Havish Chennamraj
903201642

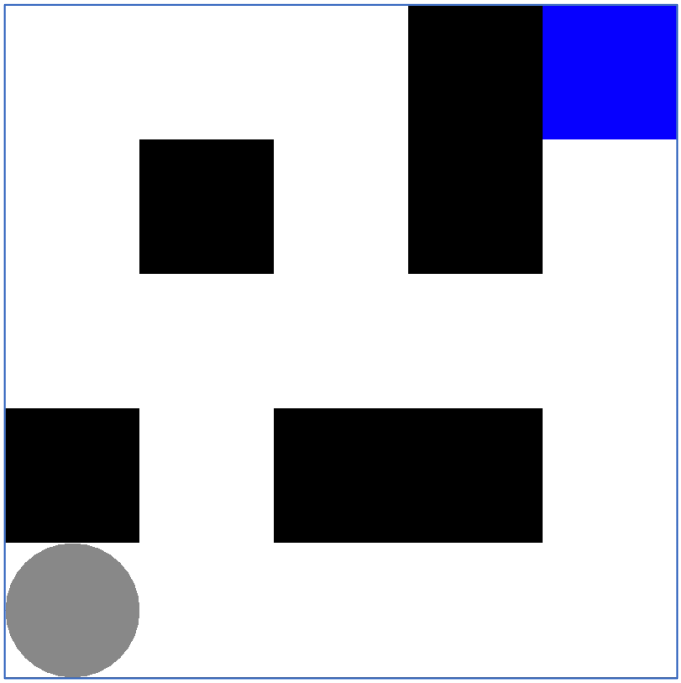
Assignment 4 – Markov Decision Processes

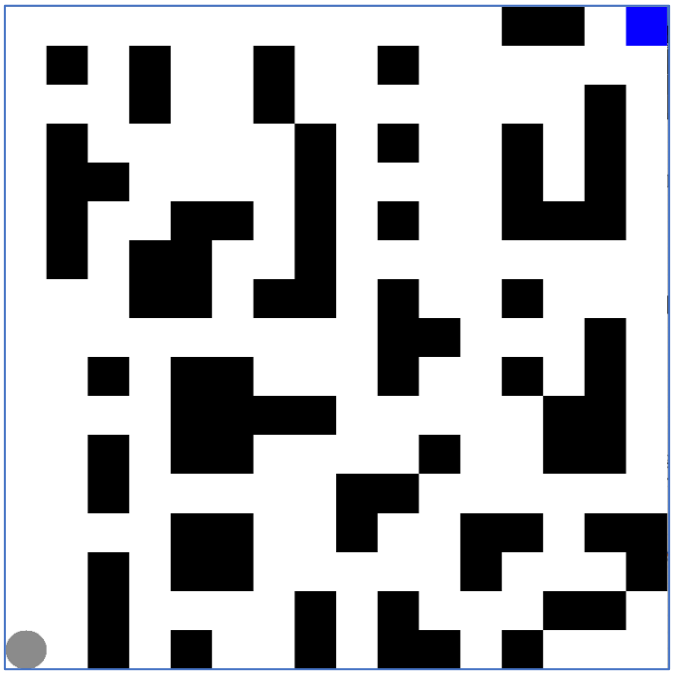
1. Abstract

This purpose of the report is to explore and draw a comparative analysis of the performance of two planning algorithms namely Value Iteration and Policy Iteration and one learning algorithm Q-Learning w.r.t two interesting Markov Decisions which shall be discussed in greater detail further in the report.

2. MDP Problems Description

Both the MDP problems chosen for this Assignment are Grid worlds where an agent has to navigate towards a goal which happens to the top-right most cell. One Grid world is of size 5 X 5 and the other is of size 16 X 16. The size of gird worlds has been chosen as such to characterize the two types of MDP problems, one with small number of states (25) and the other with large number of states (256) so that analysis of the learning algorithms is more exhaustive. It is also important to note that there are few cells in both worlds which the agent can't enter. The reward function is the same for both problems: each action of the agent incurs a small penalty of -1 point and gets 100 points for reaching the goal state. Up, down, left and right are the only permissible actions. Below is the visual representation of both the Grid Worlds.





The 'blue cell' indicates the goal, 'black' cells indicate the ones into which the agent can't move and the agent itself is depicted as a 'grey' circle. The objective of the algorithms is to find the most optimal path to the goal

such that the reward is maximized. One of the reasons the problems albeit basic are interesting because although they can be solved by Dynamic Programming, it is fascinating to watch and visualize an algorithm learn to navigate its way through a maze. Another reason I believe the problems are interesting is because of how easily they can be plugged into real life scenarios where an agent is expected to navigate an unknown terrain like in military conflicts for reconnaissance, search and rescue operations, space missions etc. One has to truly comprehend the scope of scenarios the solutions of these problems can be plugged into to actually appreciate them.

3. Algorithms

3.1 Value Iteration

Value Iteration aims to find the most optimal policy by iteratively improving the utility of the states using the Bellman equation. The algorithm starts off with random utility values for all states, and then updates the utilities of the states based on the neighbors using the equation $\boldsymbol{U_{t+1}(s) = R(s) + \gamma max_a \sum_{s'} T(s, a, s') U_t(s').}$ This iterative process is repeated until utilities converge to optimal values. The optimal policy at 't'-th iteration can be found from the computed utilities using the equation: $\pi_t(s) = \text{argmax}_a \sum_{s'} \boldsymbol{T(s, a, s') U_t(s').}$

3.2 Policy Iteration

Since the ultimate goal of the problem is to find the most optimal policy, Policy Iteration achieves the same by iteratively making the policy better. It starts off with a random policy and computes the utilities of all states for iteration 't' based on the policy for that iteration using the formula $\boldsymbol{U_t(s) = R(s) + \gamma \sum_{s'} U_t(s') T(s, \pi_t(s), s').}$ The computed utilities are then utilized to frame the policy for 't+1'-th iteration using the formula $\pi_{t+1}(s) = \boldsymbol{\sum_{s'} T(s, a, s') U_t(s').}$

It is believed that the policy iteration converges in lesser number of iterations as compared to value iteration, but every step of policy iteration is computationally more expensive. Consequently, it might take more time for Policy Iteration to converge as compared to Value Iteration.

3.3 Q-Learning

Q-Learning is a model free Reinforcement Learning algorithm which starts off by assigning random Q-Values to all state, action pairs and iteratively improves them using the bellman equation by revisiting the state, action pairs. There's also an exploratory nature to Q-Learning algorithm determined by epsilon 'ε' where the algorithm tends to take a random action with probability 'ε' instead of exploiting the already built Q-Table. This 'ε' is designed to decay over iterations. In other words, the algorithm tends to be more exploratory in the initial phases and eventually adopting a more 'exploitative' nature. One has to acknowledge the importance of this nature of the algorithm because any learning algorithm which tends to be only exploratory by acting on random choices isn't technically learning anything. The same can be said about an algorithm which relies solely on what it has learnt (exploitation) because it is susceptible to reinforce its wrong learnings or get stuck at a local optimum. Q-Learning strikes a healthy balance between the two and thus is a very versatile learning algorithm.

4. Experiments and Analysis

4.1 Small MDP

4.1.1 Value Iteration (VI) and Policy Iteration (PI)

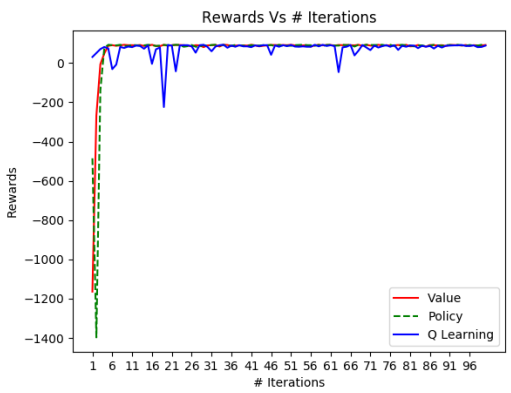


Fig 1.1

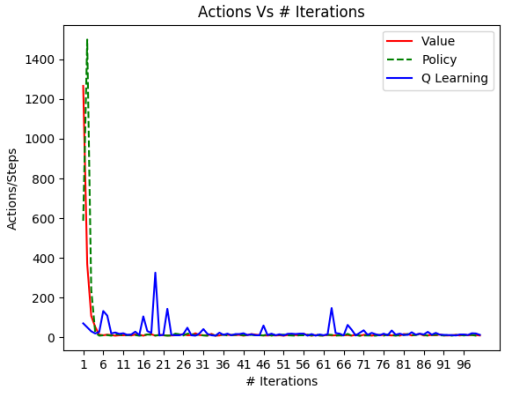


Fig 1.2

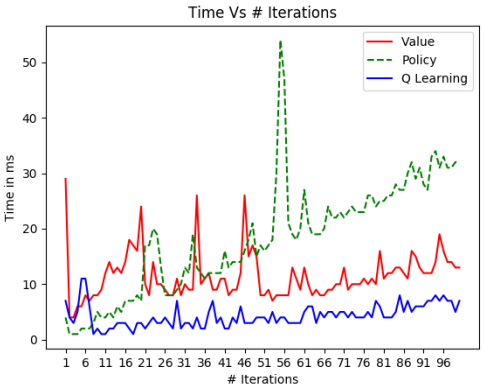
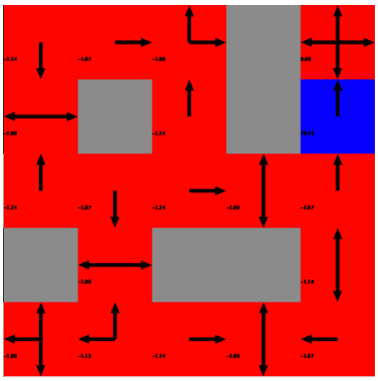


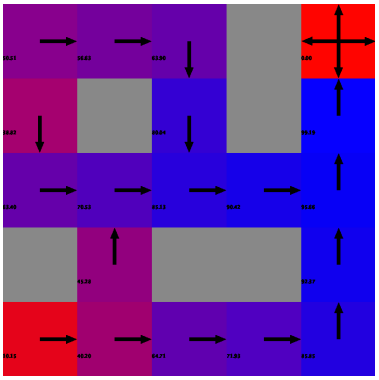
Fig 1.3

The above graphs plot the total reward, #actions & time taken against #iterations for all 3 algorithms. We shall first compare Value Iteration (VI) and Policy Iteration (PI) against each other and then take a much deeper look into Q-Learning. From the above 'Rewards' plot, we can see that both VI and PI start off with a large negative utility value but quickly converge to the optimal at around 6 iterations. Consequently, both VI and PI start off with huge #actions but quickly converge to optimal number of actions around the same 'iteration' mark of 6. This behavior is expected because both algorithms start off with random values which obviously are very likely to be far away from the optimal solution. But as more truth is added every iteration, they converge to the optimal solution. It is interesting to note that although the behavior of VI and PI is graphically the same, the time taken by PI on an average across iterations is around 2x times greater than VI which is expected because every iteration in PI is computationally more expensive.

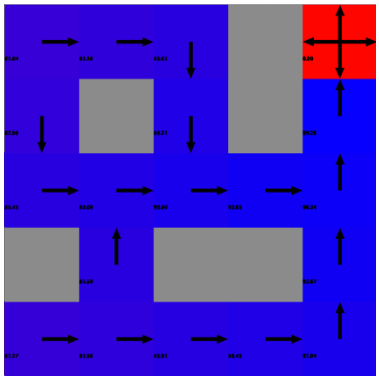
Below are the policy snapshots for both algorithms at I=1,6,20 and 100.



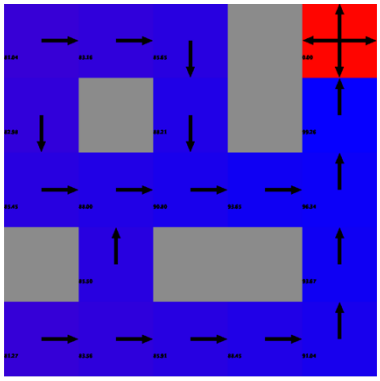
(1) VI, I = 1



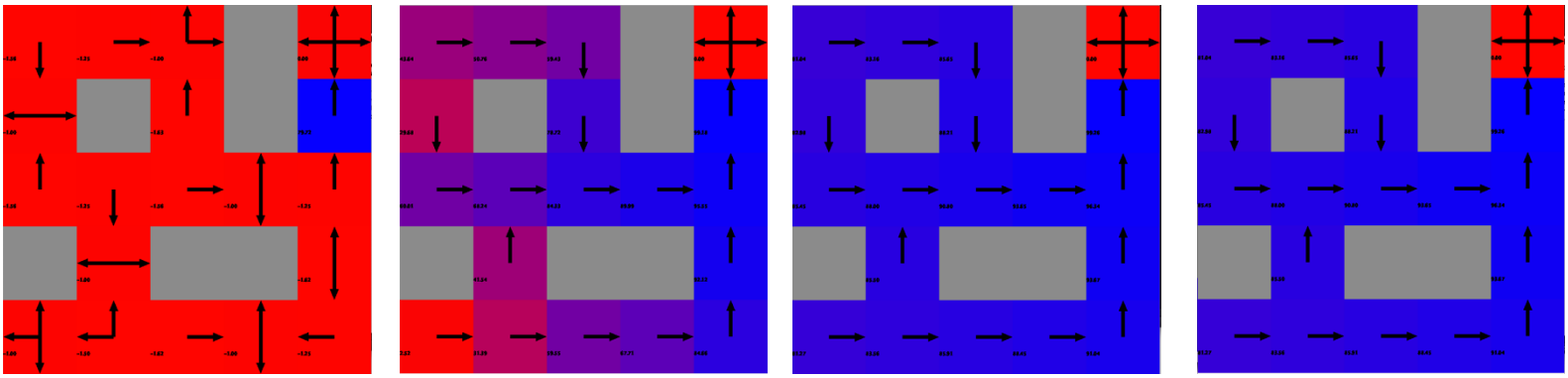
(2) VI, I = 6



(3) VI, I = 20



(4) VI, I=100



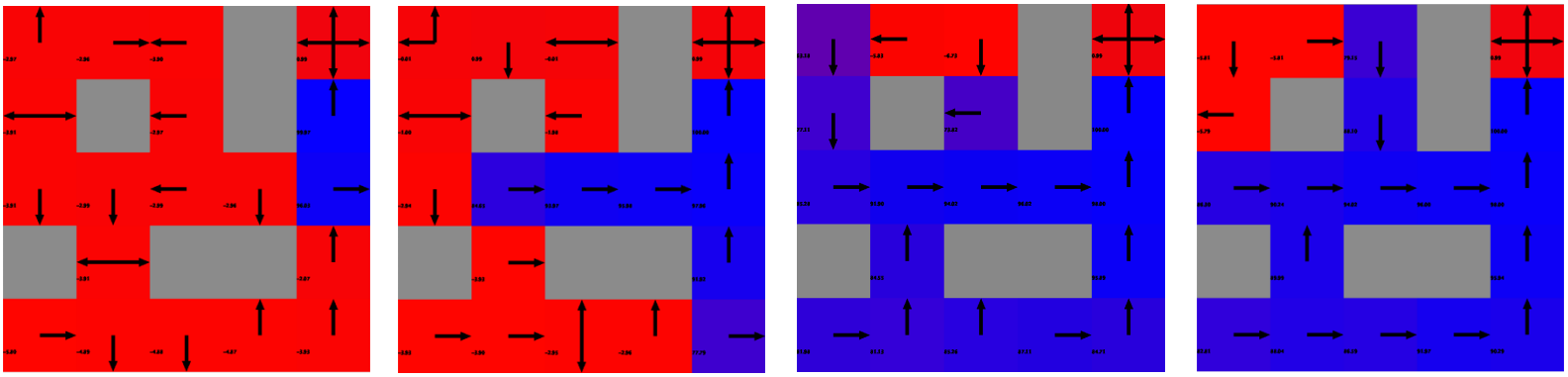
(5) PI, I = 1 (6) PI, I = 6 (7) PI, I = 20 (8) PI, I=100

Figure 2.1

The behavior of VI and PI seem to be quite similar. At I=1 for VI we can clearly see the utilities of neighboring cells of the goal (top right cell) to be updated with the reward while other cells are still in red (arbitrary actions). VI continues to evaluate the utilities of all the states until it arrives on the optimal path at I=6. The exact same applies for PI as well. However, both haven't found the optimal policy yet. From I=6 to I=20 both the algorithms strive to slightly improve the policy by avoiding walls (cells the agent cannot enter marked by grey in the above plots) because bumping into wall incurs an unnecessary cost of -1 without making any progress. The policy and consequently the utilities remain fairly constant from I=20 to 100 for both VI and PI, and thus it can be argued that both the algorithms attain true convergence with respect to policy around I=20. The only difference that is worth pointing out is the VI has higher utilities for states as compared to PI at I=6 which suggests that VI was likely closer to the optimal solution as compared to PI although both found the optimal path at the same iteration mark. Also considering that VI takes lesser time than PI, VI should clearly be the choice of learning algorithm for the small MDP problem.

4.1.2 Q-Learning (QL)

Fig 1.1, 1.2 and 1.3 contain plots for QL as well. In the interest of saving space, please refer to those plots for QL as well. From Fig 1.1 we can see that Q-Learning converges to the same reward as VI and PI. However, it has random dips at certain iterations which is because of the exploratory nature of the algorithm. In an attempt to search more of the state space, the agent ends up in sub-optimal states thereby reducing the total reward. It is also interesting to note from Fig 1.3 that time taken remains fairly constant with increasing iterations for Q-Learning because it's a model-free algorithm. Below are the policy development snapshots for QL at I=1,6,20 and 100.

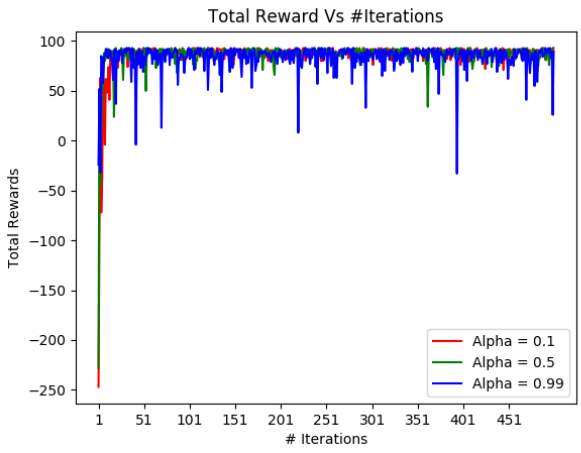


(1) Q, I = 1 (2) Q, I = 6 (3) Q, I = 20 (4) Q, I=100

Figure 2.2

The behavior of QL is considerably different from PI and VI. Unlike VI and PI, QL still hasn't figured out the optimal path at I=6 although it has converged on the reward. At I=20, QL has figured out the optimal path, but is yet to decide on the optimal policy because it strives to improve the utilities beyond I=20 mark. I=20. QL reaches a policy at around I=37 which 'almost' remains the same up until I=100, and thus can be considered as the point of convergence w.r.t policy. This is almost twice of VI and PI. However, because of the low computational costs of QL iterations, it turns to be faster than both VI and PI. Also, considering that QL doesn't have the luxury of exploiting any domain knowledge such as the reward or cost function it has performed reasonably well. Although I've claimed that at I=37, QL attains convergence w.r.t to policy it is important to note that policy was still subjected to slight changes in later iterations. The unwillingness of QL to settle on a policy is because of its exploratory nature which also explains the random dips in the total reward attained. The only difference I can see between QL and model-based learning algorithms is that the top-left quadrant of the Grid World has low Q values even at I=100. This shows that the Q-Learner hasn't learnt anything new while exploring those states.

The above experiments for QL were conducted with default values: ε (epsilon) = 0.1, γ (discount factor) = 0.99 and α (learning rate) = 0.99. High discount factor made QL value future rewards almost as equally as immediate rewards, High learning rate made Q-Learning disregard previous Q-Values. The graphs below evaluate how 'total reward' and 'time taken' change for varying values of ε, γ & α.



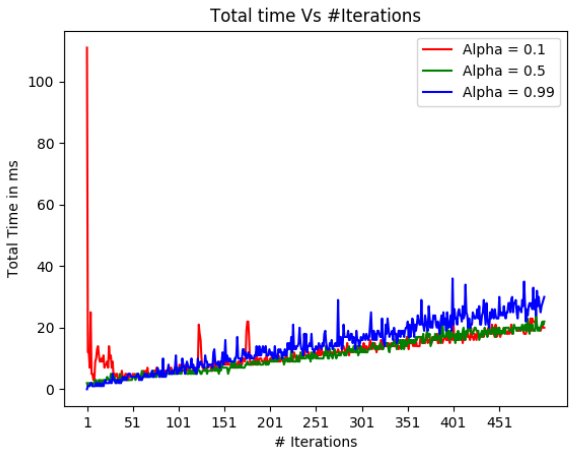
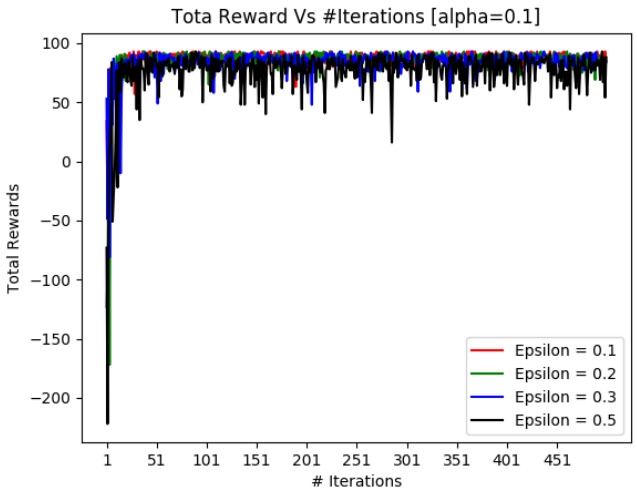


Fig 3.1

Fig 3.2

Above graphs plot the 'total rewards' and 'time taken' across iterations for learning rates 0.1, 0.5 and 0.99. Although the performance on both the metrics is quite similar, α = 0. 1 seems to give the best results from what we seen with least amount of fluctuations on the total reward and lesser time taken. So we proceed with learning rate set to 0.1.



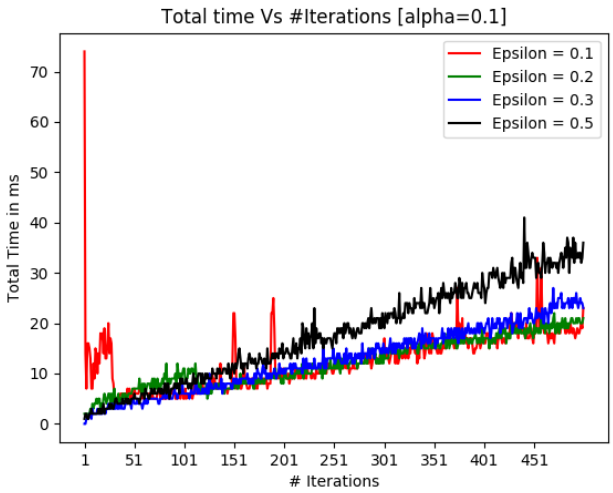
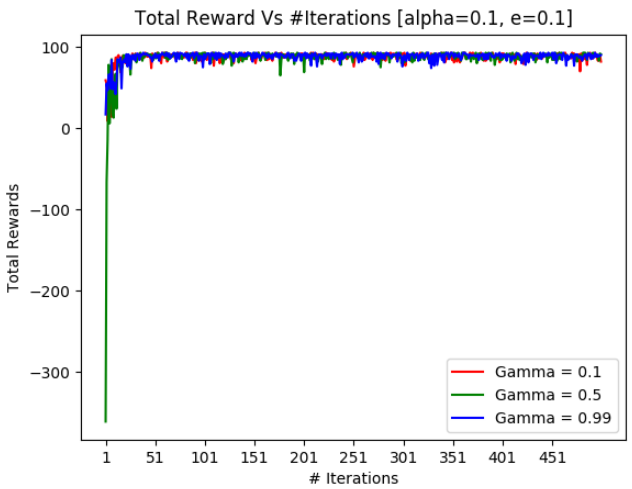


Fig 3.3

Fig 3.4

Above plots evaluate 'total rewards' and 'time taken' by the QL for varying values of ε (0.1, 0.2, 0.3 and 0.5). Again, not much of a difference is noticed owing to small state space. We see more fluctuations for ε=0.5 because the algorithm tends to be more erratic because of its higher explorative tendency. Considering the small MDP problem has only 25 states and ε=0.1 takes the least amount of time, 0.1 seems to be the best choice for epsilon. Also, since the problem is relatively simple with two straight forward optimal paths, a more exploitative approach should logically perform better than an explorative one and this agrees with our choice of ε.



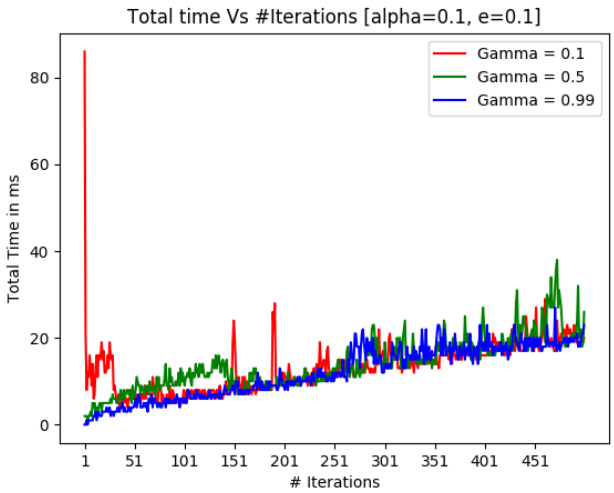


Fig 3.5

Fig 3.6

I conducted the above experiment to evaluate the performance of QL for different discount factors γ = 0.1, 0.5 and 0.99 with the following hypothesis[1]: Increasing the discount factor has the potential to improve the quality of the learning process. However, probably because of our small state space again, no significant difference is shown among different γ values. Going with the cited paper, I can only recommend a high value of γ around 0.9 for now. It remains to be seen if changing any of the above 3 parameters produces a viable difference in performance of QL for the large MDP problem.

4.2 Large MDP

Large MDP like mentioned above is a Grid World of size 16 X 16 with 256 total possible states. Except for the increased number of states, the problem set up is the same with respect to permissible agent actions and the reward function. Also, the max number of iterations has been set to 500 to give more breathing space to the algorithms for a better comparative analysis.

4.2.1 Value Iteration (VI) and Policy Iteration (PI)

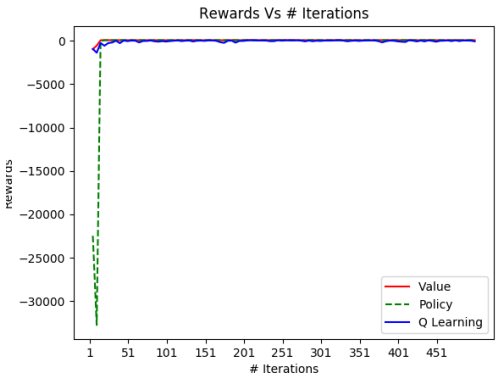


Figure 4.1

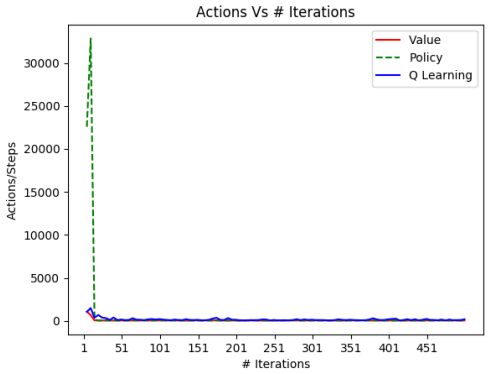


Figure 4.2

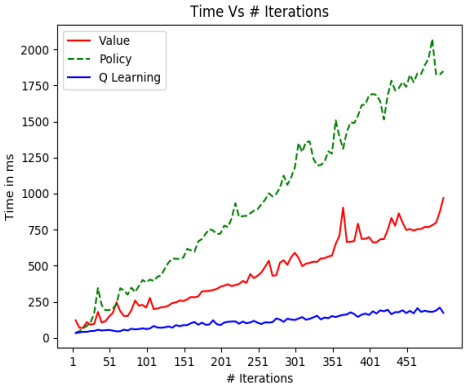
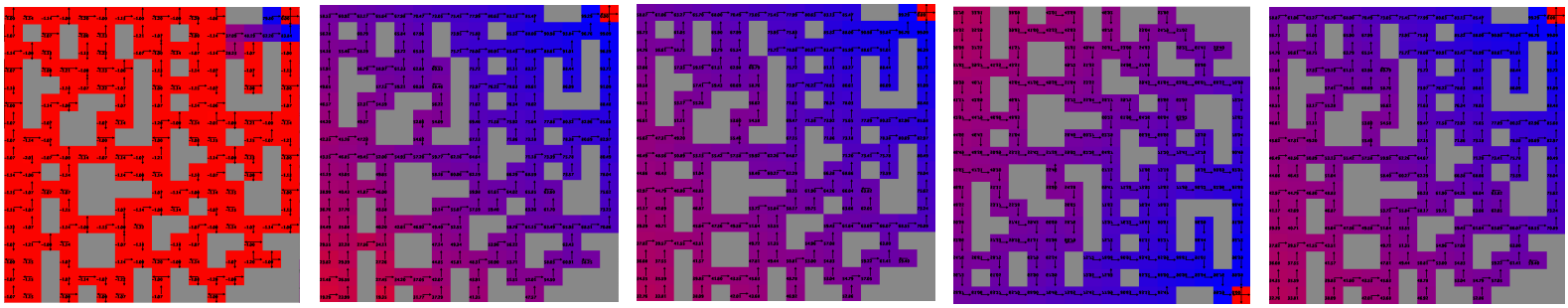
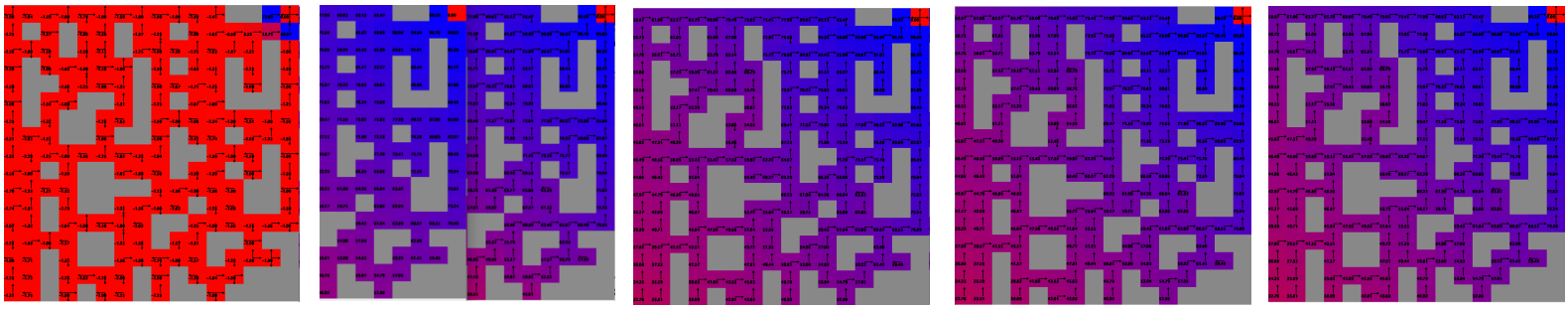


Figure 4.3

The rewards and actions for both PI and VI converge around the same iteration mark I=20 at around 62 and 43 respectively. Just like in the case of small MDP, both algorithms start off with huge negative values especially in the case of PI but quickly converge on rewards and actions. Clearly increasing the number of states has an effect on the performance of the algorithms: the number of iterations to converge and the time taken also significantly increased for both VI and PI. The slope of 'time taken' for PI seems to be much steeper than VI which reinforces the fact that one iteration in PI is computationally more expensive than VI. Below are the policy development snapshots at I=1,20,50,100 and 500.



(1) VI, I=1 (2) VI, I=20 (3) VI, I=50 (4) VI, I=100 (5) VI, 500



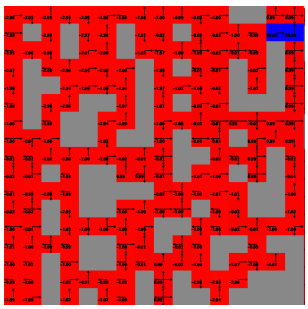
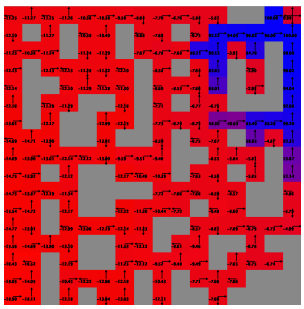
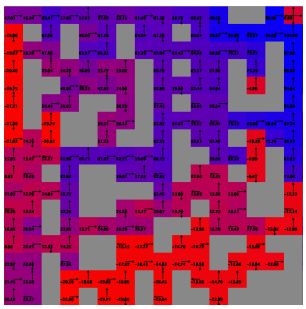
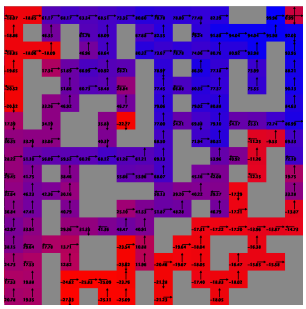
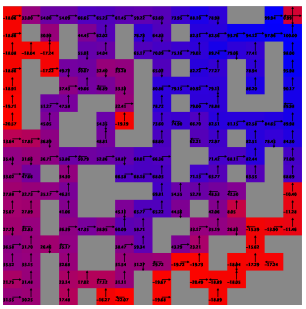
(6) PI, I=1 (7) PI, I=20 (8) PI, I=50 (9) PI, I=100 (10) PI, 500

Figure 5.1

Just like in the case of small MDP, the policy development for VI and BI seems to be quite similar here as well. At I=1, the utilities of states for VI and policy for PI are randomly assigned and consequently all the cells have low values with arbitrary actions. We've noticed that both PI and VI converge on a reward of 62 at I=20, and therefore expect it to have found the optimal path at that point. The above plots at I=20 support this conjecture. It is important to note that there is a gradual improvement in utilities and consequently the optimal policy developed for both VI and PI from I=20 to 50, where both the algorithms strive not to bump into walls (grey cells) because that would just incur a penalty of -1 whilst not making any progress. This phase of minor improvement of the optimal policy was also noticed in the small MDP problem, but the phase was much short-lived w.r.t to iterations as compared to large MDP. This goes on to prove that increased number of states warrants greater number of iterations to find the best possible policy. Considering that the policies remain fairly constant beyond the I=50 mark, we can safely establish that both VI and PI achieve true converge w.r.t to policies at that point.

4.2.2 Q-Learning

From Fig 4.1 it is evident that Q-Learning converges on the reward at around I=50 iteration mark. Also, converged total reward of QL is around 38 as compared to 62 for VI and BI, and sometimes dipping close to 0 for reasonably high number of iterations as well. The random dips and the unwillingness of QL to settle on a policy can be attributed to tendency of the algorithm to explore the state space by acting randomly sometimes. However, the advantage of QL with respect to 'time taken' is far more visible in this problem in Fig 4.3: 'time taken' remains fairly constant even at high iterations while for VI and BI it increases linearly.

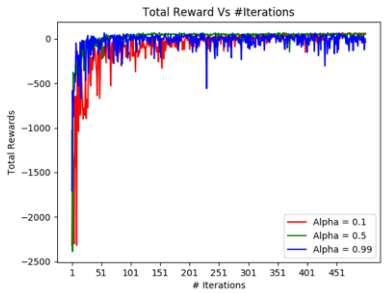






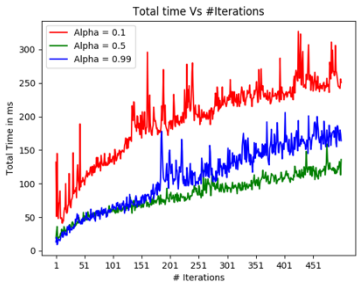
(1) Q, I=1 (2) Q, I=20 (3) Q, I=50 (4) Q, I=100 (5) Q, 500

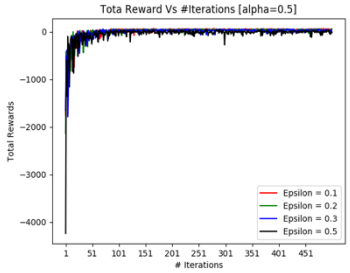
Figure 5.2

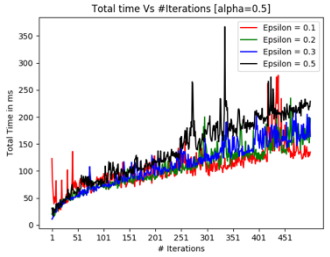
Above plots show the policy development for QL across 500 iterations. At I=1, owing to the large state space QL hasn't really learnt anything to take any meaningful actions even at the states closer to the goal. At I=20, by virtue of exploitation QL was able to figure out optimal actions for states closer to the goal. At I=50, QL was able to find the optimal path to the goal and thus consequently started converging on the 'total reward' as discussed above. Just like VI and PI, it continues to improve its policy beyond I=50. The policy snapshot at I=100 and I=500 are almost the same with minor fluctuations popping up between 100 and 500 because of the explorative nature of the algorithm which leads us to believe that QL was able to zero in on an optimal policy somewhere between I=50 and I=100. The policy development for QL remained fairly similar to the QL in small MDP problem. Other than a slightly diminished total reward and faster computational time, the only difference I can make out between QL and the other two algorithms is its reluctance to learn anything new from the states away from the optimal path. If you look closely at the policy plot for I=500, you notice a fair number of red cells with low Q values on the top-left and the bottom-right quadrants of the grid which happen to the farthest from the optimal path connecting the top-right and bottom-left points diagonally. This shows that QL doesn't really learn anything from the discounted and the immediate rewards of such states. However, VI and PI have a policy which redirects the agent back to the optimal path in case it happens to find itself in such sub-optimal states.

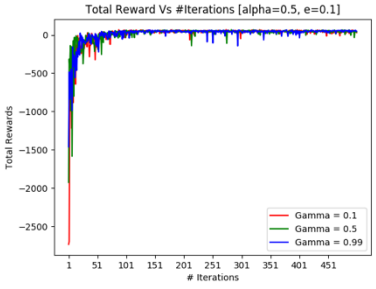
The graphs below evaluate how 'total reward' and 'time taken' change for varying values of ε, γ & α.

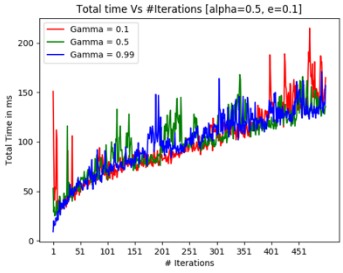












The first two plots in the first row show the performance of QL for varying values of α (0.1, 0.5, 0.99). Just like small MDP problem, nothing much can be inferred here. Looking at the plots suggests that α=0.5 is the best value. Second two plots evaluate the effect of varying ε (0.1,0.2,0.3 and

0.5). Again because of the straight forward nature of the MDP problem which has only two possible optimal paths cutting right across the diagonal, exploitation is preferred to exploration and thus ε=0.1 is the best choice. The last 2 remaining plots evaluate the impact of γ. Since no substantial inferences can be drawn because the state space is still not big enough, going with the paper [1], I can only recommend a high value of γ for now.

5. Conclusion

Value Iteration performs the best for both MDP problems. Although VI and PI function similarly, VI achieves the optimal policy is lesser. QL also performs commendably well in both cases requiring significantly less computational time. It also performs way better than VI and PI in the first few iterations where the total reward of VI and PI dips to very large negative values as compared to QL which keeps it relatively grounded w.r.t to true reward. However, QL fails to achieve max attainable reward even at high iterations because of its tendency to explore.

Considering that both the MDP problems were relatively simple, the ability to explore doesn't yield better results. It makes more sense to take an exploitative approach which clearly supports the fact that VI is the best choice for both algorithms. Again because of the simplicity of the algorithm, analysis of discount factor, learning rate and epsilon didn't yield any further insights. Therefore, it is recommended to perform the experiments in this paper with different grid layouts, reward and cost functions and even different set of permissible actions. In addition, it makes sense to apply the experiments on much tougher problems like resource management in computer clusters, traffic light control, personalized recommendations, bidding and advertising etc.

6. References

[1] https://arxiv.org/pdf/1512.02011.pdf
[2] Code Referred: https://github.com/XuetingChen/cs7641-assignment-4.git