Markov Decision Processes

CS 7641 – Machine Learning

Manish Mehta

Introduction

The purpose of this project is to explore and implement some of the techniques from Reinforcement learning we have learnt to make decisions and/or allow the agent to act in the world more directly. We have to explore two Markov Decision Processes, one with small number of states and the other with large number of states. We then have to solve each of them using both Value Iteration and Policy iteration. Then, we can pick a Reinforcement learning algorithm and apply that to solve the two MDPs selected before. We then have to compare the performance to the earlier case when we knew the model, rewards, etc. For the purpose of this assignment, Q-learning was selected as the RL algorithm of choice.

Problem Description

The Markov Decision Processes (MDPs) can be thought of as instances of grid world problems, which can in turn be thought of as simplest and one of the first games we must have played as a kid. As a kid, if we have one board with a grid in it, and someone tells us a starting point, and offers to hand us a chocolate every time we reach the end, without getting stuck or banging into the obstacles, that itself is a MDP. The grid world can actually be considered as one of the classic MDP problems because it can be thought of as a version of real world problems. We as kids were the agents who needed to traverse the grid squares, one at a time, and avoid any obstacles and reach the goal as early as possible. For the purpose of this assignment, we have considered similar MDP problems where the start state is at bottom left and the goal is at top right.

As mentioned in the assignment description, we have taken two MDPs, one easy, which is a 5x5 grid having 25 states in total (including start, end, obstacles) and one hard, which is a 23x23 grid having 529 states in total. The agent has to start from the bottom left and reach top right where the reward for getting there is 100 (same as chocolate for the kid). Everywhere else, the reward is -1, so that the agent is forced to quickly get to the goal (as low number of steps as possible, using the reward function). Also, we have transition matrix defined to indicate the probability with which the agent moves in a direction. We have that set as 0.8 for the agent moving in the correct (towards the target) direction, and for all the other 3 directions, it is $0.2/6 = 0.0667$.

The problems that we have selected here are quite interesting because they kind of model some of the real world scenarios. For example (given my background was in Uber), we are considering autonomous vehicles trying to reach from one place (start) to another (goal). In a well-developed city layout, where the streets form grids, an interesting problem to think about is how to make the autonomous vehicle/robots reach the destination the fastest, with obstacles like walls, buildings, etc. Another use is Google maps, where we are provided with the fastest route based on the start and end points. Thus, it will be fun to explore the grid world version of such real-life problems. The problems selected look like this:

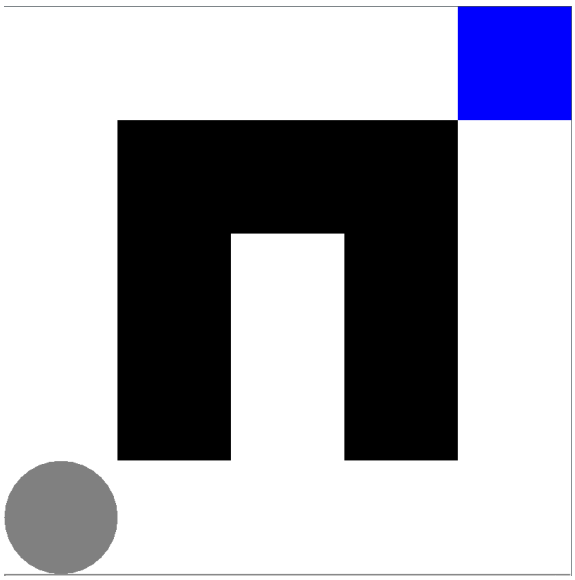
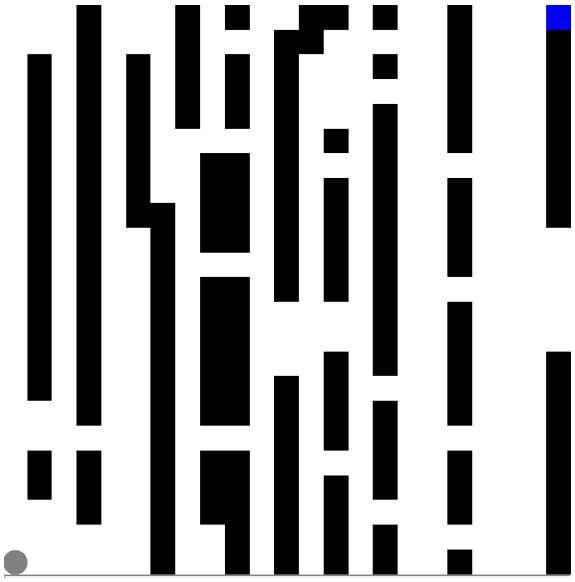


Fig 1 (a) Easy Grid World



(b) Hard Grid World

Description of Algorithms, Experiment Methodology and Analysis

The three algorithms that are being used to solve the MDP are defined below:

1. **Value Iteration:** The optimal policy gets computed by updating iteratively the expected utilities present at each state based on an action that maximizes it. This is done using the expected utilities, transition probabilities and rewards of neighbor states. Random initialization of initial values. Bellman equation is used for evaluation of states until convergence is achieved.

2. **Policy Iteration:** The agent usually cares about the optimal policy. In case of value iteration, the value function keeps increasing until an optimal is achieved, even after the policy already has converged. Policy iteration computes the utility values for the new policy, and converges when there is no improvement. It converges in lesser iterations but takes more computational time.

3. Q-Learning: This is one of the Reinforcement Learning models that does not presume anything beforehand about the rewards and probabilities for transition. Value and Policy iteration use domain knowledge but Q-learning is model free. The agent in this case only has information about the possible set of states and possible set of actions, and it can learn through experience with environment. The initialization is via assignment of Q-values to the states. After a new state is visited by the agent, an updated Q-value is assigned using immediate and delayed rewards. There are chances that this algorithm selected the policy randomly, but over time, Q-values are seen converge to converge to the true values for each state. We used epsilon greedy policy where each action is taken randomly is taken with some probability of epsilon. With time, it will converge, and epsilon can be decreased. This is allows more exploring of states.

The algorithms were implemented using BURLAP library and written in Jython. Basically the following were performed:

1. Value Iteration and Policy Iteration for solving the MDP
2. Q-Learning to solve the MDP
3. Compare and contrast the results

Using various relevant plots we will determine various metrics such as number of iterations to converge, which converged faster, which performed better, etc. Overall, the algorithms were run for 100 iterations in case of value and

policy iterations and for q-learning the cutoff was kept at 1000 iterations. After some exploring and hyperparameter tuning, the selected values of learning rate and epsilon were selected to be 0.1 and 0.7 (shown in Q-learning section). The learning rate usually indicates the importance of present rewards in comparison with future rewards while epsilon determines whether the action gets selected greedily or randomly and with what probability.

Value and Policy Iterations:

Easy Grid World:

Let us observe the states of the grid world after the first and 100th iteration, both in case of value and policy iterations, for different values of gamma ranging from [0.99, 0.90, 0.7, 0.4].

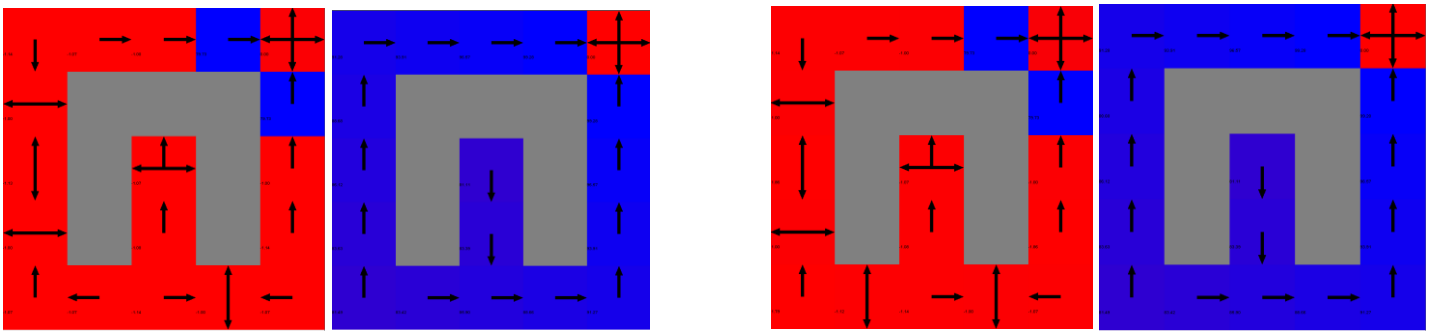


Fig 2. Value Iteration, gamma = 0.99, Iteration 1 and 100 Policy Iteration, gamma = 0.99, Iteration 1 and 100

These plots for other gamma values will not be repeated for the sake of reducing redundancy and more compactness. The final convergence states for each of the gamma values for Value and Policy iterations is shown below (In the above figure red represents start of the state and blue represents converged state):

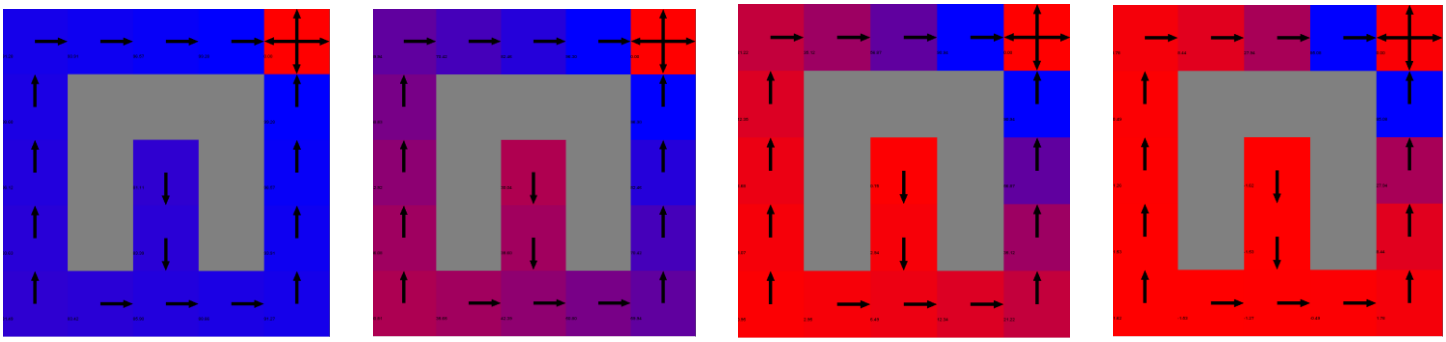


Fig 3: Value Iteration converged states (after 100 iterations) for gamma 0.99, 0.9, 0.7 and 0.4 respectively

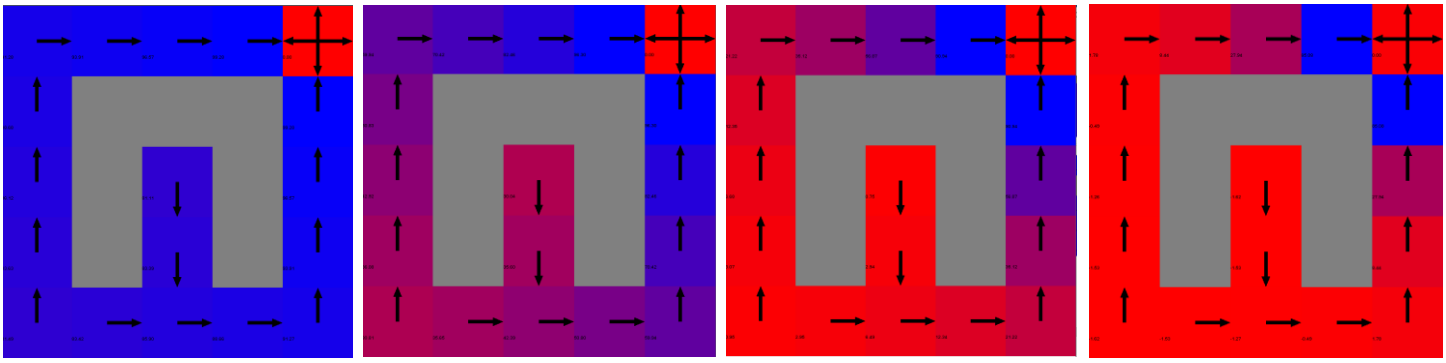


Fig 4: Policy Iteration converged states (after 100 iterations) for gamma 0.99, 0.9, 0.7 and 0.4 respectively

Here we can see the policy evolution in both Value and Policy iteration schemes, and we can see that for higher values of gamma, we attain the optimal policies (all states are blue, that is optimal), while for the lower value of gamma, the value function is still lower than optimal for the same number of iterations.

The plots for Rewards, Convergence and Number of Steps vs Iterations for both Value and Policy iterations is shown

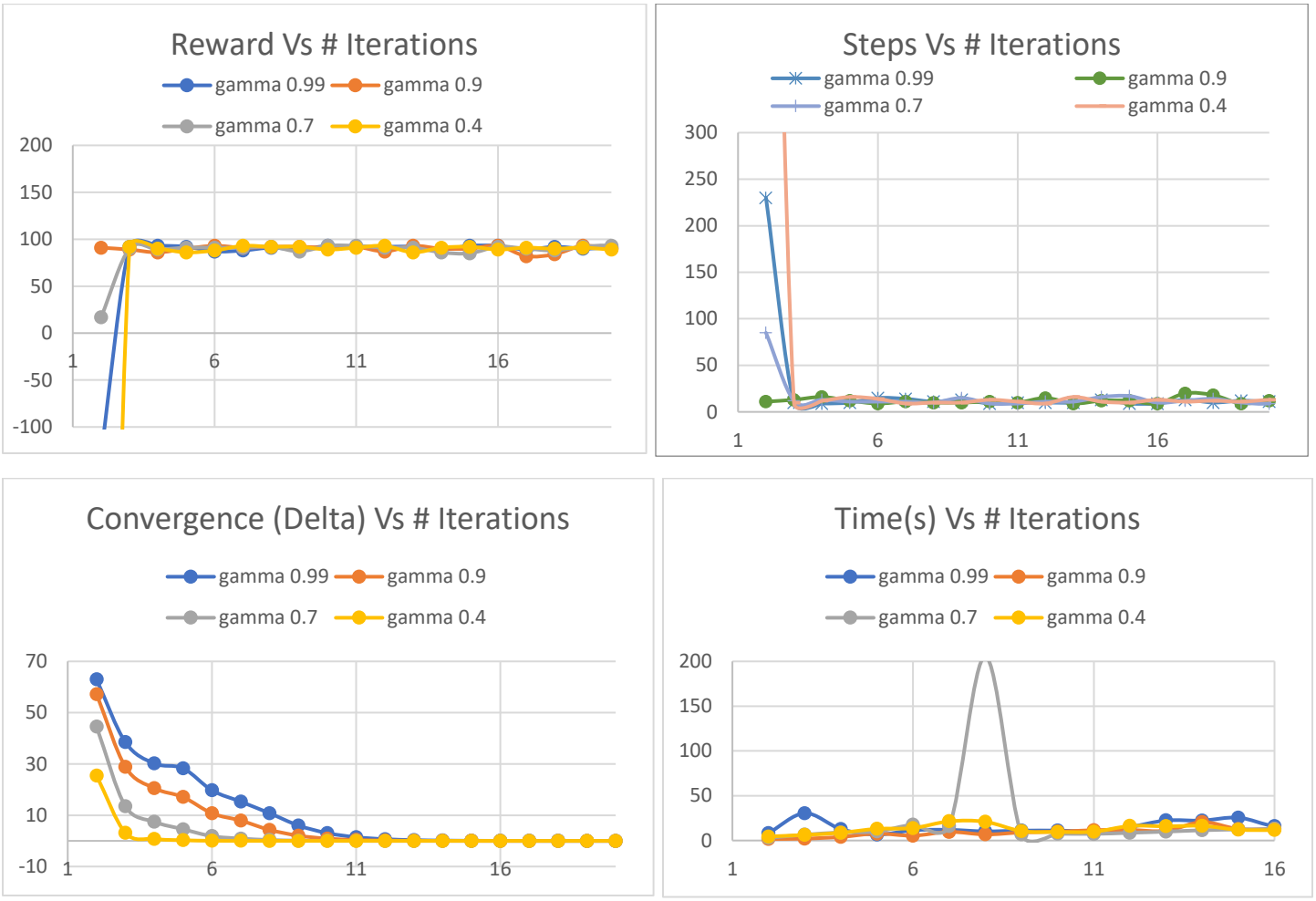


Fig 5: Value Iteration Plots

Similar plots for Policy Iteration are shown (only rewards and convergence are shown for sake of simplicity):

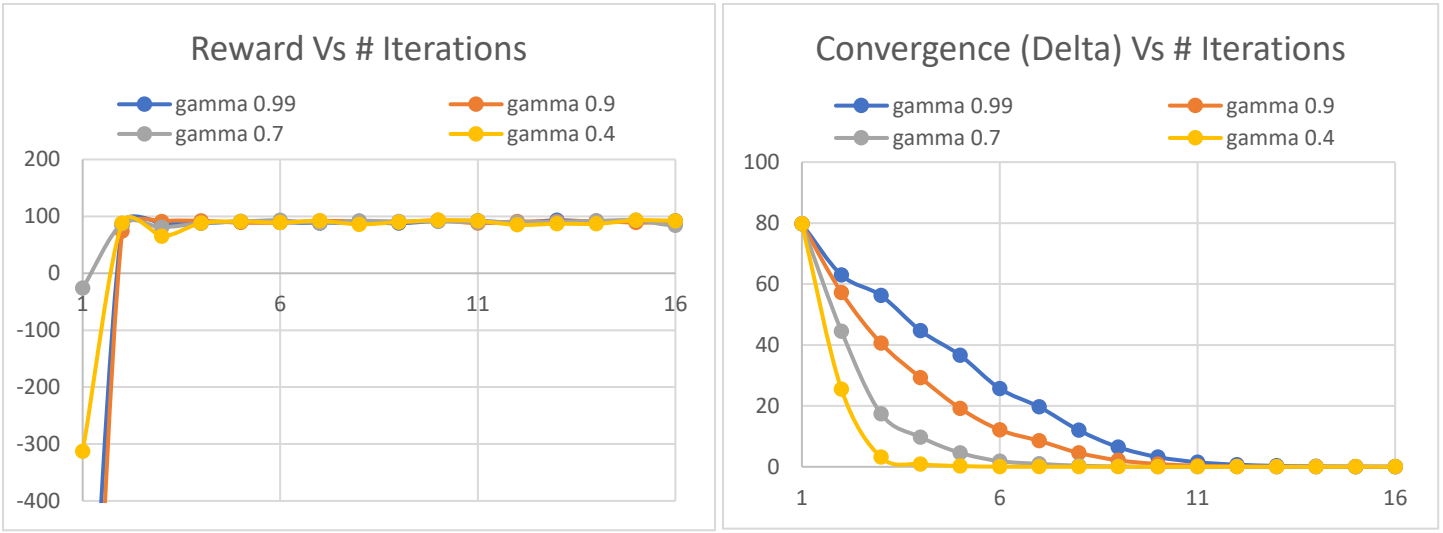


Fig 6: Policy Iteration Plots

As we can see from the plots of both Value and Policy iterations, with decrease in the gamma value, the optimal reward is attained slower than when it is higher. As we can see, the number of iterations it took to converge in Value iteration for gamma=0.99 is ~3, while it takes slightly higher number of iterations to converge for

gamma=0.4. Also, number of iterations taken for Policy iteration scheme to converge at gamma 0.99 is ~4. Both the value and Policy iteration schemes converge to the same states eventually. But, another interesting point to observe is the convergence plot, which shows that for higher values of gamma, the convergence below a certain threshold happened in larger number of iterations than the case with gamma = 0.4. This is intuitive and was observable from the Figures 3 and 4, where the system is giving higher importance to the past states, as a result of which the value function becomes faster soon, and the system now has no reward for speeding up its movement towards the target/goal. On the other hand, for smaller gamma values, the discounting happens less, the system remembers the past state less, and it is forced to rush towards the final goal/target, without achieving the optimal value of the value function. Thus, there is a tradeoff, in both Value and Policy iterations. The optimal reward obtained is ~93, which is obtained quicker for higher gamma. Time of iterations keep increasing with number of iterations. Once we visualize the results of Q-learning too, we will compare and contrast the overall results. For the hard grid world, only more relevant and less redundant plots will be shown:

Hard Grid World:

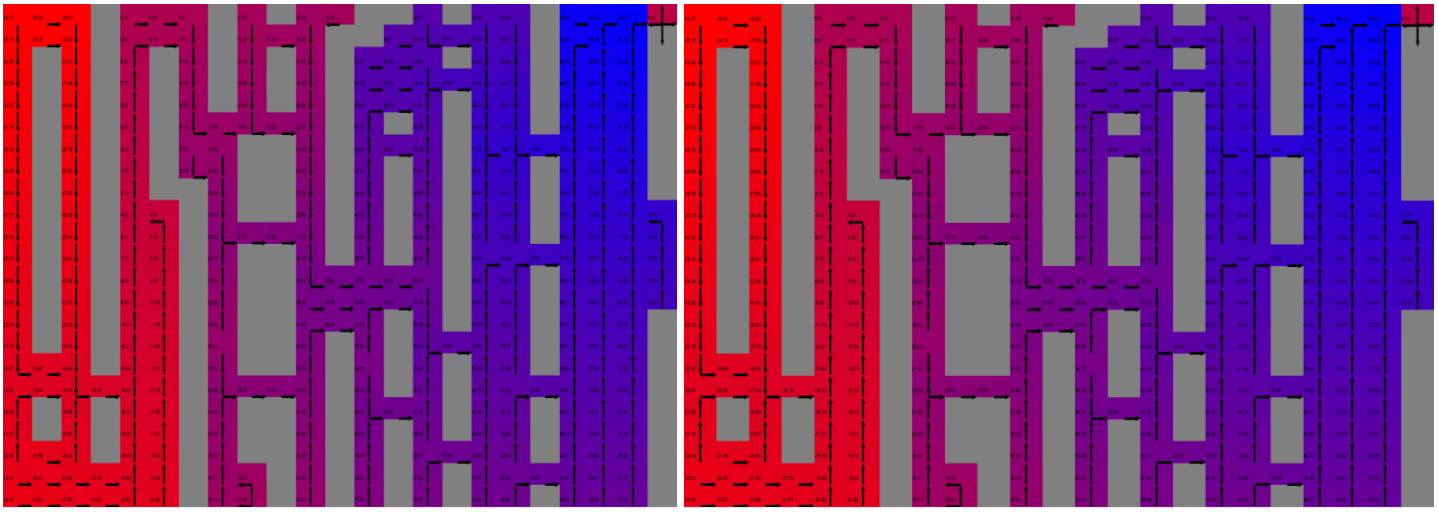
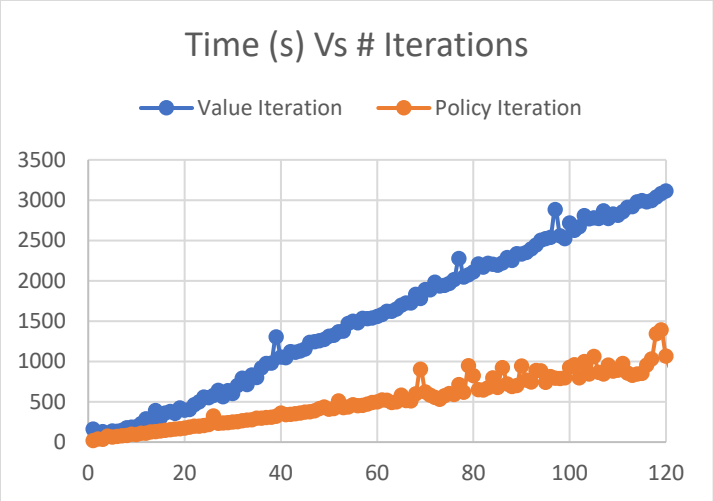
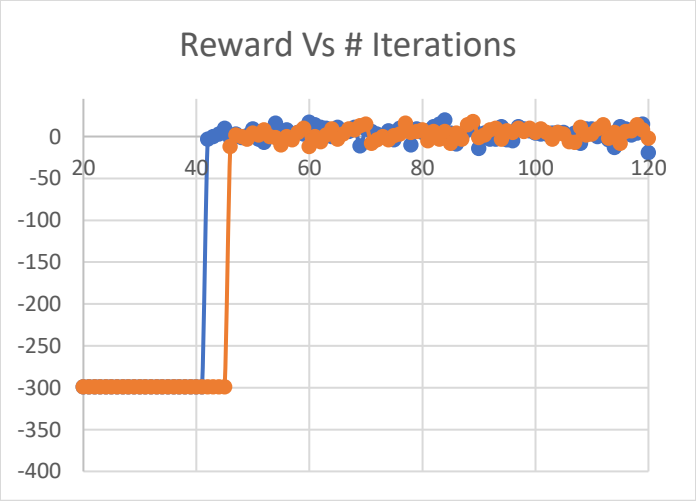
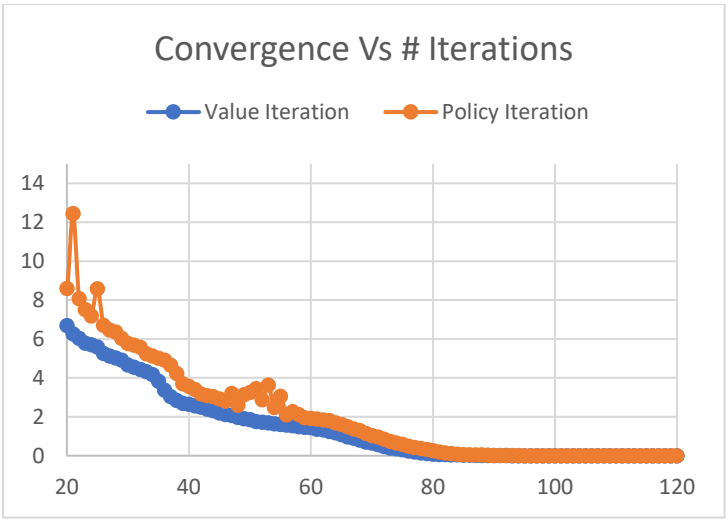


Fig 7: Value and Policy iteration plots for converged states of Hard Grid World problem

Again, they converge to the same states. The other plots for Rewards, Time, Steps and Convergence (Delta) are not shown because they followed similar behavior and analysis as before. But plots for comparison of Value and Policy iteration are shown:







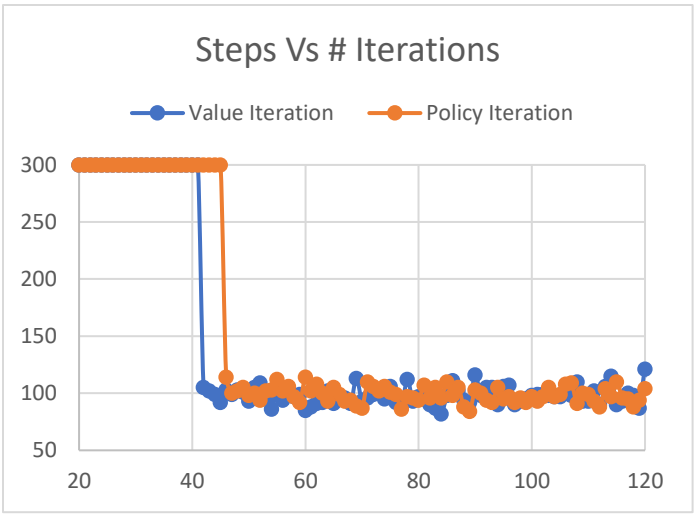
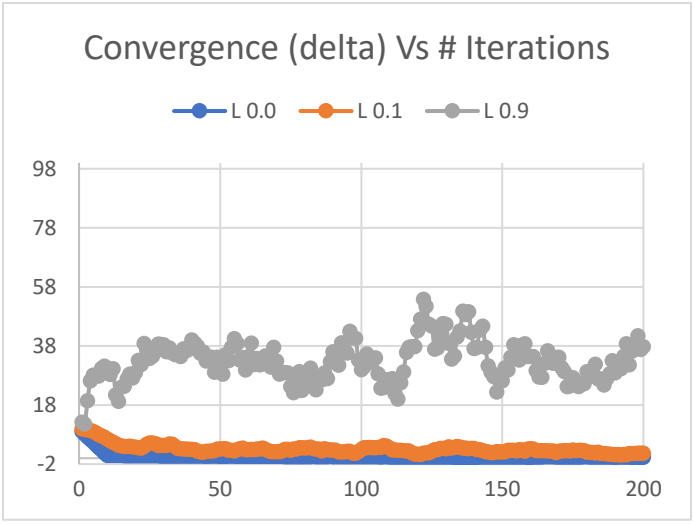


Fig 8: Plots for comparing b/w Value and Policy Iterations for Hard Grid World

Thus, we can see that for larger number of states, the Value iteration takes a lot of time with increase in number of iterations. Also, optimal reward is achieved in lesser number of iterations for Value as compared to Policy Iterations. It takes about 43 iterations for Value scheme to attain optimal value function, while it take about 50 iterations for policy scheme to do the same.

Q-Learning:

The algorithm was run for different value of learning rates and different value of epsilon. The plots for convergence (delta) vs number of iterations for various Learning rates and Epsilon values is shown:



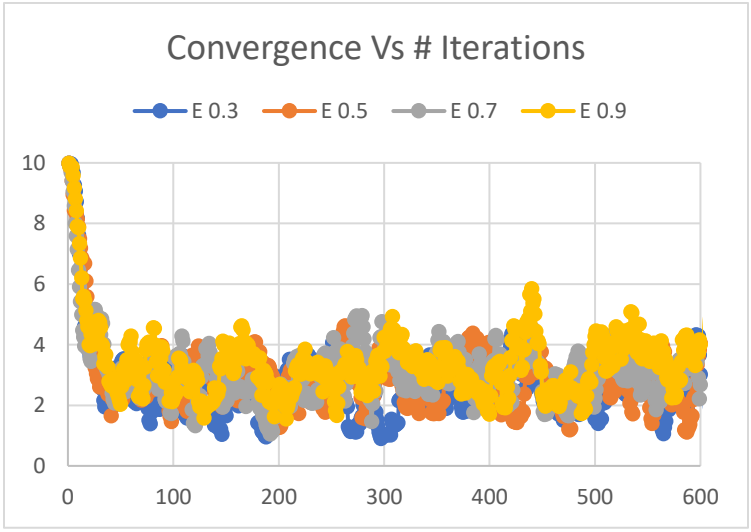


Fig 9: Plot of Convergence for different Learning Rates and Different Epsilon (Easy Grid World)

Also (not shown for the sake of readability of the report), the Reward was plotted for different learning rates and different epsilons. The best set of parameters was found to be L = 0.1 and epsilon = 0.7 since the Reward attained its optimal/highest value at that combination. For the Hard Grid World too, the plots had similar behavior, so optimal parameters remained the same. The grid world problem in both cases is shown:

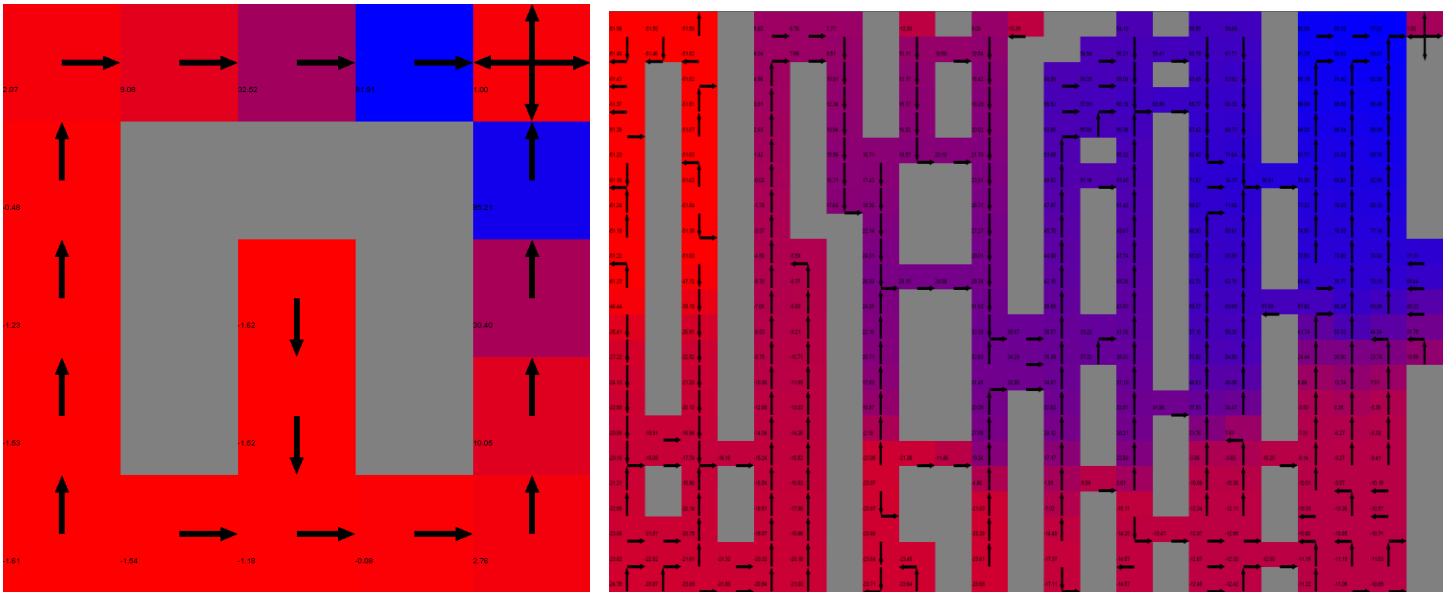
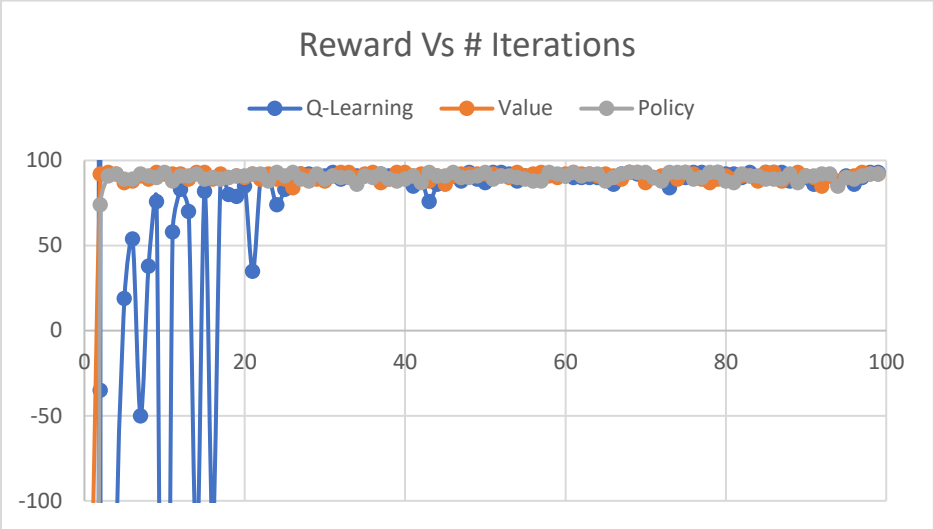
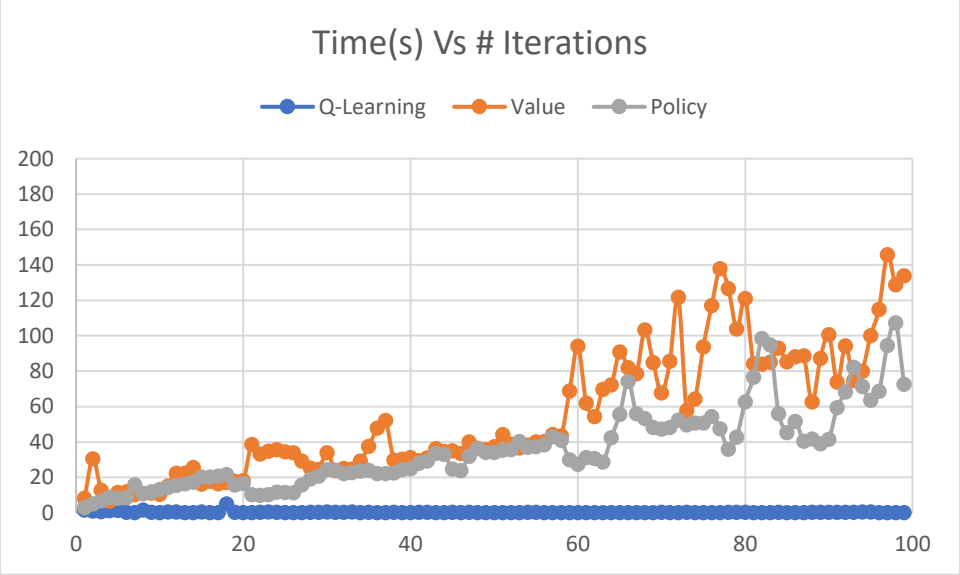


Fig 10: Easy and Hard Grid World problems final converged states

We can see that as compared to the Value and Policy iteration schemes, Q-Learning didn't reach the optimal value completely. The number of iterations taken for Value iteration are lesser than that by Policy Iteration than that by Q-learning. More analysis and conclusions are below.

The plots comparing all three algorithms for Easy Grid World and Hard Grid World are shown below:





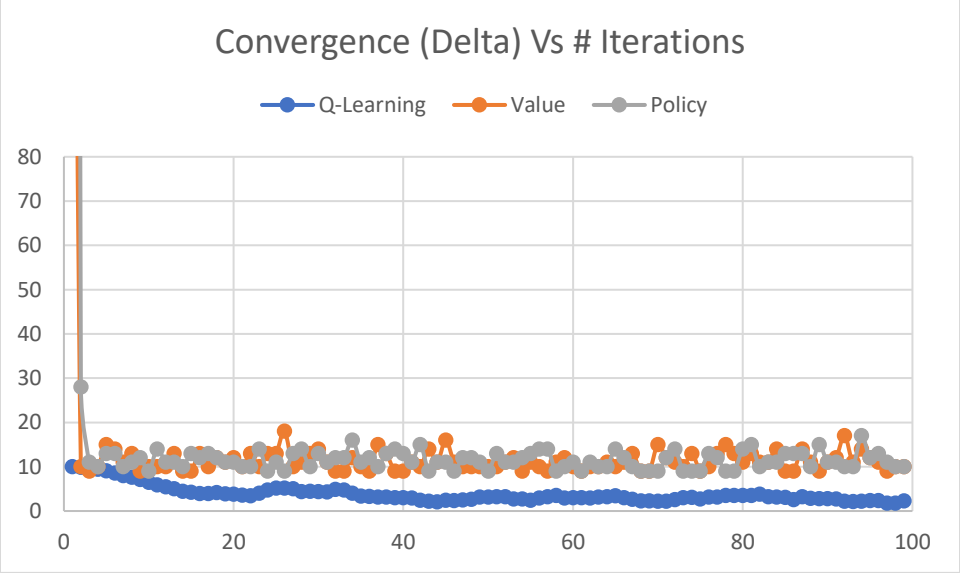
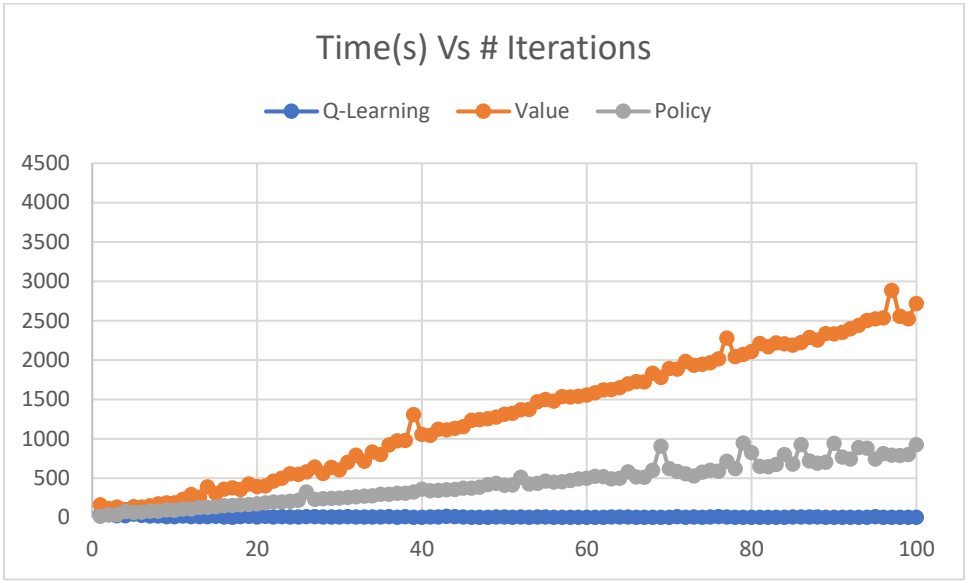
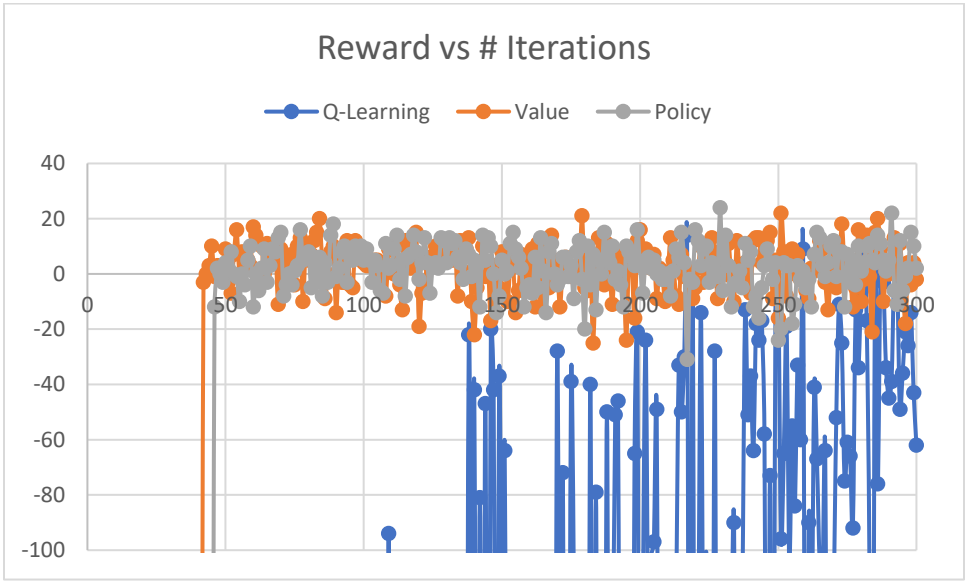


Fig 11: Plots for Each of the algorithms for Easy Grid World





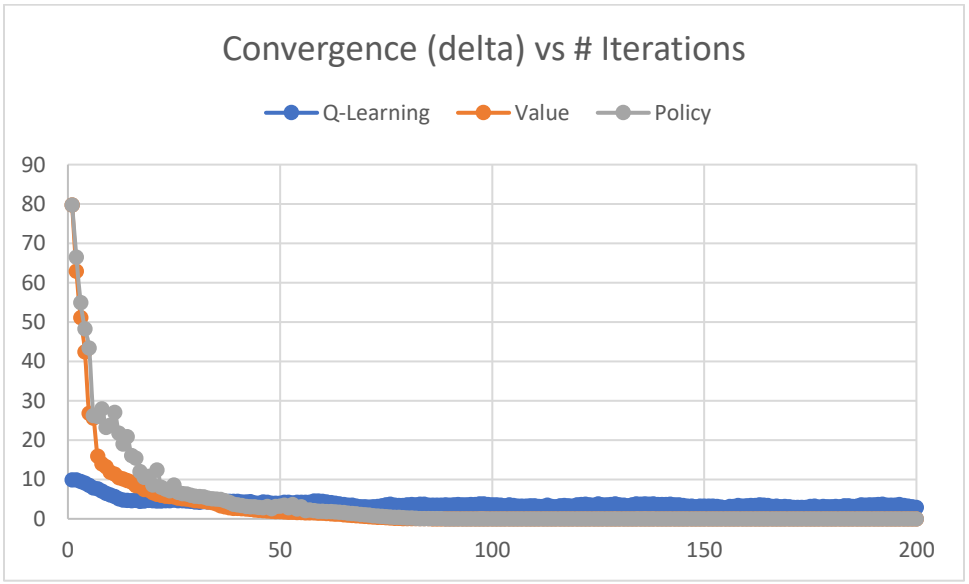


Fig 12: Plots for Hard Grid World

All the plots show that Q-learning takes less time per iteration, and that the time increases with increase in number of iterations for Value and Policy iteration schemes.

Conclusions

Both MDP show that the Value and Policy schemes converge faster than Q-learning and are faster in terms of achieving the optimal state. But, Q-learning runs at a constant time but the other 2 algorithms increase with the increase in number of iterations. With increase in number of states, the number of iterations for Q-learning increase manifold, but it does not require any knowledge of transition probabilities or any knowledge of the reward functions. Thus, we can easily conclude that for cases where we know the reward function and transition probabilities, Value and Policy reach optimal value better (converge in smaller number of iterations) as compared to the case where we don't know these and then Q-learning will perform way better.

References

[1] https://gatech.instructure.com/courses/32819/assignments/116546

[2] https://github.com/danielcy715/CS7641-Machine-Learning/tree/master/

[3] Simulation and Animation of Reinforcement Learning Algorithm, V. Mehta, R. Kelkar, A. Moore, CMU

[4] T. M. Mitchell, Machine Learning. McGraw Hill, 1997

[5] A. Moore, "Reinforcement learning: Tutorial slides." [Online]. Available: http://www-2.cs.cmu.edu/awm/tutorials/rl06.pdf

[6] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. London, England: The MIT Press, 1998.

[7] Brown-UMBC Reinforcement Learning and Planning (BURLAP) java code library: http://burlap.cs.brown.edu/