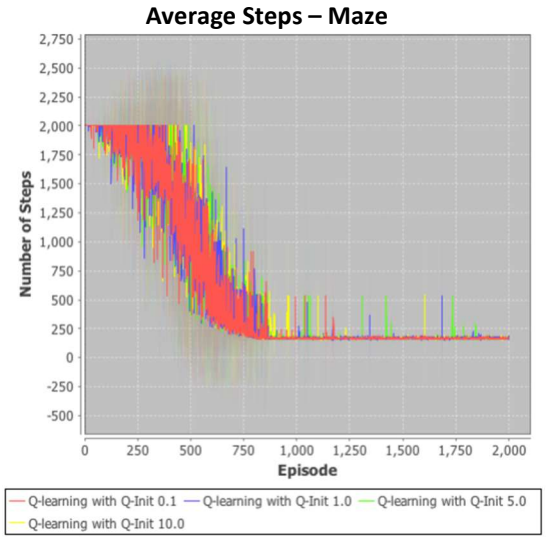


In the maze graph, a clear demarcation in the graphs for varying alphas can be noticed. Maze converged within 300 episodes with α = 0.7. It converged in 450 episodes when α = 0.4 and in 1000 episodes when α is 0.1 and 1500 for 0.2. In maze problem a clear preference for higher α values can be noticed. This implies that the agent constantly relies on new information for larger grid world problems. For smaller problems like four rooms, the difference in α is less important and has less influence over the learning outcome.

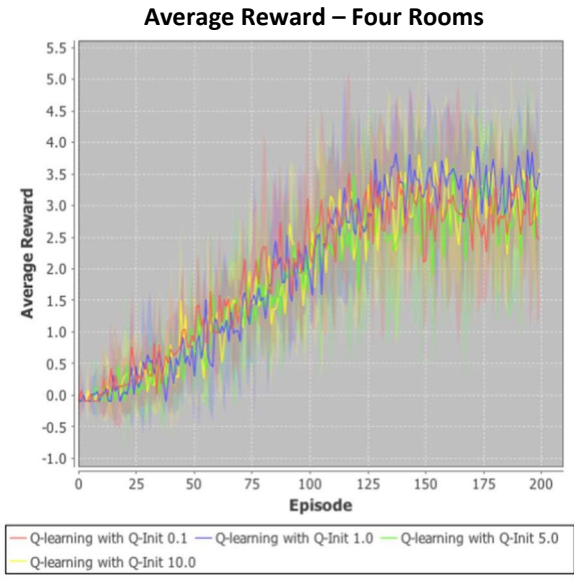
Next, the impact of various values of Q_{init} on the Q-Learning algorithm outcome is analyzed. Similar to that of α and ε analysis, the number of steps and average reward are plotted for varying values of Q_{init} for various iterations until convergence. It can be notices that lower values of Q_{init} converge faster than the higher values of Q_{init}. The average rewards per episode show similar behavior. For higher Q_{init} the convergence takes longer time. The difference between different values of Q_{init} is significant for maze problem than it is for simpler four room problem. It can be observed from the graphs that the number of steps decreased gradually from 200 to 50 from 0-75 episodes and remained constant at 50 steps for four rooms problem. Similar to four rooms problem, there is a decrease in number of steps as the number of

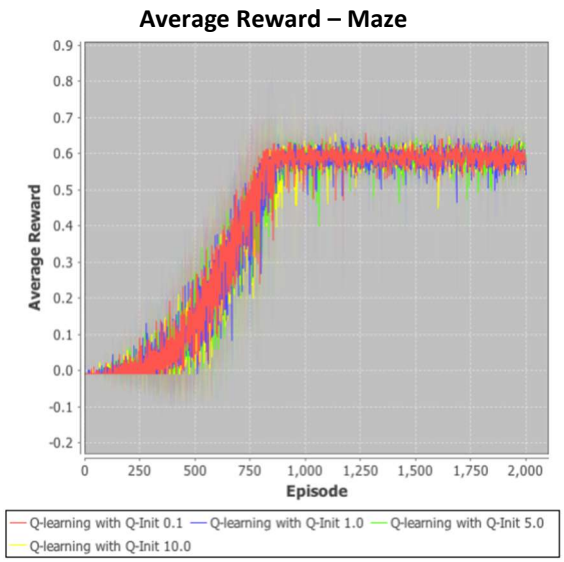
episodes increased for maze problem. The number of steps decreased gradually from 2000 to 250 between episodes 0 – 750 and remained constant at 250 from episodes 750 to 2000.





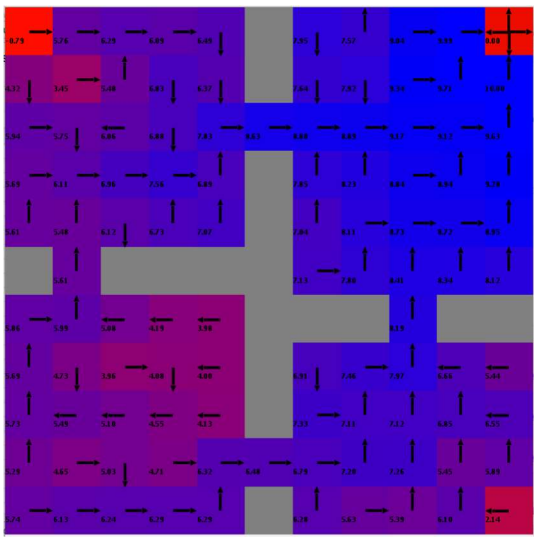
The average reward per episode increases gradually as the number of episodes increases. The difference in increase for various values per episode is significant got maze problem than it is for the four rooms problem. Four rooms problem converges faster for lower values of Q_{init} at 125 iterations. Same behavior is observed in the maze problem as well. For lower values of Q_{init}, the algorithm converged faster at 750 iterations.

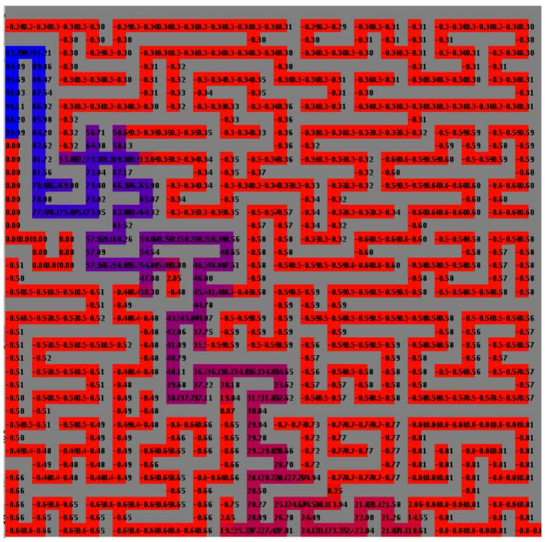




From these observations of various values of learning rates α, epsilon ε and Q_{init} values, it can be concluded that complex grid world problems like maze that have high number of obstacles perform well for low values of ε, low values of Q_{init} and high values of α. This is also true for small smaller problems like four rooms. However, for smaller grid world problems, the different in varying the values of the parameters is not as significant as it is in complex grid world problem like maze.

Policy maps obtained from Q-Learning for both four rooms and Maze problems are given using a ColorBased Interface in the adjacent graphs. The maze problem shows a clear distinction between blue and red. There is a change in color gradient from red to blue from starting point to the goal state. QL has performed significantly well over Value Iteration and Policy iteration in the maze problem. Policy map obtained for four rooms problem also shows a well-defined path from the starting point to the finish point. The strength of Q-

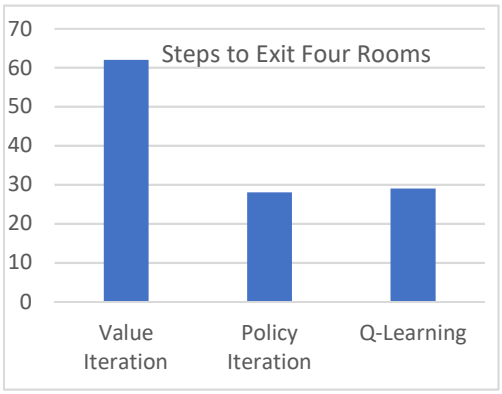


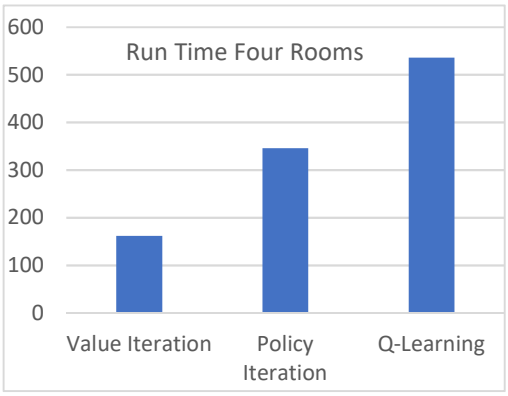


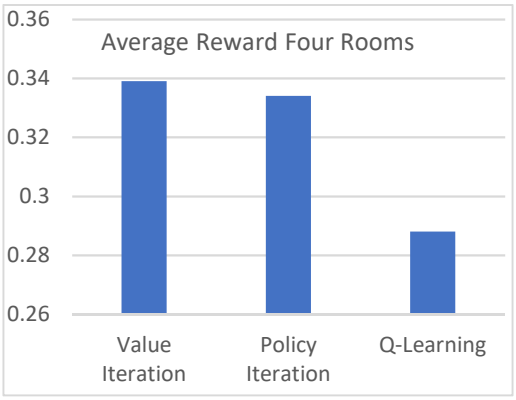
Learning is evident in the maze problem. For maze, Q-Learning is able to converge in less than 1000 iterations within 250 steps.

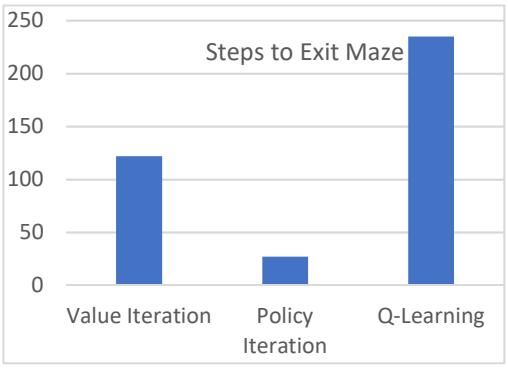
5. Conclusion

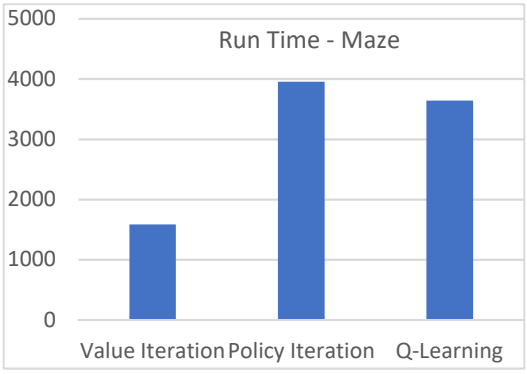
The below plots show the comparison of results of three algorithms implemented for this assignment. The comparison is performed for both four rooms and maze problems. Comparison is performed on three key metrics that are output of implementation of the three algorithms. They are number of steps to converge, Run time and Average reward

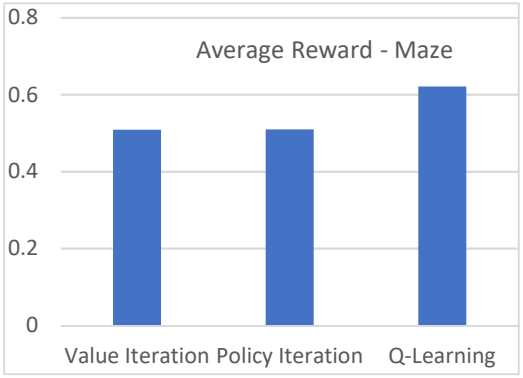












These plots are plotted for the best parameters obtained for all three algorithms for both the problems. It can be concluded that Q-Learning has performed the best for maze with highest average reward. Even through the number of steps taken and run time are higher. Value iteration has performed the best for four rooms problems. It has the least run time, even though the number of steps to convergence is high. Policy Iteration is very close to Value iteration for four rooms problem. There is a very little difference is the average reward obtained compared to Value iteration. Q-Learning has performed very poorly in four rooms problem.

It can also be observed that the average run time for Q-Learning is high for both the problems. Policy Iteration takes significantly less number of steps to converge than the other two algorithms for both the problems. Value iteration runs in lesser time compared to the other two algorithms for both the problems. In conclusion Q-Learning performs the best for complex grids and Value Iteration/Policy iteration can be used for simpler grid world problems.