Markov Decision Processes

Surabhi Amit Chembra
surabhichembra@gatech.edu

Abstract
This paper applies planning algorithms of Value Iteration, Policy Iteration and Reinforcement Learning algorithm of Q-Learning on Markov Decision Processes and compares the performances in Part-1. The variation in VI and PI performances with the size of the MDP, is analyzed in Part-2. Various exploration strategies are tried on Q-learning and the results are analyzed in Part-3.

Introduction:
The assignment is implemented using BURLAP in a Windows 10 machine. Two MDPs are selected for this assignment- MDP1 with a smaller number of states and MDP2 with a larger number of states. In Fig.1, MDP-1 is a 4*4 grid-world with 16 states, modeling a street and a robot starting from the downmost left corner (blue block) trying to reach the shop in the rightmost top corner (green block). Each time hitting the building wall (black block) costs 1 point and reaching the terminal state (green block) gives a reward of +100 points. The agent stays put if it hits the wall. Each step costs 1 point and this will urge the robot to reach the goal quickly. And there are a few pits (red) which penalizes the agent by 50 points. The second MDP in Fig.1 is 20*20 and hence has 400 states. It models a bigger street where a robot starts from bottom-most left corner(blue) and needs to reach a shop in topmost right corner(green). For both MDPs, the agent can move north, south, east or west. Also, the actions are stochastic, so 80% of the time, it moves in the intended direction and the rest of the time, it can move in any of the other three directions with equal probability. Stepping on an orange block costs 30 points, yellow block costs 10 points and red block 50 points. The agent's goal is to reach green block from blue block, in both MDPs. To account for time running out and to quicken the process of reaching the goal, the future rewards are discounted by a factor of gamma (discount factor). It is set as 0.99 because gamma values higher than that might not urge the agent to achieve the goal quickly as the terminal-state reward is not reduced much in value with time, and gamma equals to 1 gives an infinite horizon MDP. Smaller gamma values will highly reduce the effective terminal-state reward value within a few steps, thus making it less attractive by the time the agent is midway, and this can cause the agent to lose interest in pursuing the final reward. For each MDP, the results with a smaller gamma of 0.6 are also analyzed in part-1, in comparison with gamma equal to 0.99, and the effect of gamma on convergence rate of algorithms is also explored.

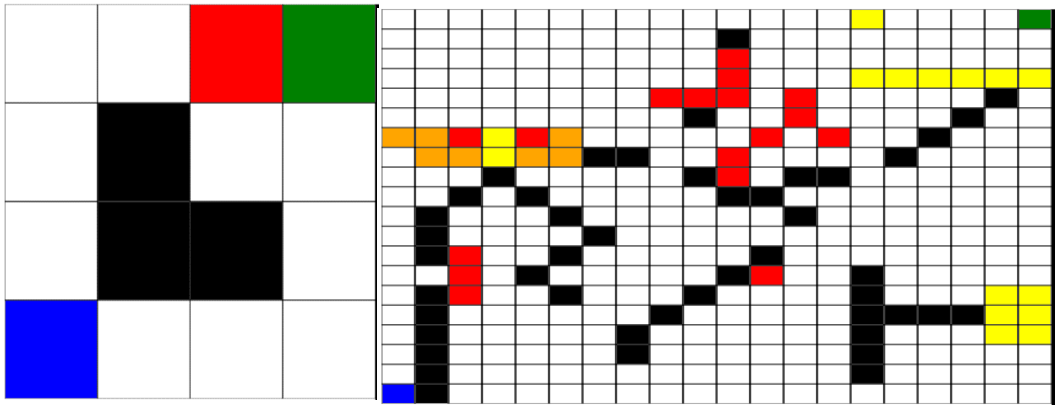

Fig.1.MDP-1 [LEFT] and MDP-2 [RIGHT]

Why are the MDPs interesting?
It is important in robotics that the humanoid robots be able to walk and cross streets like humans do. Also, humanoid robots should be able to traverse the shopping complex floors without hitting on shelves, and reach the products of interest. For house-helping robots, being able to reach the goal position quickly, say, a room in the house, is a prominent functionality. So, solving these MDPs where robot reaches the goal without hitting obstacles or falling in pits, helps in improving efficiency of these robots, thus making these grid-world MDPs interesting **from application point of view**. The difference in states between MDP1 and MDP2 can help understand how the number of states affects the algorithmic performance. MDP2 clearly shows performance variations between policy iteration, value iteration and Q-learning. For Q-learning, varying hyperparameters like learning rate, epsilon and initial Q-values, gave different results. For MDP2, while planning algorithms converged to maximum rewards within a few iterations, Q-learning took around 20K iterations with tuned hyperparameters and did not even converge in 20K iterations when higher learning rates were used. Also, high constant learning rate with greedy strategy made Q-learning(QL) stuck in local optima in MDP-2. On the other hand, for MDP1, all three algorithms converged and gave maximum reward in reasonable number of iterations. These **interesting differences and observations** led to choosing these MDPs for this assignment. Also, for both MDPs, several grid-worlds with different reward structures and different number of states were tried and the ones whose performances emphasized the differences between algorithms, were chosen. It was noticed that changing the reward structure can change the optimal policy, even when the MDPs have same number of states. This is because the agent is actually optimizing the path to terminal reward, so changing the rewards or changing its location can change the optimization problem the agent is trying to solve. However, if there is no relative change in rewards, say when all rewards are added or multiplied by a constant number, there is no change in optimal path. Some of the experimented grid worlds were too simple thus giving good results often, or too complex to give good results at all. For example, out of the many 20*20 grid worlds tried with different reward structures, some always had QL getting stuck with a sub-optimal policy and taking hours to converge when there were a lot of obstacles in the path to reward, and some always found the optimal path in a few iterations when

the grid-world had less obstacles and pits. Small MDP is chosen as 4*4 because any lesser size will not be complex enough to showcase the differences between performances of different algorithms and any bigger size would not help differentiate the algorithmic performance differences between small and large MDPs. Large MDP is chosen as 20*20 so that it is large enough to demonstrate how the size of gridworld affects algorithmic performance and not too large that the policy development figures are too big for this assignment.

Algorithmic Methodology: Three algorithms of Value Iteration (VI), Policy iteration (PI) and Q-learning (QL) are used. For value iteration, initially, only the value of the terminal state is known. Using Bellman's equations, all other state values are iteratively calculated until the values from consecutive iterations don't change beyond a specified delta value (error threshold). The policy is extracted after finding the optimal converged value function. Policy iteration starts with a random policy and each iteration has a policy evaluation step that computes the value function of that policy using Bellman operator and a policy improvement step that finds a new improved policy based on the previous value function. Until convergence, the policy strictly improves in each step and since there are only fixed number of policies (Number of states $^{\text{Number of actions}}$), PI converges. Though both VI and PI are guaranteed to converge, since VI convergence looks for the exact utility values to stabilize while PI only cares about the relative order of values in states (action with maximum value selected in each state), PI converges in fewer iterations. The third algorithm, Q-Learning, is a model free reinforcement learning algorithm and thus does not need the domain knowledge required to create models (Transitions and Rewards). It is an online action-value function learning algorithm with an off-policy exploration. QL uses an exploration-exploitation approach to learn state values and exploit more as it progresses. In Q-learning, learning rate decides the relative importance of older and recent information in determining Q-values, which in turn decides action-selection. Q-learning is guaranteed to converge to optimal values for a finite MDP, due to contraction property. "Q-Init" values are the initialization values for all the states in Q-table, which later gets updated with the "true" values. Since the terminal reward is 100, the maximum Q-value for any state is 100. So, all Q-values can be optimistically initialized as 100, or neutrally as 0 or pessimistically as -100. All these three values are tried for each MDP and results analyzed in Part-1. The exploration strategy decides when to explore or exploit. The epsilon-greedy approach is used, where when a randomly chosen value is less than epsilon, actions are selected randomly and otherwise, the action with maximum Q value is chosen. The parameter for exploration, epsilon, is between 0 and 1, and a larger value means more exploration and a smaller value indicates more exploitation. Higher epsilon values can make the agent more exploratory but reduces the rewards obtained. Lower epsilon values can make the agent stuck in local optima as it was not given enough chances (randomness) to explore the path to global optimum. Several epsilon values were tested for each MDP in Part-1 to determine which is best in terms of rewards, step counts, convergence, runtime etc. Other exploration strategies like greedy policy, random policy and Boltzmann policy (a type of simulated annealing) are also tried and compared with epsilon greedy approach in Part-3.

Part 1: MDP analysis

The convergence for Value Iteration is defined in such a way that when the maximum change in the value function is smaller than a threshold value, VI will terminate. PI convergence is when the policy does not change in consecutive iterations. According to PolicyIteration.java in BURLAP which uses "MaxPIdelta" as convergence threshold, when the maximum change between policy evaluations is smaller than a threshold, the planning terminates. The threshold(delta) for both VI and PI were set as 1e-6 because smaller threshold values give higher precision. Also, since we are concerned only about action-selection (policy) based on values, and not about convergence to the exactly true utility values, only the relative order of the values matter. So, making delta less than 1e-6 unnecessarily increases the required iterations to converge without changing the relative order of values in later iterations and thus not affecting the policy. For Q-learning, however, convergence was tougher to obtain when the threshold was set as 1e-6 and was run for 25K iterations to see the best possible convergence achievable.

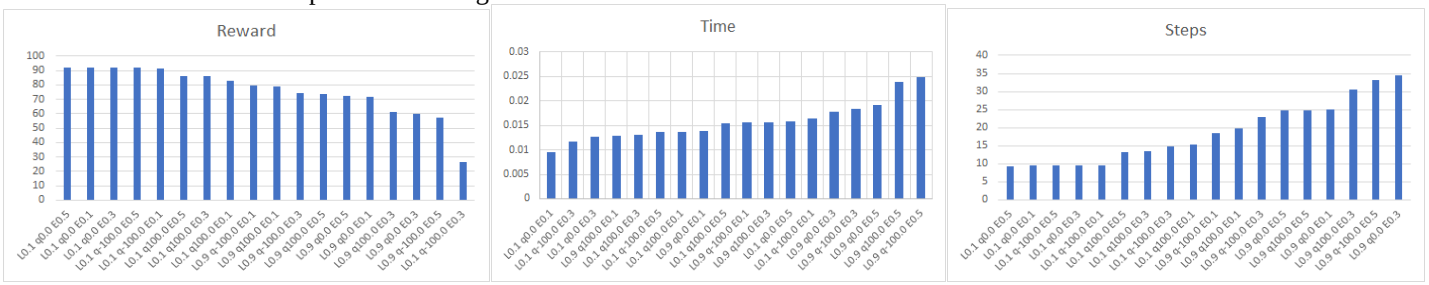

Fig.2. Hyperparameter tuning for Q-learning

When PI convergence is defined such that the policy remains the same between two consecutive iterations, the convergence is obtained by 4 iterations in 20 microseconds and the algorithm converged to a reward value of 93 in 9 steps. However, in the subsequent iterations, though the policy did not alter, the maximum value value-function did alter by good amount. So, in terms of convergence threshold mentioned earlier for value-function (policy evaluation) differences in consecutive iterations, PI converged at around 16 iterations in 49 microseconds to a Reward value of 93 while VI converged at 36 iterations in 47 microseconds to 93. The best hyperparameters for Q-learning in terms of rewards and time cost is when learning rate(L)=0.1, Initial Q values(q)=0 and epsilon for epsilon-greedy approach (E)=0.1 in Fig.2. This QL setting converged with a delta of 0.5, to 93 in 240 iterations but within 21 microseconds. 0.5 was chosen as convergence delta (error threshold) for QL because lower delta values will require lot more iterations and once this threshold was reached, the policy in all states had converged to optimality. All algorithms took an average of 9 steps from start to goal. Though 6 is optimal number of steps, the stochasticity of actions causes the agent to take more steps on average. In Fig.3, PI and VI converges to optimal reward value within the first few iterations. Though convergence of value function in VI takes more iterations (36), the optimal policy (relative order of utility values of states) in each state is discovered early itself. QL shows

fluctuations in the initial phase but eventually converges. PI takes most runtime due to the policy evaluation and policy improvement steps. Policy evaluation is effectively calculating value functions for all states like in VI but is faster than VI because the action for each state is already specified by the policy, unlike in VI, where the best actions must be discovered using max operator. However, policy improvement is slow because based on the converged values in policy evaluation step, the policy, if not optimal, must be updated by choosing the best action using one-step lookahead (so need to use argmax). PI is followed by VI in runtime, which needs time for calculating the updated utility values for all states after finding the best action for each state using time-consuming max operator. QL takes least time as it updates the Q-values of only those states traversed in its path, in an iteration.

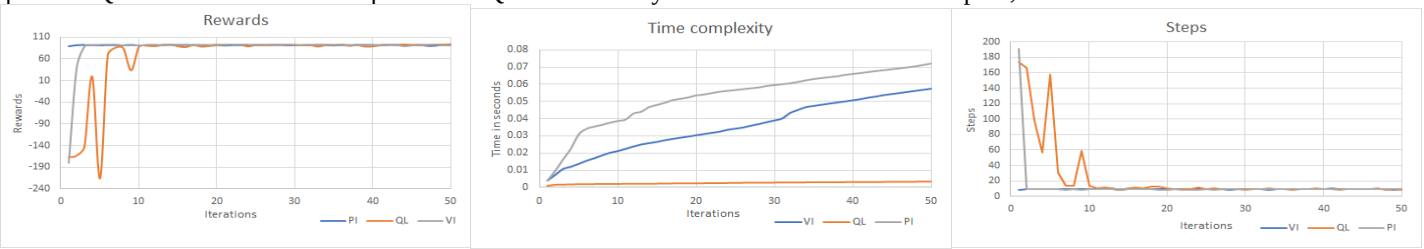

Fig.3. Comparison of VI, PI and QL for MDP-1

Since Q-learning does not know the effect of a state-action pair unless it is executed at least once, and VI only needs to enter a state at least once to discover all its successor states, VI is more powerful than Q-learning and reach convergence earlier than Q-learning. The step count is high for Q-learning in the initial phase because being a model-free learning, QL takes non-optimal actions initially as there is no model to guide it. Though model-free methods are easy-to-understand, they waste information about state connections available from transitions which can otherwise be used to build model. However, they learn from experience and lessen the step count in later iterations. Also, they learn each state separately thus taking many iterations to learn, exemplified by the fact that QL took lot more iterations to converge in both MDPs of this assignment.

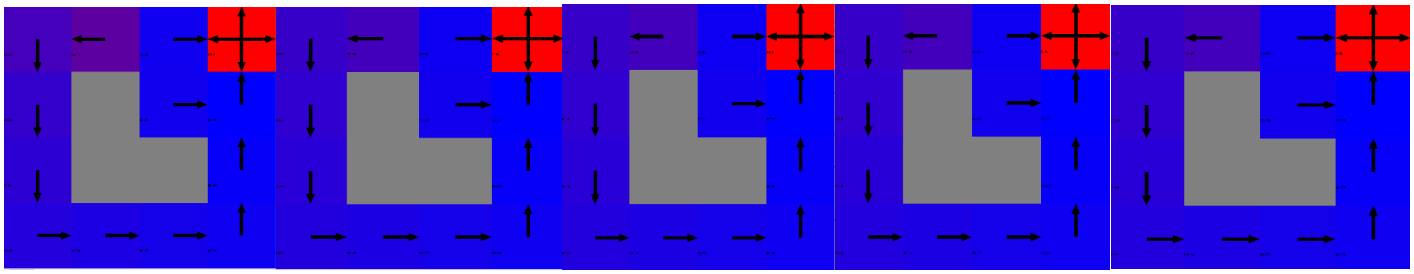

Fig.4.Value Iteration visualization for iteration =1,2,5,15,50 respectively

For Value Iteration, in Fig.4, it is noticeable that the policy remained the same from first iteration itself, however, the value function converged only at 36 iterations. From iteration 1 to 2, the color of second cell of top row has changed indicating the large change in value function of that state (62 to 73).

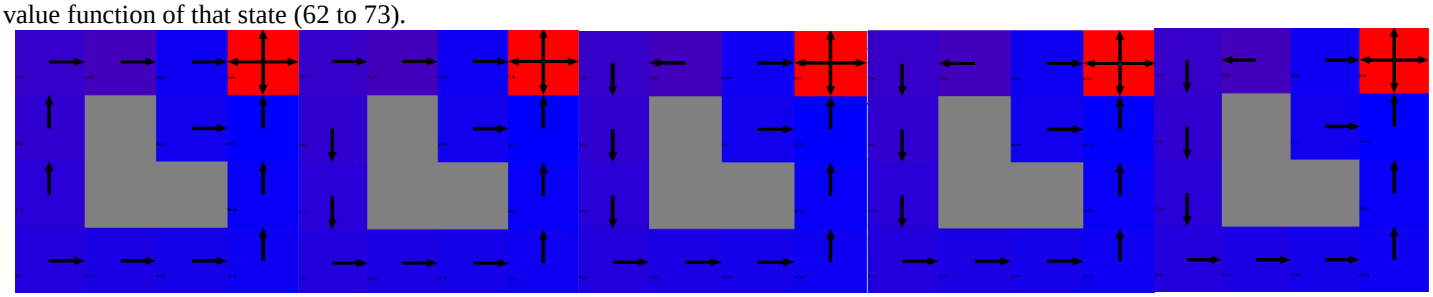

Fig.5.Policy iteration visualization for iteration =1,2,5,15,50 respectively

For PI, in Fig.5, for first and second iterations, the policy of second cell in top row is to go east while the optimal policy is to go west due to the penalization of 50 points in third cell of top row. In second iteration, the agent learned the optimal policy for 2nd and 3rd cells in the first column and in third iteration in Fig.6.a, the agent started choosing south in first cell of first column, thus distancing itself away from the (-50) cell. By the fourth iteration, agent learned the optimal policy. Thus, in the sense of policy convergence, the agent took only four iterations to converge. However, when the convergence is defined such that the maximum value value-function (corresponding to chosen action) should not differ by more than 1e-6 in policy evaluations of consecutive iterations, PI converges only in 16th iteration.

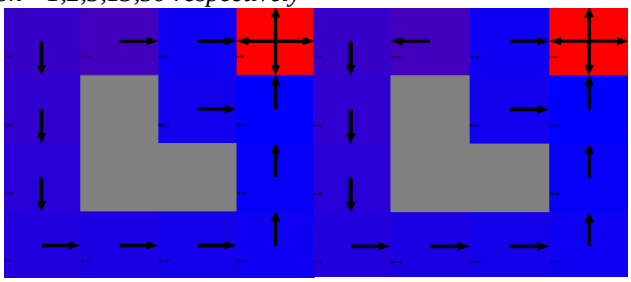

Fig.6.PI visualization for iteration =3 and 4 respectively

Fig.7 shows Q-learning policy development. In the first iteration itself, the policies are optimal in all states. However, the value-function kept improving through iterations. The color change in 1st and 2nd cells of the first row from first to second iteration indicates the commendable improvement of values (26 to 66 in first cell and -88 to 36 in second cell). The Q-table was initialized with 0, and some values went down like in second cell and then rose to the true utility values while others directly got increased to true values like in first cell. Though the policy became optimal very early, the value function took 240 iterations to converge. This is because the truth

of terminal state reward needs many transitions and subsequent Q-value updates to propagate to all the states, thus making the ultimate Q-values reach the true utility values.

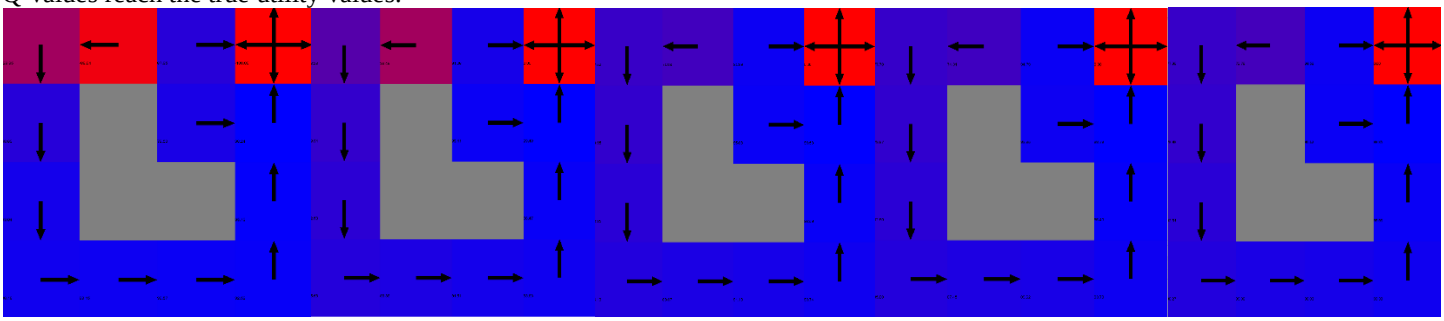

Fig.7.Q-learning visualizations for iteration =1,2,5,15,50 respectively

Another interesting observation was made when the discount factor(gamma) was reduced to 0.6 from 0.99. The iterations needed to converge got reduced from 16 to 6 for PI and 36 to 16 for VI when the delta for convergence is 1e-6. The runtime till convergence, with 1e-6 as delta, got reduced from 49 microseconds to 29 microseconds for PI and 47 to 28 microseconds for VI. Though convergence rate improved for both PI and VI, the rate of improvement is more for VI. This is because, in the worst-case scenario, the number of iterations required to reach the optimal value function for VI grows polynomially in 1/(1-gamma) and so when gamma is reduced, 1/(1-gamma) is reduced, thus reducing the number of iterations needed to converge. Also, when gamma is increased, since the iteration count needed to converge to optimal value function might increase heavily, iteration count increases a lot for VI but not so badly for PI. This is because PI only cares about the relative order of values and not the exact utility values, and it need not wait till the optimal value function is reached. Regarding QL with gamma 0.6, the number of iterations got reduced from 240 to 13 and the runtime till convergence, from 21 microseconds to 3 microseconds. The QL agent got stuck in a local optimum, with step count increasing to 200 from 9, and the reward dropping to -182 from 93. More exploration opportunities are needed for the agent to come out of this local optimum. Also, for lesser gamma values like 0.6, the reward is not attractive enough because by the time the agent takes several steps and reach it, it's effective value would be highly discounted. This leads to the agent hovering over state space because it cannot find the reward attractive, as indicated by the large increase in number of steps (200) and the drop in reward value caused by the -1 accumulated by taking several steps and falling into pits (-50, -30 and -10) during its quest for an attractive reward.

MDP2 analysis: In terms of BURLAP definition of maximum change between consecutive policy evaluations not differing by more than a threshold value, PI converged at around 15 iterations in 2.41 seconds while VI converges at around 48 iterations in 1.06 seconds, both to a reward of 17 using 64 steps, when the convergence delta is 1e-6. So, the policy stopped changing between consecutive iterations in PI after 15 iterations. Since the actions are stochastic, even after convergence of policy, the rewards and step counts need not exactly be the same in every iteration thereafter. This is because even though the agent chose correct action, the stochastic nature of the problem can cause incorrect action to be executed. This also explains why the rewards and step count slightly vary in converged results of different runs of different algorithms.

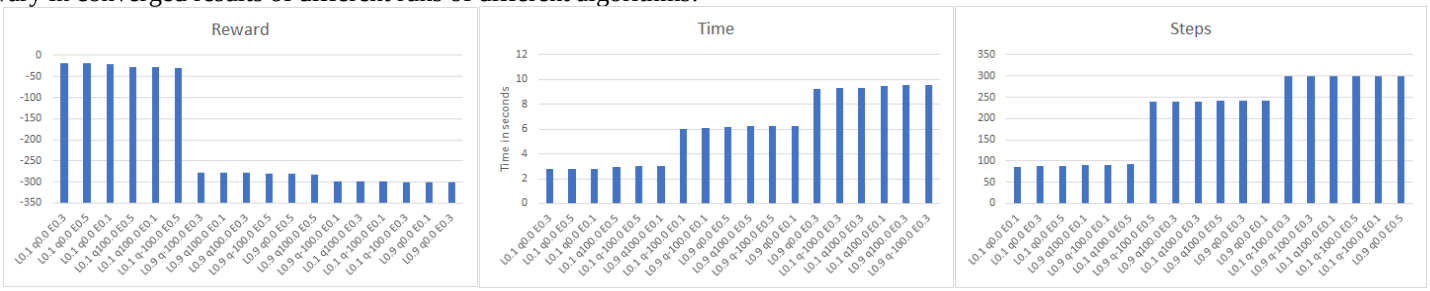

Fig.8.Hyperparameter tuning for Q-learning

Q-learning did not converge to an error threshold of 0.5 for any of the hyperparameters tested but showed signs of convergence by 20K iterations. The averages of reward, time, steps for 20K iterations for different parameter settings are plotted in Fig.8. L=0.1, q=0.0, E=0.3 gave best performance as it's average values for reward is -19 (highest), runtime is 2.76 seconds (least) and steps is 87 (less). When learning rate was low (0.1) and initial Q-values set to 0 or high (100) and with epsilon value set as 0.3 in epsilon-greedy approach used, after 10K iterations, the results from consecutive iterations did not diverge and only differed by a delta value (error threshold) around 2 and started giving 15 as reward value in 2.99 seconds and 65 steps from 14193th iteration onwards. QL was run for 501K iterations and it could only converge to a delta of 1.4 at the end of 501000 iterations. Moreover, when the initial Q values were set low (-100), Q-learning was found to get stuck in a local optimum of reward value -299 using 300 steps, within 15 iterations and 0.12 seconds. Very low Q-initial values make all actions less tempting and hence the agent might settle at a suboptimal place in fewer iterations well before the true value of terminal reward is propagated to all states. This is because when Q-values are initialized pessimistically and sub-optimal actions happen to be chosen in earlier exploratory iterations, the Q-values of these sub-optimal actions will be improved and if the optimal actions are not executed much, their Q-values might just remain to be the low initial Q-values and if the exploration phase ends prematurely before their values are well-updated, the agent might assume that the already visited sub-optimal actions are the best, thus leading to local optimum. However, if infinite tine and infinite exploration is provided, Q-learning will always eventually learn the optimal path. So, to let the agent come out of local optima, it should be given more iterations and

allowed to take random actions and explore the optimal path. Also, when the learning rate was high (0.9), it was noticed that the rewards were mostly very low (around -100 to -500) and used the maximum 300 steps in each episode and the consecutive iterations had value-function differences of more than 60. The intuitive explanation behind it can be that, large learning rates take bigger steps and thus overshoot the optima and bring in instability. Decaying the learning rate or using a small learning rate will help in convergence as they take smaller steps towards global optima with lesser learning rates, and this avoids overshooting and instability.

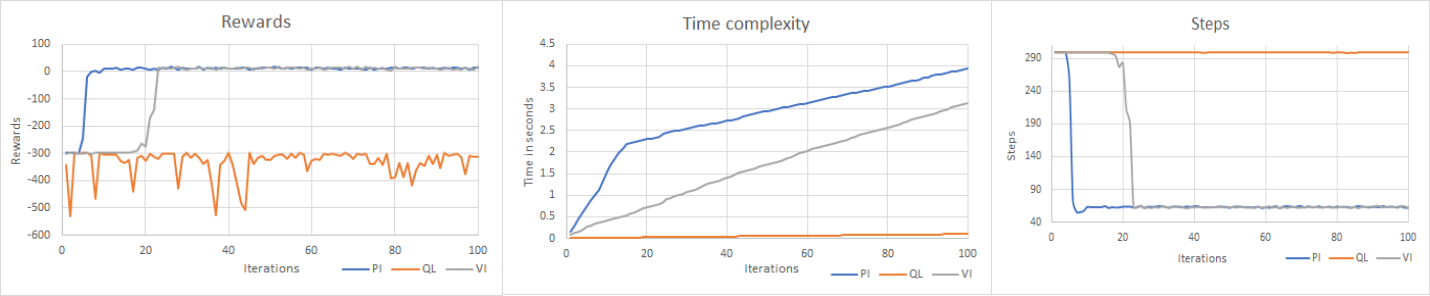

Fig.9. Comparison of VI, PI and QL for MDP-2

In Fig.9, PI learns optimal policy quickly and attains maximum reward within first 10 iterations, as indicated by the high rewards and low step counts from early iterations itself. VI takes more iterations but converge to the same optimal path as PI. The fluctuations in reward values for QL is due to random action selection (exploration phase). QL being a model-free algorithm, needs a lot more iterations to converge and by 20K iterations, it was found to converge to a band with width 2 (error threshold =2). QL was run for 501K iterations and it only converged to a delta of 1.4 by 501K iterations. This explains why reward is low and step count is high for QL till 100 iterations. However, representing 20K iterations in this plot can compress the specific characteristics for VI and PI in early iterations. Regarding time complexity plot, runtime per iteration follows same trends in MDP-1 where PI takes most time, followed by VI and then QL, for reasons explained earlier for MDP-1. However, unlike in MDP-1 where PI took less time to converge than VI, here PI took more time to converge even though count of iterations is more for VI. This is because for this complex MDP, each PI iteration takes lot more time than for MDP-1.

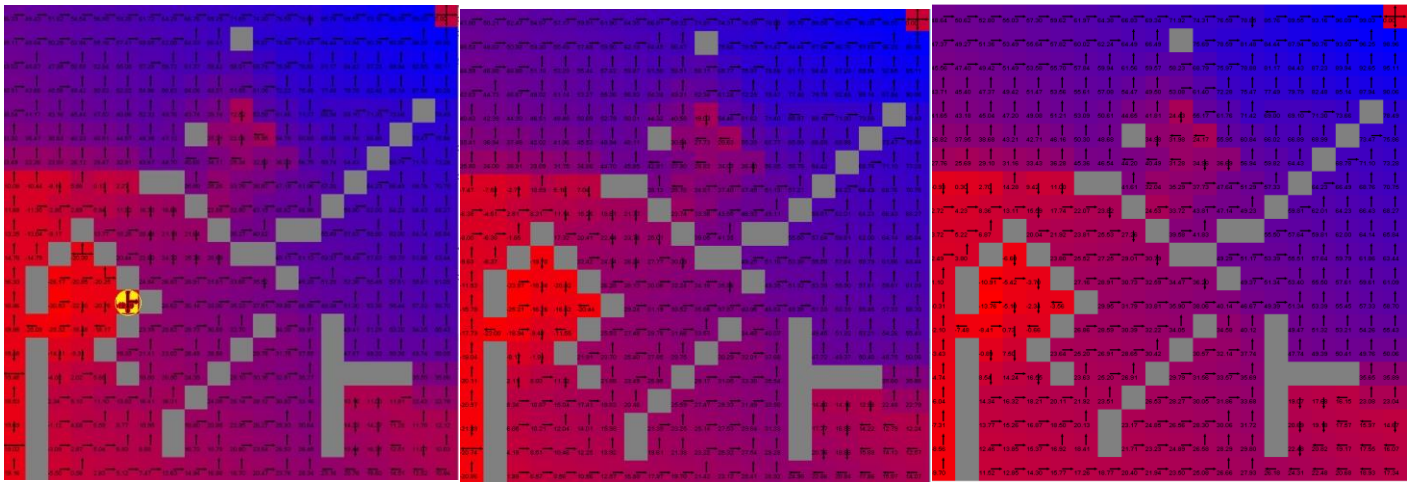

Fig.10.Value Iteration visualizations for iterations = 1,5 and 50 respectively

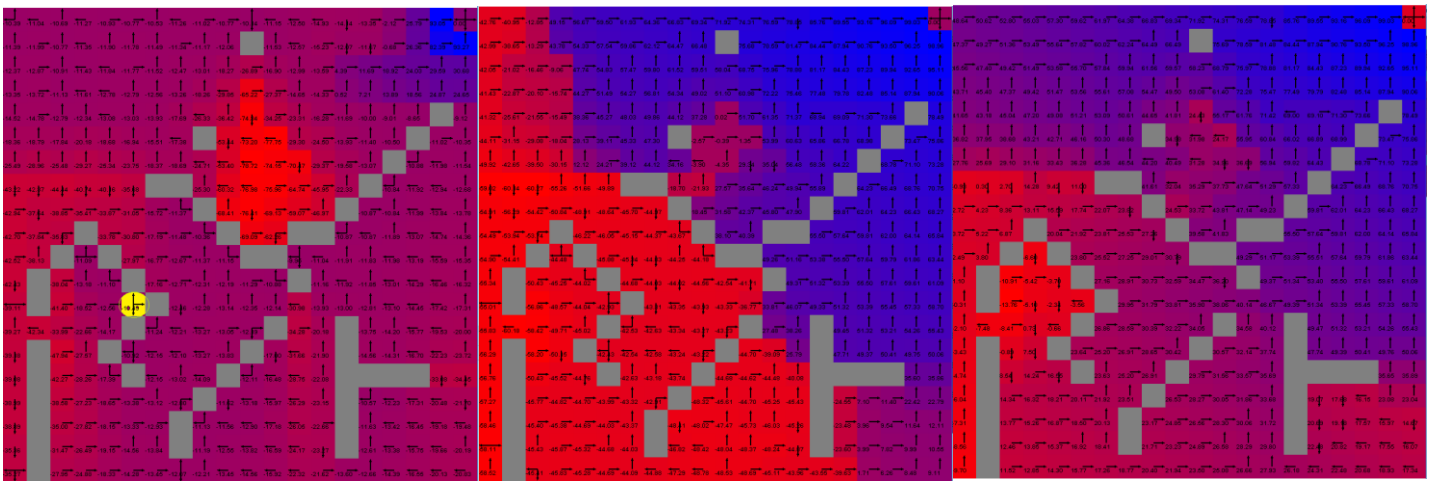

Fig.11.Policy Iteration visualizations for iterations =1,5,25 respectively

In Fig.10 showing the VI policy development for MDP-2, from iteration 1 to 5, though the policies of most of the states remained intact, the value functions of all states varied. The policies also improved in certain states. The utility values and policies of all states

evolved to optimality through iterations and by 48 iterations, they converged. For example, in the first iteration, the utility value of the state highlighted by yellow circle in Fig.10.a is (-19.09) and choosing north, south or east were equally attractive. However, in iteration 5, the value changed to -20.04 and the policy became the optimal action of choosing west. In 50th iteration, the value of the highlighted state became -3.56 and that is the converged true utility value of that state with optimal action being west. Another example of policy development is the second-last state in the first column. Initially, the best action was either west or east, however by 50 iterations, the policy evolved to the optimal action of choosing north. Fig.12 shows the policy development in PI. The evolution to optimal policy in most states is similar to Fig.11, but the values are different until convergence. To illustrate, the yellow-highlighted state in Fig.12.a chooses north, south or east and has value -10.49 in iteration 1 and the value gets updated to -42.29 by 5th iteration with policy becoming the optimal one of choosing west, and then value converged to -3.56 with optimal policy like in Fig.11.c.

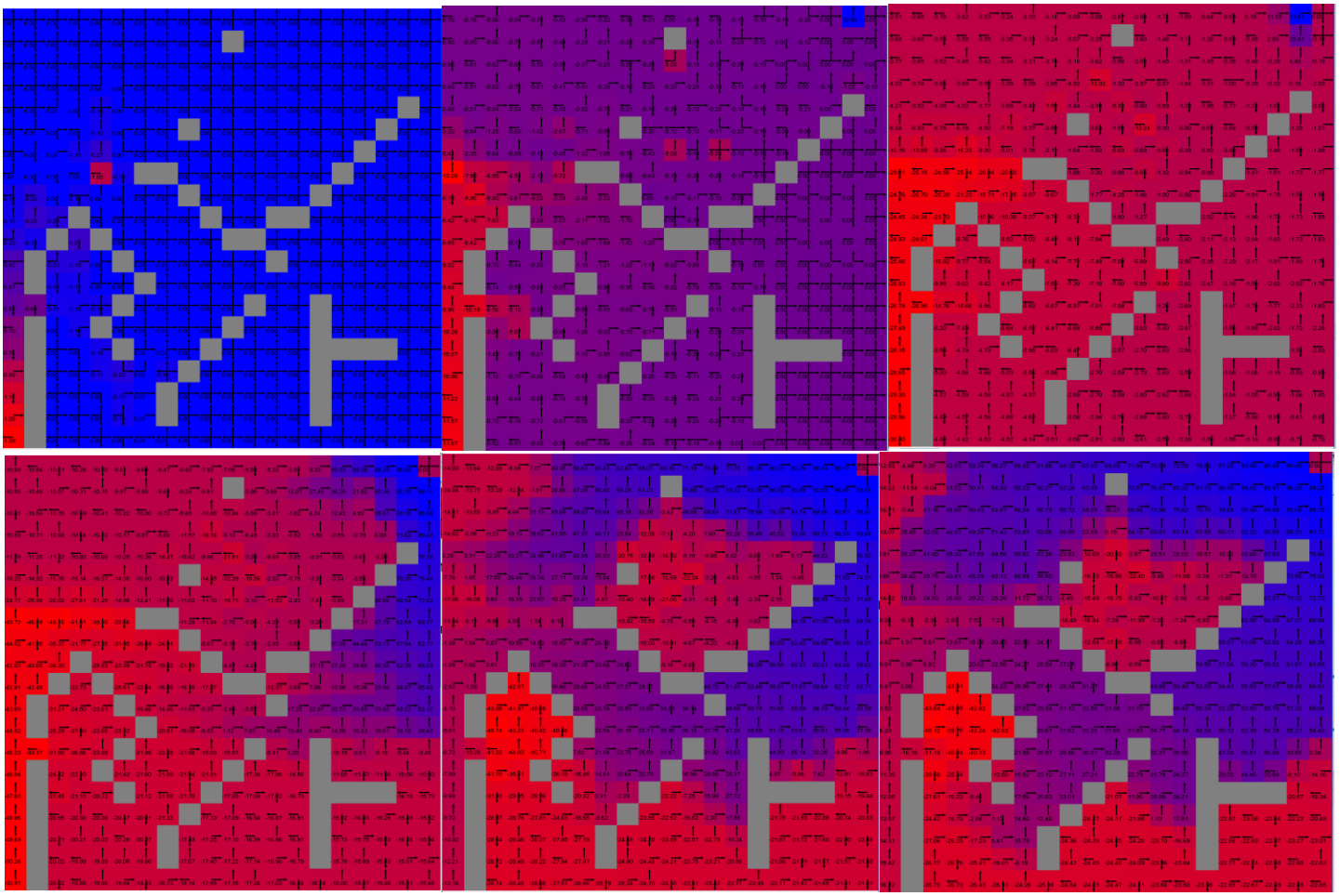

Fig.12. *Q-learning policy development for iterations = 1,20,100,500,5000,15000 respectively*

Since Q-learning takes 14193 iterations to converge to a delta of 2 and 501 K iterations to converge to a delta of 1.4, the policy development is relatively slow between iterations in Fig.12. The hyperparameters used are learning rate of 0.1, epsilon of 0.3 and initial Q values of 0, from the hyperparameter tuning performed earlier. In the first iteration, since most of the states still have 0 (initial value) as current-value, all actions are equally probable in most states. In 20th iteration, around half of the total states have their values updated from the initial value of 0 and the policies of states in left side of Fig.12.b are updated, and some to optimal actions. In 100th iteration, the Q values of almost all states have changed from the initial value and the policy is evolving to the optimal one. Later iterations show constant policy improvement but slow paced. An example of policy improvement can be the fourth-last state from bottom in the first column. At iteration 1, it had value of -0.99 and choosing either south or east was the policy. By iteration 20, the Q value of choosing east decreased to -10.96 but Q-values for other actions were even lesser. So, the policy became choosing east. Next 70 iterations caused the Q value to be -28.65 but the policy is still the suboptimal action of choosing east. 500th iteration made the value of choosing north -48.86 and lessened the values for other actions even more. Hence, the policy became the optimal one of going north. 5000 iterations improved the value for optimal action to -9.72 and by 15000 iterations, the Q value became -13.57. Similarly, the policy and value in several other states evolved to optimality through several iterations. **Another interesting observation** I made is that Fig.10.e and Fig.11.e are identical with respect to utility values and policies of all states. Hence VI and PI converges to the same policy and values in all states. However, the values of states of QL in Fig.12.f are different from the converged results of VI and PI. Though the policies of most states in Fig.12.f are same as in Fig.11.e, some states show difference. For example, the second-last state in last row chooses west (optimal) in Fig.11.e but south (suboptimal) in Fig.12.f. Thus, Q-learning results are sub-

optimal and presumably stuck in a local optimum even after 15K iterations, unlike VI and PI which reached the global optimum by 50 iterations. The reason for this can be that Q-learning has no model or domain knowledge encapsulated in the form of transitions and rewards, unlike model-based planning algorithms of VI and PI. Thus, the model free Q-learning expects to visit all state-action pairs infinitely often so that the truth of the reward structure gets propagated to all the states. Thus, infinite time and infinite exploration should be given to QL to come out of the local optimum. Also, stochasticity of the problem makes the agent unable to execute intended actions, hence state-action pairs cannot be visited strictly as per planned. However, this stochasticity also induces randomness thus helping the agent explore less attractive actions also.

Another interesting observation was made when the discount factor was decreased to 0.6 from 0.99. The iterations needed to converge dropped from 48 to 14 for VI, 15 to 6 for VI and from 14193 to 26 for QL. The runtime till convergence also dropped heavily while the average step count per iteration rose from around 60 to 300. The rewards for all 3 algorithms was -299 when gamma is 0.6, as compared to around 15 with gamma equal to 0.99. The converged policy and values are exactly similar in PI and VI and is shown in Fig.13[LEFT]. For VI and PI, the states near the goal state have good utility values but the reward got highly discounted for the states in the left side in Fig.13[LEFT], and the bottom states in the first column does not even think that going north (optimal) is better than going south. In other words, by the time the final reward value is propagated to bottom-left states, it got too discounted to entice agent to pursue the terminal reward. The performance deteriorated in Q-learning with 0.6 gamma because, similar to in MDP-1, the effective reward value gets highly discounted, thus making it less attractive to pursue, and resulting in the agent hovering over state space for maximum number of steps (300) in an episode and accumulating step costs (-1 per step) and penalizations of falling into pits in the quest for an attractive reward. Fig.14[RIGHT] shows the policies in different states when gamma is 0.6 for QL after 500 iterations. It can be noticed that the Q-values of states in the right half are not updated much as they were not visited much. Since epsilon greedy approach with epsilon 0.1 is used, the agent focusses on exploitation 90% of the time and so when the terminal state reward is not attractive enough, the agent is not motivated to go all the way till right-top corner, thus resulting in a sub-optimal convergence as in Fig13[RIGHT]. Though the agent will eventually understand the goal state and converge to optimal policy given infinite time and infinite exploration, highly discounting the reward will make the process slower and more susceptible to get stuck in local optima as agent is less motivated to find the global optimum. Choosing a random exploration strategy or increasing epsilon in epsilon-greedy approach made it easier for the agent to come out of the local optimum and explore more to learn optimality. Choosing a greedy strategy or making epsilon 0 made the agent never come out of the local optima, since it always exploits (choose best action depending on current Q-values) instead of updating Q-values to true utility values.

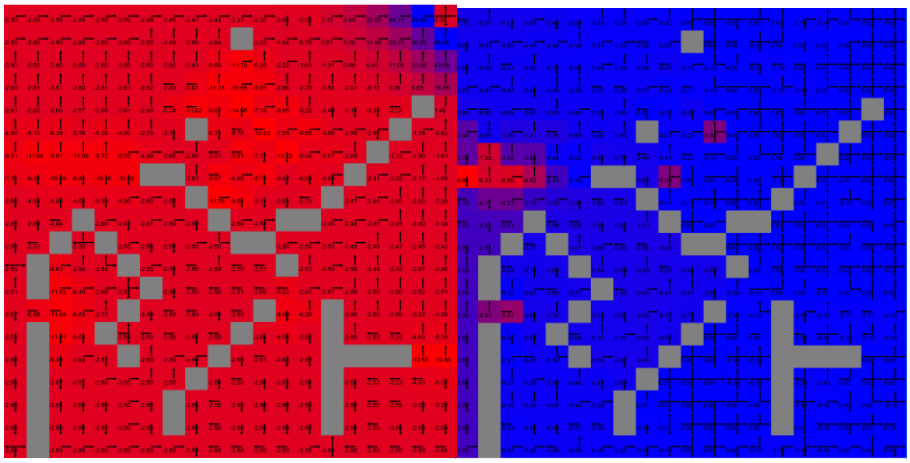

Fig.13.Policy after convergence for VI &PI [LEFT] and QL [RIGHT]

Part 2: The effect of size of Grid on algorithmic performance

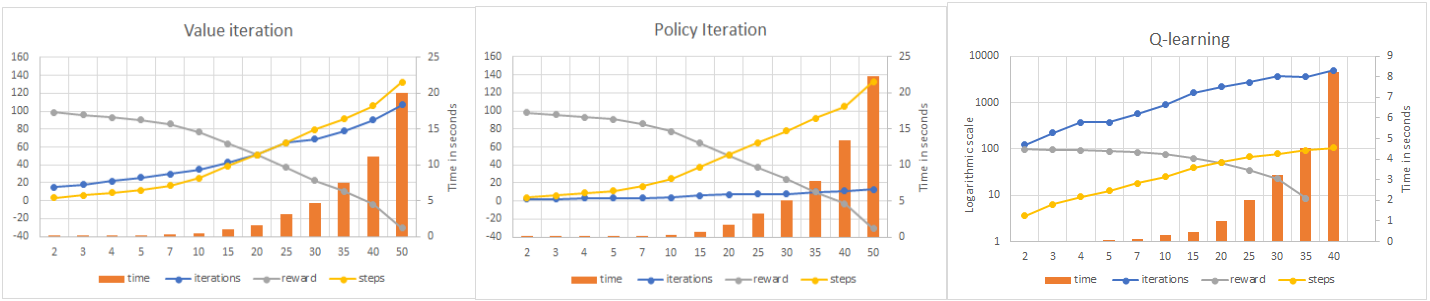

Fig.14.Effect of shape of grid-world on VI, PI and QL respectively

Fig.14 shows the results of VI, PI and QL applied on many grid-worlds with different sizes. Each grid world of shape n has n*n states, all reachable with no walls or pits, thus with very less complexity. The goal is to reach the rightmost top corner from the leftmost bottom corner and each step incurs a penalty of -1. X-axis shows the size of grid world such that 2 indicates a 2*2 grid world. The actions are north, south, east and west. Each action is stochastic such that 80% of the time the intended action gets executed but the rest of time, any of the other three actions can happen with equal probability. For VI in Fig.14.a, as size of grid world increased, step count (yellow line) increased, and number of iterations needed to converge (blue line) increased, thus increasing the overall runtime (orange bar) for VI. As number of states increases, since the value and policy in each state must be updated in each iteration, the

runtime also increases. As the state-count increases, the number of steps required to reach top-right block (terminal state) from bottom-left block (starting state) increases, thus increasing the required step-count. Also, since the steps increased, and each step has a penalty of -1, the reward value decreased (grey line). PI also shows similar trends, but the number of iterations needed to converge has a lesser slope than VI. That is, when the shape of grid world is doubled, the increase in number of iterations for VI is more pronounced and higher than in PI. But the increase in runtime with shape of grid world is steeper in PI. This is because PI generally needs fewer iterations and higher runtime till convergence, than VI due to earlier explained reasons. For QL, the number of iterations needed to converge is very high compared to VI and PI, as expected. As shape of grid world increased, the step count and iteration count increased with a good slope. And due to the increasing step count and corresponding penalization of -1 per step, the reward decreased.

Part 3: Comparing various exploration strategies on Q-learning

As explained in the experimental methodology, "epsilon greedy approach" was used for the above experiments, where the exploration action is chosen uniformly randomly from the set of all possible actions. "Boltzmann's approach (softmax exploration)" assigns a probability to the actions to create a graded function of estimated values. The approach has a temperature parameter T which when equals to 0, leads to exploitation and when equals to infinity, the agent does full-fledged exploration. This simulated annealing kind of approach makes the agent select actions according to the assigned probability when T has intermediate values. Another policy, exploitation-intensive "greedy Q policy", where the action with maximum Q value is greedily chosen always, to maximize rewards and in case of a tie, the actions are randomly chosen from the tied actions, is also experimented. Random Policy where the actions are always randomly chosen is also experimented.

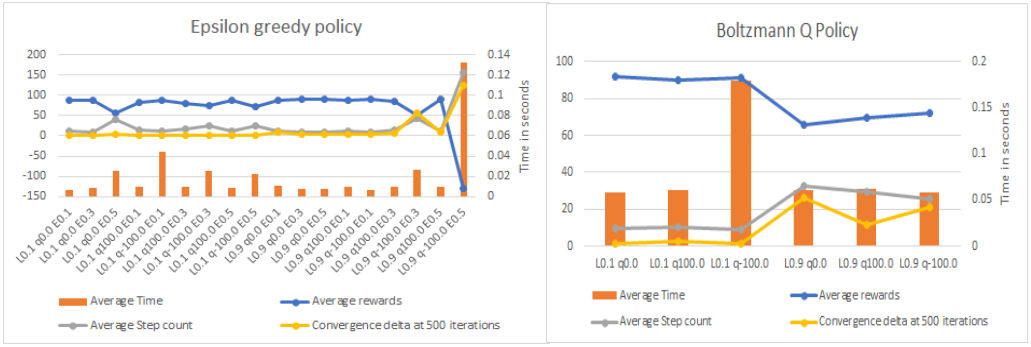

Fig.15.Hyperparameter tuning for Epsilon greedy policy [LEFT] and Boltzmann Q policy [RIGHT] for MDP-1

MDP-1 hyperparameter tuning: In Fig.15.a, L=0.1, q=0.0 and E=0.1 are chosen as the best hyperparameters because they gave one of the highest rewards within one of the least execution times and least step counts and converged within 500 iterations. For Boltzmann Q policy, in Fig.15.b, L=0.1, q=0 gave highest rewards within one of the least execution times and least step counts. Also, convergence is obtained within 500 iterations. In Fig.16, for greedy-Q policy, L=0.9 and q=-100 performed the best, and L=0.1, q=0.0 performed the best for Random Policy.

MDP-2 hyperparameter tuning: The epsilon-greedy hyperparameter tuning gave best results for L=0.1, Q initial values =0.0 and epsilon =0.3. Greedy Q policy also gave best results when L=0.1 and Q =0.0. Due to space constraints, the hyperparameter tuning plots of individual exploration strategies are only shown for MDP-1 and only results are described for MDP-2. When learning rate is higher, around 0.9, the agent gets stuck in a local optimum of reward value (-299) in 300 steps for both exploration strategies. Lower learning rates take a longer time to converge because of the very small steps taken towards optimum. Initializing all Q values to -100 also makes the agent get stuck in that local optimum. Initializing

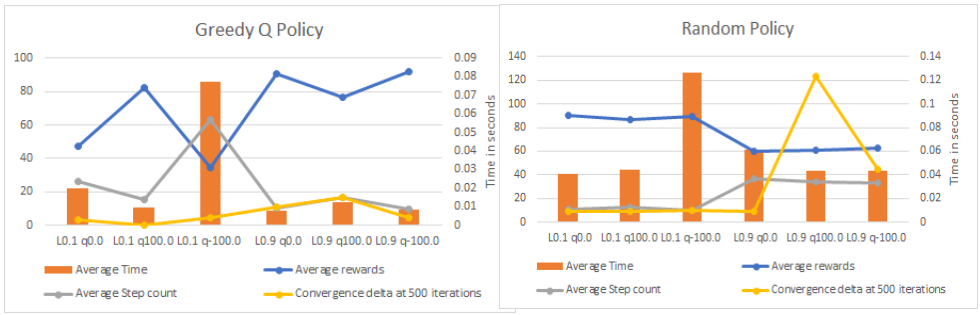

Fig.16. Hyperparameter tuning for Greedy Q policy [LEFT] and Random policy [RIGHT] for MDP-2

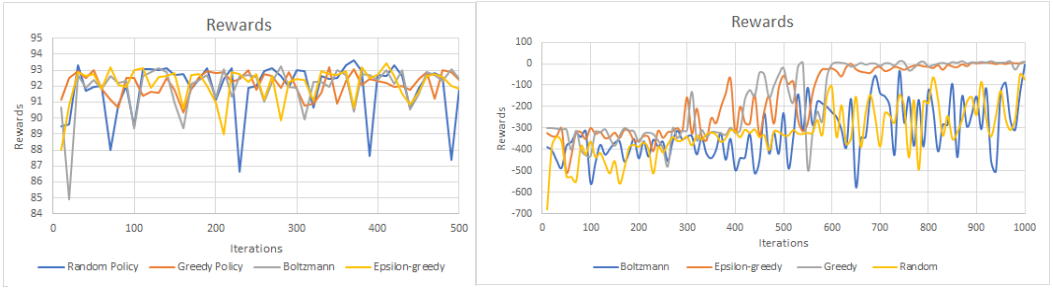

Fig.17.Rewards for MDP-1 and MDP-2 using various exploration strategies for QL

Q-table values as 100 gives negative reward of -70 in around 80 steps. Initializing all Q values to 0, with a learning rate of 0.1 gives a reward of 10 on average, in 60 steps, for epsilon-greedy and greedy-Q strategies, and epsilon=0.3 gives the most consistent results for epsilon-greedy strategy.

For random policy, none of the hyperparameter setting gave good results. However, L=0.9 and q=0.0 gave comparatively better results. Lower learning rates of 0.1 and initializing Q values to -100 always made the agent get stuck in the local optimum of achieving negative reward of -299 in 300 steps. Initializing Q values to 100 resulted in a worse reward of -350 in 290 steps. When Q values were initialized to 0 with learning rate set as 0.9, an average of -100 reward was obtained in around 100 steps. Regarding Boltzmann policy, setting learning rate as 0.1 with initial Q values as -100 landed the agent in a local optimum of -299 reward in 300 steps. With the same learning rate, when the initial Q values were changed to 0 or 100, the reward worsened to -350 with the same number of average steps. When learning rate is high (0.9), setting initial Q values to 100 yielded similarly bad results but setting it to 0 improved rewards to -150 with an average step count of 70, and setting it to -100 gave -100 reward in 60 steps. Thus L=0.9, Q=-100 are selected as best hyperparameters for Boltzmann strategy.

In Fig.17.a, random policy shows most fluctuations in rewards, because of its exploration nature which does not check for the best action at all. Epsilon-greedy and Boltzmann also fluctuates indicating the randomness in action-selection of those strategies. Boltzmann policy shows large fluctuations in the beginning because the temperature parameter is high, and it induces more exploration making it behave like Random Policy. As temperature decreases with iteration count, the agent focusses on exploitation and acts like in greedy policy. For greedy policy, the Q values need not be accurate initially and as iteration count progresses, the Q-values become more closer to the true utility values of the states and then, the reward increases, and fluctuations decreases as agent always chooses the best action in a state. For MDP1 in Fig.20.a, none of the policies converge to a fixed reward but except random policy, all others fluctuate within the band of 90 to 93 in later iterations. The stochasticity of the problem causes randomness at any iteration and this explains the minor fluctuations in rewards in the later iterations. In Fig.17.b, both random and Boltzmann policies show high fluctuations due to randomness in action-selection, and greedy and epsilon-greedy converges to a reward of around 10 as iteration count increases. Although Q-learning does not give a reward as good as VI or PI for this MDP, epsilon-greedy and greedy policies do the best among exploration strategies for Q-learning. For MDP1, in Fig.18.a, though epsilon greedy takes more steps in the beginning, in the later iterations, it takes lesser steps than others. For random and Boltzmann policies, the step count highly fluctuates for all iterations in Fig.18.b, due to their randomness in action-selection. The temperature of Boltzmann has not decayed enough within the iterations shown, hence it acts exploratory. Higher iteration counts showing Boltzmann's low T performance, are not included in the plots because high iteration count of graphs will compress the prominent early-iteration characteristics of other policies. In Fig.18, the greedy and epsilon greedy policies converge for MDP-2 to around 60 steps, while for MDP1, all policies converge to a band of 8 to 11 as iterations increases. Stochasticity of MDPs explain the minor fluctuations in later iterations. Fig.19 shows the cumulative execution time for different exploration stragies. Boltzmann and random take more execution time in both MDPs, because their exploratory nature causes them to take more steps in each episode, and mostly they end up using maximum allowed steps in each episode, thus increasing overall excution time. For MDP-1, epsilon-greedy and greedy policies found the optimal policy very early (within 5 iterations) and hence their runtime per iteration decreased for iterations after convergence. For MDP-2, all policies took similar time in early iterations, because being a complex grid world, it took

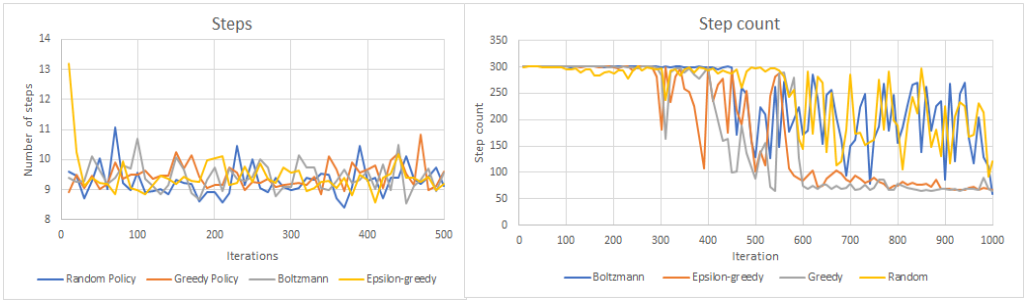

Fig.18.Step counts for MDP-1 and MDP-2 using various exploration strategies for QL

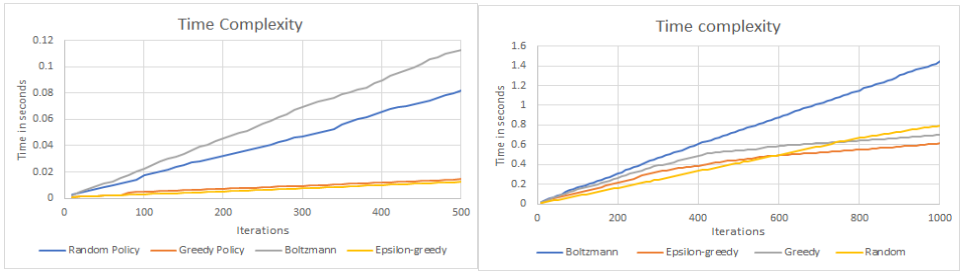

Fig.19.Time complexity for MDP-1 and MDP-2 using various exploration strategies for QL

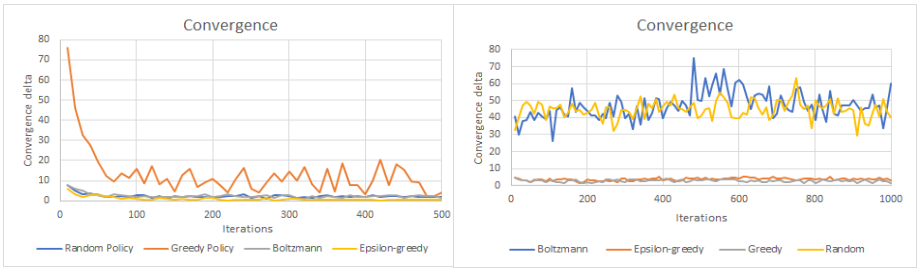

Fig. 20. Convergence for MDP-1 and MDP-2 using various exploration strategies for QL

them all reasonable number of iterations to find optimal policy. As optimal policy is found in more and more states, the execution time for greedy and epsilon-greedy policies kept on decreasing as optimal policy helps to reach terminal state in fewer steps, thus decreasing the slope of the cumulative time plot in Fig.19.b. Once best policy is found in all states, the runtime per iteration became so less that the cumulative plot in Fig.19.b appears slightly plateaued(slope very low) at around 800 iterations. So in both MDPs, epsilon-greedy and greedy policies are best in terms of execution time. For MDP-1, greedy policy does not show convergence in Fig.20.a, while all the other three algorithms converged within 50 iterations. The best fitting hyperparameters of greedy policy for MDP1 was learning rate =0.9 and initial Q values =-100. The high learning rate can cause instability in the Q-table values and overshoot the optima. Also, initializing all Q-values to -100 (low values) makes all actions equally less attractive. Since true utility values (from VI or PI in Fig.11.e) of all states are much higher, converging to those values might take more iterations. So, an intuitive explanation is that the policy or the relative order of Q-values of actions in each state, are found in greedy Q policy early enough, thus giving high rewards in less steps (Fig.17.a and Fig.17.b), but the convergence of Q-table values to exact true values took a lot more iterations (Fig.20.a) presumably because of the instability brought in by the high learning rate (0.9). Also, when the learning rate was decayed with time, greedy policy was found to converge early like other exploration strategies. In Fig.20.b, for MDP-2, random and Boltzmann also does not show convergence as their learning rates are 0.9 each (found from hyperparameter tuning) and, Boltzmann strategy shows lots of fluctuations because its Q-table was initialized with very low values (-100) and needs many iterations to reach the true values (values in Fig.11.e). Greedy and epsilon-greedy converges as their learning rate was 0.1 (low) thus making smaller and stable steps towards true utility values and Q-table was initialized with 0 which is nearer to true utility values (Fig.11.e) than -100. Overall, epsilon greedy is the best policy for MDP1 because it shows less fluctuations in rewards than random and Boltzmann policies, and gives high rewards like greedy policy, and shows lesser execution time and step count, and converges early enough. Greedy policy would be the second best because it performed like epsilon greedy with respect to rewards, runtime and step-count except that it did not converge early. For MDP-2 also, epsilon greedy did best regarding rewards, runtime, step count and convergence closely followed by greedy policy.

CONCLUSION:

Policy iteration is a planning algorithm that is guaranteed to converge using fewer iterations than VI. However, runtime per execution is high for PI and this makes VI attractive in certain scenarios. VI is also guaranteed to converge to optimal values. QL is useful when there is no model of the environment. So, if the domain knowledge to build the model is absent, model free RL methods like QL can prove useful. In large state space scenarios, PI can take many iterations as there are many possible policies. Also, regarding VI, updating values of all states, after the time-expensive process of policy extraction by maximizing over all actions in a state, in each iteration, is computationally expensive. So, function approximation methods like Deep Q-learning can be used in such cases, even when state-space is continuous. And as it generalizes earlier experiences to previously unseen states, it quickens the learning. Since model based approaches have more parameters to learn, namely transitions(T(s, a, s')) with $|S|^2|A|$ parameters and rewards(R(s, a)) with |S||A| parameters, where |S| denotes number of states and |A| denotes number of actions, and model-free approach has only |S||A| parameters (Q(s, a) for Q-learning) to learn, when the number of state or actions are huge, model-free methods are faster. Given infinite time and infinite exploration, Q-learning converges to optimal values. Practically, provided the learning rate is sufficiently small or decreased over time, Q-learning converges, given enough iterations and exploration. To conclude, comparing algorithmic performances in MDP-1 and MDP-2, PI took least time (20 microseconds) and iterations (4) for MDP-1, QL took most iterations but lesser time than VI and VI took most runtime to converge, due to reasons explained in part-1. For MDP-2, VI took least runtime (1.06 seconds) for convergence followed by PI (2.41 seconds), though PI took least iterations. QL did not give satisfactory convergence even after 501000 iterations in MDP-2. Regarding runtime per iteration, QL takes the least, followed by VI and then PI, for both MDPs. MDP-1 converged to same optimal policy and values for all 3 algorithms, and MDP-2 converged for PI and VI to the same optimal policy and values in reasonable number of iterations but only showed signs of convergence in QL even after 501000 iterations, as detailed in part-1. Overall, this assignment has been a great opportunity to experiment with and learn more about planning and reinforcement learning algorithms in relation with MDPs. This assignment taught me how to apply reinforcement learning techniques for decision-making in real-world scenarios.

REFERENCES:

1. BURLAP (2017). MacGlashan, J., Retrieved from https://github.com/jmacglashan/burlap
2. Tay, J., Github (2017). Retrieved from https://github.com/JonathanTay/CS-7641-assignment-4
3. BURLAP tutorials (n.d.). Brown University. Retrieved from http://burlap.cs.brown.edu/tutorials/cpl/p1.html
4. Sutton, R., and Barto, A., 2018. "Reinforcement learning: An Introduction". Chapter 4, pp.89-107