HW4: Markov Decision Processes

Sowmya Yellapragada (syellapragada3)

April 23, 2018

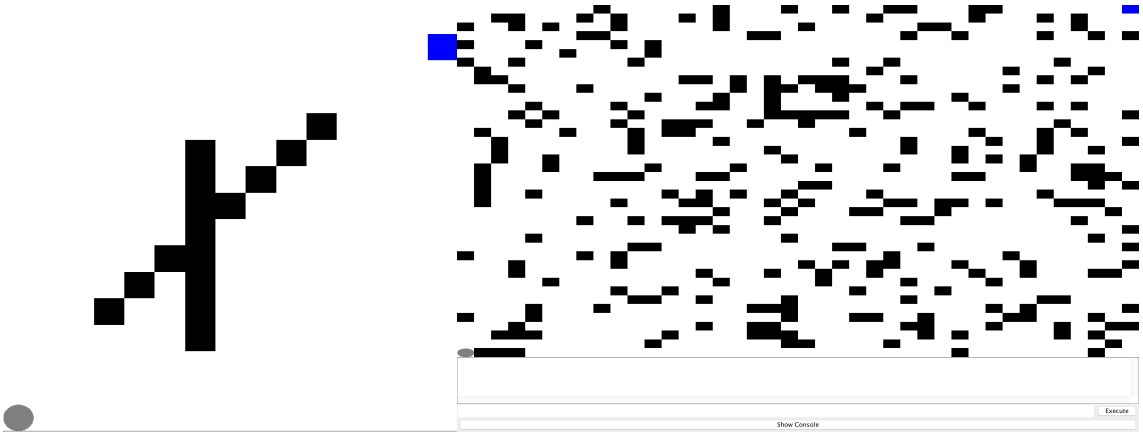
Abstract

In this paper we discuss the results of value iteration and policy iteration on two commonly known Markov decision process. We perform these methods on the grid world problem of two levels - easy and hard. The aim of this paper which algorithm works best for each of these problems, and how the performance of each of the algorithms varies between the two problems. We will also attempt to vary the discount rate and discuss any significant changes in performances. We use the burlap framework provided in this class for performing experiments and analysis.

1 Introduction

A Markov Decision Process is a discrete, time stochastic control method. They provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. In this report, the problem being modeled is that of autonomous mobile robots. More specifically, starting from an arbitrary placement in the grid, using purely localized computations, the robots must move so to reach in finite time a final state with maximum reward.

To model this problem, I have used the Grid World MDP. In it, an agent can be in one of many discrete physical states. The goal of the agent is to navigate the plane to reach the final destination by avoiding the walls. In order for the agent to navigate it has to perform 4 actions in any cardinal direction. Each action is associated with a specific cost. In the easy grid world, which is a 15x15 grid 1a, each state other than the final state incurs a cost of -1. In the hard version of the grid world, which is a 40x40 grid 1b, the cost associated for each cell has been set randomly with values -20, -10, 0, 10 or 20 before the MDP is initiated. The reward received at the terminal state is 100. The objective of the agent is to maximize the net reward. Once the agent chooses a particular direction to navigate in, he will successfully be able to go in that direction with a probability of 80%. There is a 20/3% chance of him navigating in the other three directions. The agent uses a discount rate to prioritize present rewards over future, by a certain factor.



(a) Easy Grid (b) Hard Grid

Figure 1: Grids

2 Why is the problem interesting

Grid world has many real life applications. Some of the famous ones are autonomous household cleaning robots, that are been widely used today. An extrapolation of grid world with moving walls can be a good training space for autonomous vehicle simulations as well. Additionally, the reason that the GridWorld problem is interesting is that it has a relative small problem space and mostly the planner and learner will converge within 50 iterations. Meanwhile, there are a lot of parameters we can tune for this problem in order to have a better understanding the impact of the model on the solutions. For example, we can have different number of walls scattered at random locations and have different stochastic success probability of each move. The flexibility on the parameters of the model gives us a better understanding the MDP problem as well as the planner and learner.

Examining the easy and hard grid worlds will be a good way for us to estimate the scalability of autonomous robots into the real world, beyond experimental setup. In this report, the hard grid world has been given an inherent randomness (in rewards), in order to replicate the chaotic real world systems.

3 Algorithms

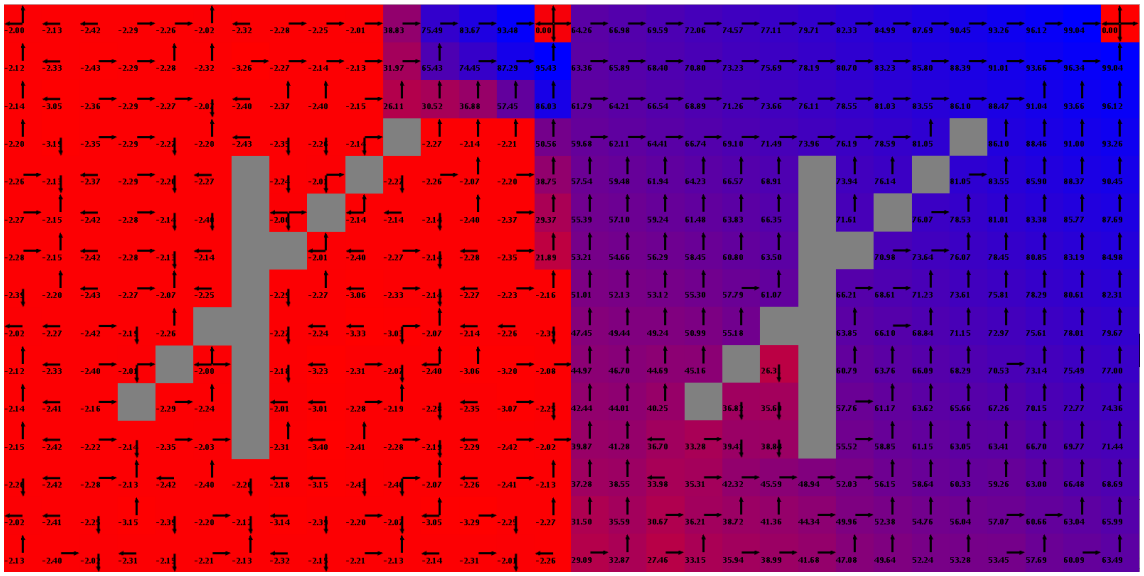
3.1 Value Iteration

Value iteration is a planning algorithm. The essential idea behind is to always choose the action that maximizes expected utility. The expected utility of a state is the immediate reward for that state, plus the expected discounted reward if the agent acted optimally from that point on:
$U(s) = R(s) + \gamma max_a \Sigma_{s'} T(s, a, s')U(s')$

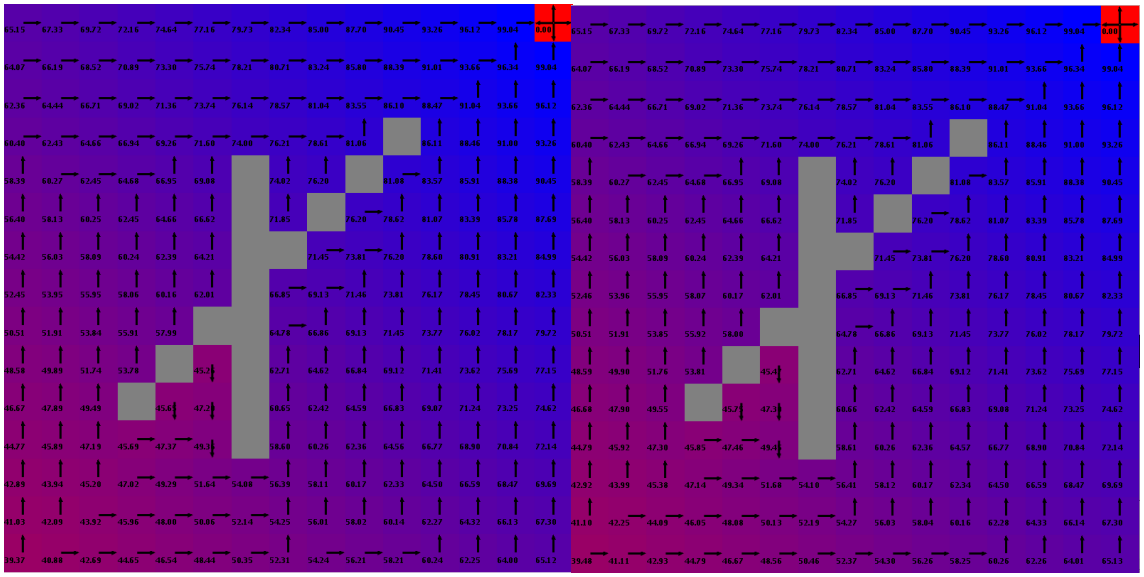
Note that the value for each state can potentially depend on all of its neighbors' values. If our state space is acyclic, we can use dynamic programming to solve this. Otherwise, we use value iteration algorithms, as follows:
1. Assign each state a random value
2. For each state, calculate its new U based on its neighbor's utilities.
3. Update each state's U based on the calculation above: $U_{t+1}(s) = R(s)+\gamma max_a \Sigma s' T(s, a, s')U_t(s')$
4. if the updated values don't different from that of the previous by more than δ(max delta), stop.
This algorithm is guaranteed to converge to the optimal (within δ) solutions but this takes a long time to converge in specific situations.

3.1.1 Easy Grid

We vary iteration size and observe how the actions and utility values in the grid vary. As can be noticed clearly from the 2a, for low iteration value, there is no prospect for many states to get a positive utility as the actions they take will not lead the agent to the final state(reward of 100). It is also interesting to note that the states that are surrounded by walls, show action policy towards the wall, as choosing away from the wall is not yielding a positive utility in the few iterations that it is allowed. As the iterations are increased, as can be seen in 2b the actions in these states are seen to be moving away from the wall and towards a path that would lead to the highest rewards to the terminal state. Further increasing the iteration size, we observe that the utility values of each state attains remains constant within a delta range, as can be clearly observed this convergence should have be achieved at iteration size greater 15 but less than 25. This can also be observed from 3a. The rewards 3b and steps 3c to the terminal state remain fairly constant even at lower values at of iteration size. However, convergence is achieved later. This can be attributed to the stochastic nature of our action sequences.

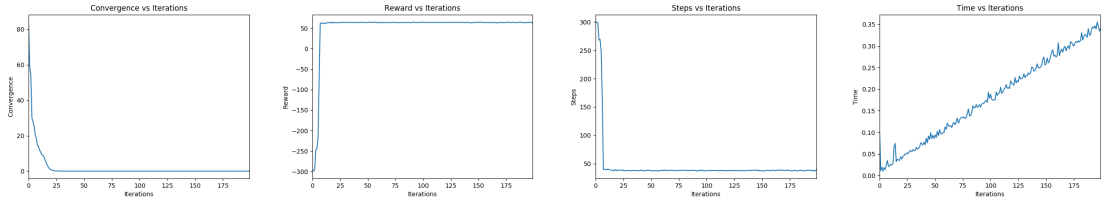


(a) Iteration size = 2 (b) Iteration size = 15



(c) Iteration size = 25 (d) Iteration size = 100

Figure 2: Value Iterations - Easy Grid



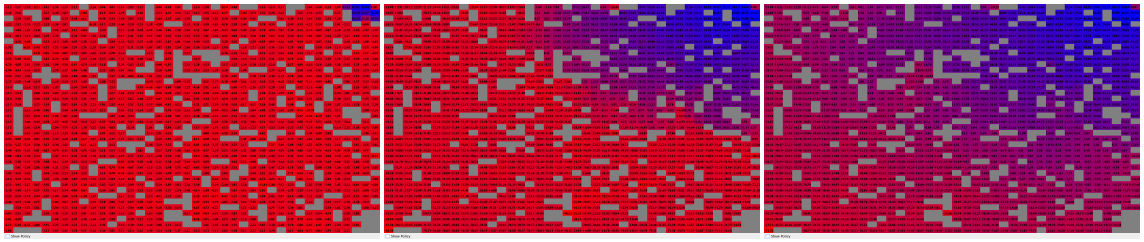
(a) Value Convergence Plot (b) Value Rewards Plot (c) Value Steps Plot (d) Value Time Plot

Figure 3

3.1.2 Hard Grid

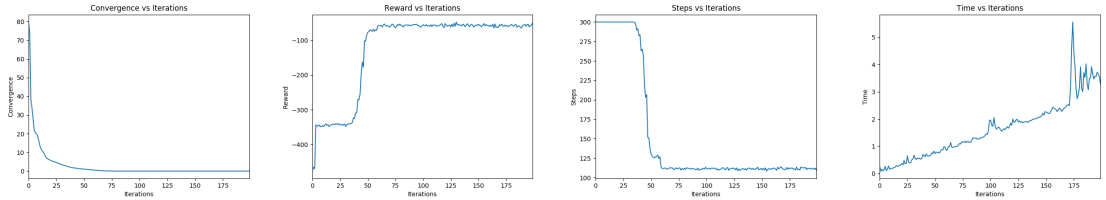
As can be noticed from 5a is much more gradual than that of the easy grid. Convergence in the hard grid appears to be occurs close to an iteration size of 75 from this graph. This seems

consequential, since the hard grid has a a much larger size than that of the easy grid. The rewards in the hard grid are not as uniformly -1, as discussed before, the rewards in this grids are -20, -10, 0, 10 and 20. Hence the gradation in color for iteration size=100 can be observed. Even though a convergence has been achieved around the iteration size of 100, not every state has a positive utility value. The rewards plot for the hard grid is quite interesting as well. Unlike in the case of easy grid, the reward did not achieve a instant jump in the rewards. It attained two maxima. This can be attributed to the large grid size. 5c exhibits a similar nature. Hence it can be concluded that the whole grid hasn't been explored until we have increased the iteration size to at least 60.



(a) Iteration size = 2 (b) Iteration size = 20 (c) Iteration size = 100

Figure 4: Value Iterations - Hard Grid



(a) Value Convergence Plot (b) Value Rewards Plot (c) Value Steps Plot (d) Value Time Plot

Figure 5

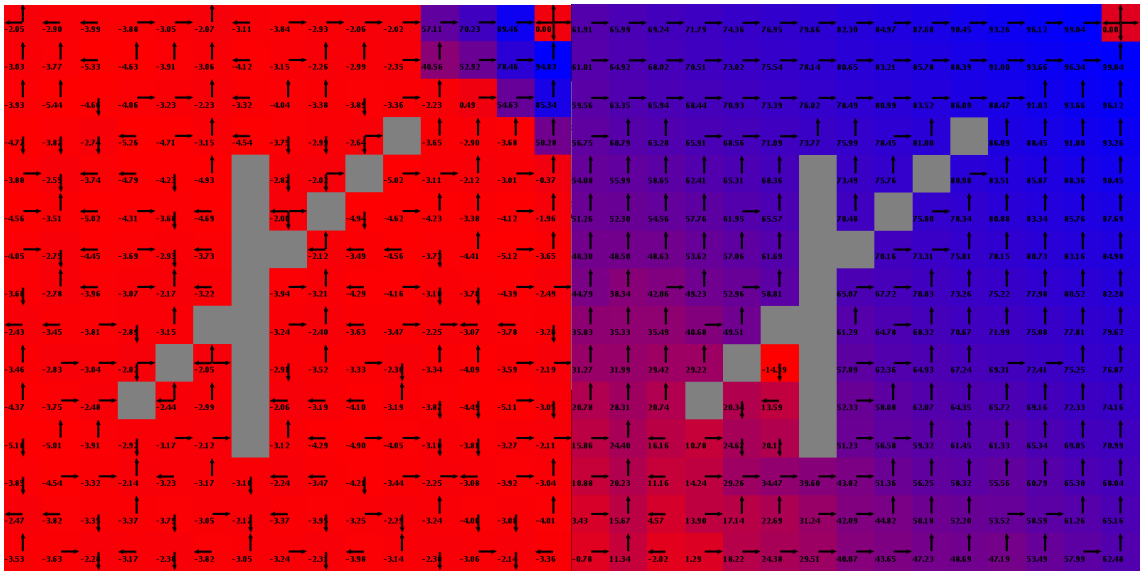
3.2 Policy Iteration

Unlike value iteration, policy iteration aims at the convergence of policy (actions) rather than the utility value. We assume an initial policy and computer utility and thereby policy values iteratively. This computation doesn't require you to explore all actions while computing utility value of a state. Max over all possible actions made the equation non-linear in the case of value iteration. Eliminating max, makes the equation linear. Thereby easier to solve/compute. The policy iteration algorithm is as follows:

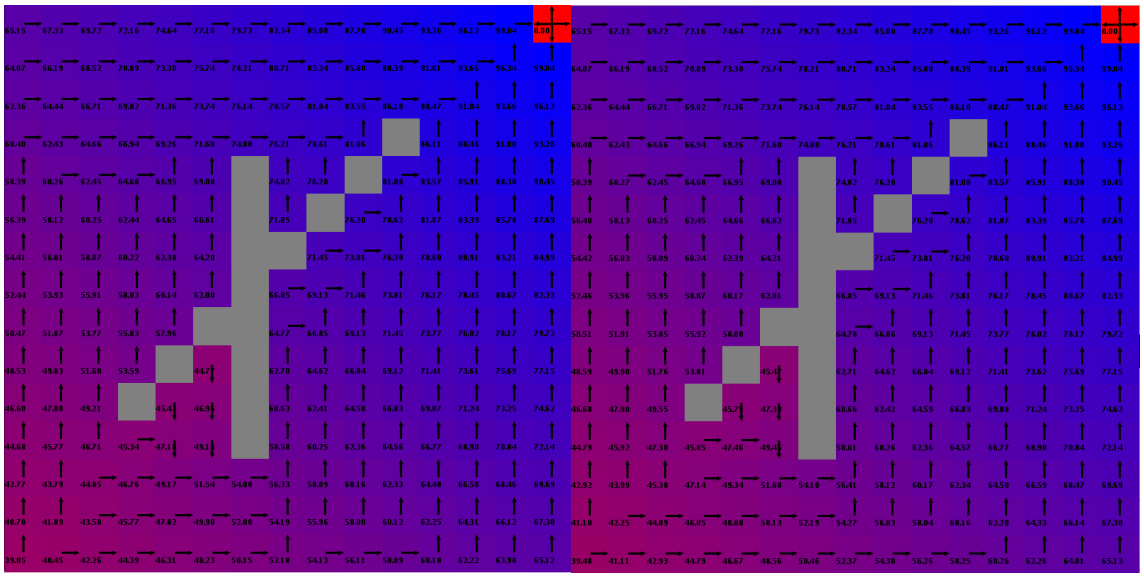
1. Guess an initial policy π_0
2. Evaluate : Given π_t calculate $U_t = U^{\pi_t}$
3. Improve : $\pi_{t+1} = argmax_a \Sigma T(s, a, s')U_t(s')$

3.2.1 Easy Grid

A similar convergence can be observed as in the case of value iteration, with an iteration size just below 25. The trends in rewards 7b and steps 7c too are quite similar to those observed in case of value iteration. Notice that although we have mentioned a convergence value of around 25, there are a few action the differ in 6c and 6d indicating that we are observing convergence under a delta range. Exact convergence is time taking to attain. Convergence within a delta range is a quite useful technique, as it doesn't give rise to much variation in the results. If we increase our delta tolerance a little more, we could observe a convergence even before the value iteration

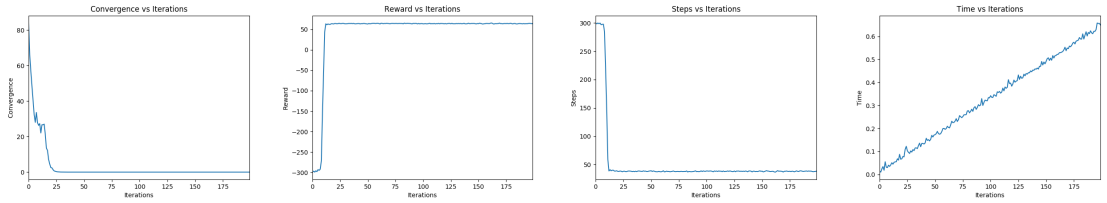


(a) Iteration size = 2 (b) Iteration size = 15



(c) Iteration size = 25 (d) Iteration size = 100

Figure 6: Policy Iterations - Easy Grid



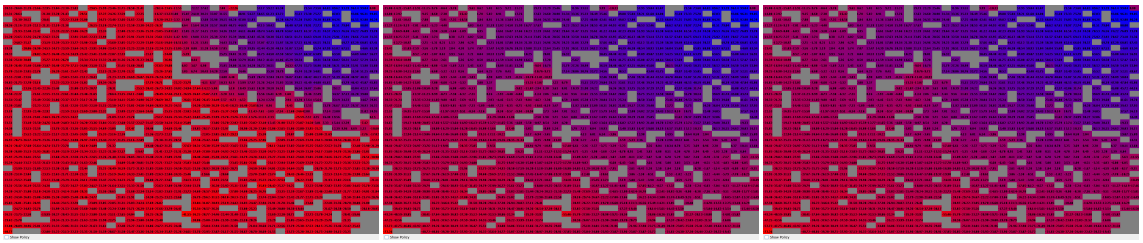
(a) Policy Convergence Plot (b) Policy Rewards Plot (c) Policy Steps Plot (d) Policy Time Plot

Figure 7

3.2.2 Hard Grid

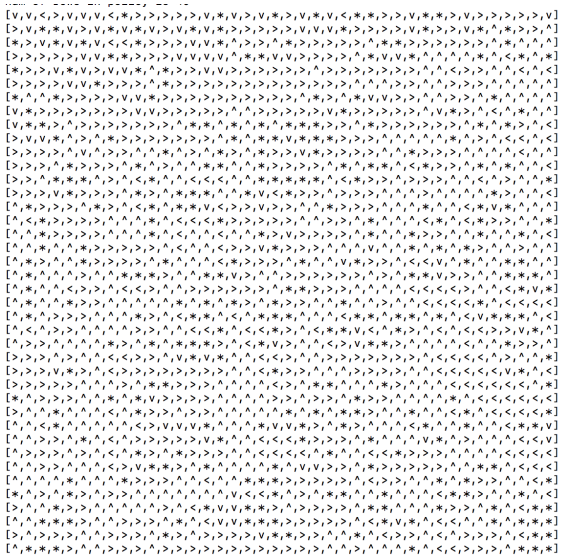
Policy iteration on the hard grid exhibits a similar behavior as the value iteration. Convergence is observed after 50 and below 75 iterations. Similar multi peak trend is observed in policy rewards

and steps plot, as was seen in the value iteration

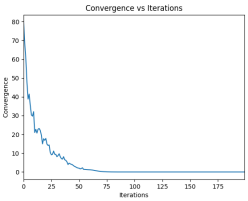


(a) Iteration size = 2 (b) Iteration size = 20 (c) Iteration size = 100

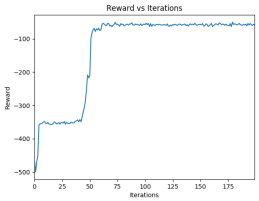
Figure 8: Policy Iterations - Hard Grid



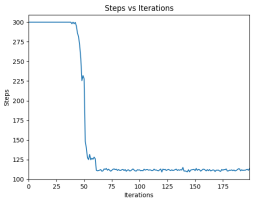
(a) Optimal policy



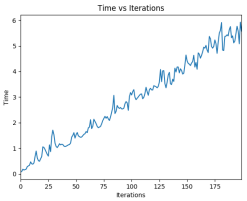
(b) Policy Convergence Plot



(c) Policy Rewards Plot



(d) Policy Steps Plot

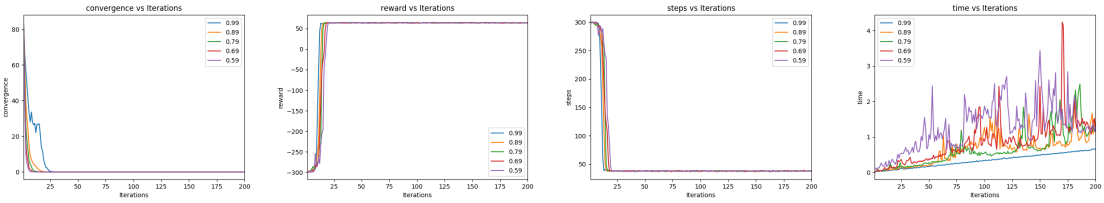


(e) Policy Time Plot

Figure 9

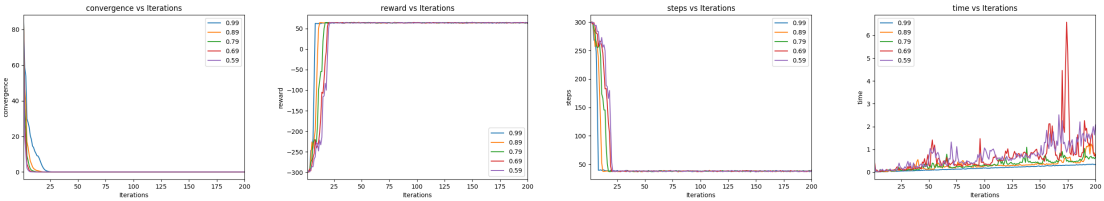
3.2.3 Varying discount rate

In this section we vary the discount rate and re-run the policy and value iterations for both easy and hard grid. For easy grid's policy iteration, we can observe that convergence for lower values of discount rate is much more rapid against the iteration size. However, the time taken for lower discount rates is much higher that our baseline time at 0.99 discount rate. Lower discount rates imply that the future rewards are given less importance. Hence, the number of steps for an iteration size to converge is high, as low discount rate restricts its view to all the future utilities. Similar trends can be observed in the value iteration of the easy grid.



(a) Convergence Plot (b) Rewards Plot (c) Steps Plot (d) Time Plot

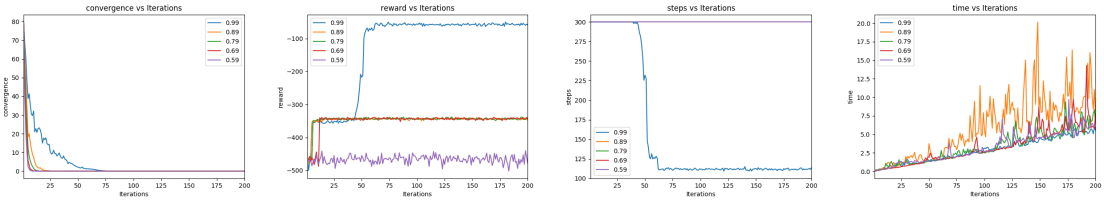
Figure 10: Easy Grid - Policy



(a) Convergence Plot (b) Rewards Plot (c) Steps Plot (d) Time Plot

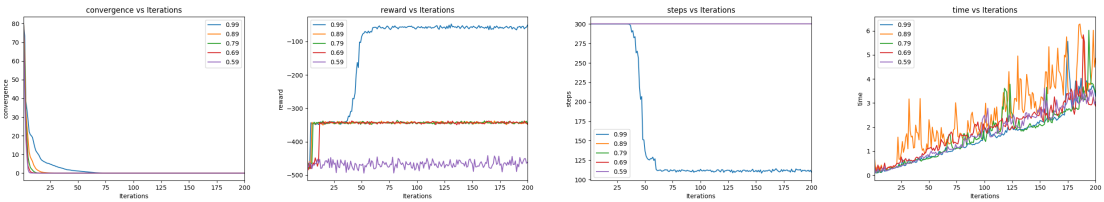
Figure 11: Easy Grid - Value

In the policy iteration of the hard grid, interestingly, the jump in the reward around iteration size 50 is not observed for lower discount rates. Bringing to light the inability of the system to see the high rewards in the long term, even when large number of iterations are allowed. The number of steps too are sufficiently large, across the iteration sizes. This is because, the system is limited from seeing the high reward in the termination state, and so is unable to converge for any iteration size. Similar trends are observed in the case of value iteration on hard grid



(a) Convergence Plot (b) Rewards Plot (c) Steps Plot (d) Time Plot

Figure 12: Hard Grid - Policy



(a) Convergence Plot (b) Rewards Plot (c) Steps Plot (d) Time Plot

Figure 13: Hard Grid - Value

3.3 Q-Learning

Q-learning is a reinforcement learning technique used in machine learning. The technique does not require a model of the environment, i.e., does not know about or use models of the transitions T or the rewards R. Instead Q-learning builds a table of utility values as the agent interacts with the world. These Q-values can be used at each step to select the best action based on what it has learned so far

For any finite Markov decision process (MDP), Q-learning eventually finds an optimal policy, in the sense that the expected value of the total reward return over all successive steps, starting from the current state, is the maximum achievable. Q-learning can identify an optimal action-selection policy for any given finite MDP

Q represents the value of taking action a in state s and there's two components to that:

- The two components are the immediate reward you get for taking action a in state s plus the discounted reward which is the reward you get for future actions
- An important thing to know is that Q is not greedy in the sense that it just represents the reward you get for acting now, it also represents the reward you get for acting in the future
- Q[s,a] = immediate reward + discounted, i.e,
  $Q'[s,a] = (1-\alpha)Q[s,a] + \alpha(r + \gamma Q[s', argmax_{a'}(Q[s',a'])])$

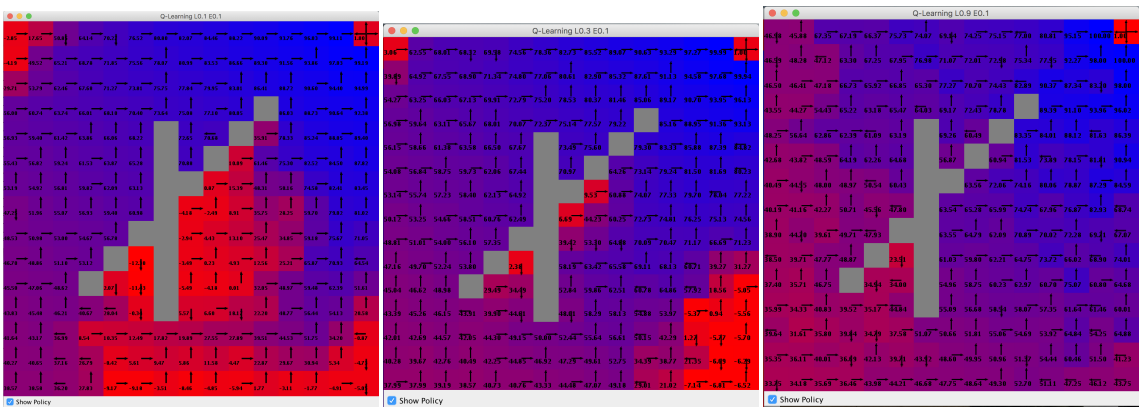
Once Q has been computed, policy at a state s is simply given as - $\pi(s) = argmax_a(Q[s,a])$ with a probability of $1-\epsilon$ or random action with a probability of ϵ

Here, α is the learning rate and ϵ dictates how much we believe our predictions. Low ϵ implies that we believe our predictions and don't allow for any random actions. This might lead us to the problem of getting stuck at a local minima. Initially we set a high ϵ value so that we don't overly believe our initial data. We decay this value, therefore probability that the new value will be chosen increase. The dilemma is that whether we should chose the new value because we want to explore and exploit at the same time. So its always a trade-off. These are hyper parameters to our algorithm. We will vary these hyper parameters and apply Q learning to both easy and hard grids and compare the results.

The α, ϵ are varied as follows - (0.1, 0.1), (0.1, 0.3), (0.1, 0.5), (0.3, 0.1), (0.3, 0.3), (0.3, 0.5), (0.5, 0.1), (0.5, 0.3), (0.5, 0.5), (0.9, 0.1), (0.9, 0.3), (0.9, 0.5)

3.3.1 Easy Grid

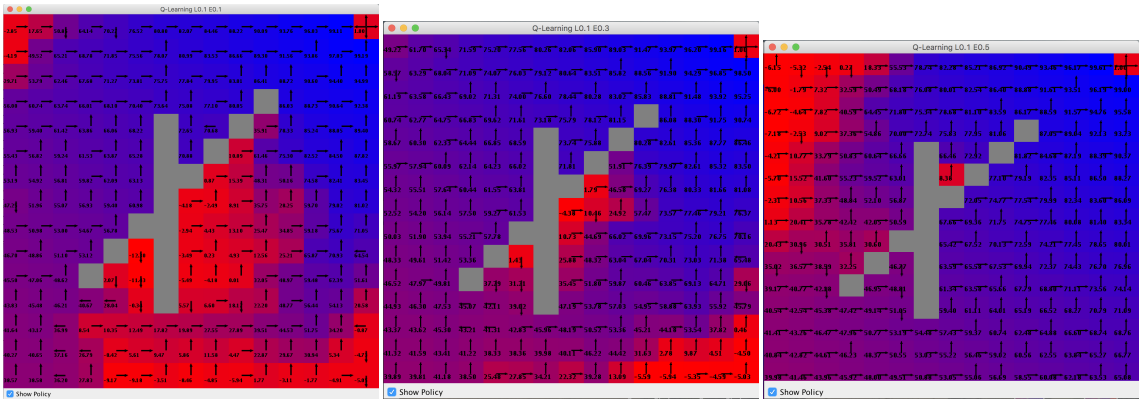
As can be observed in Fig 10 as the α value increases, the learning in the grid has gradually increased. Initially the grid has all negative cells (red cells) as the α increases, the grid learns and takes actions that give high Q values.



(a) $\alpha = 0.1, \epsilon = 0.1$ (b) $\alpha = 0.3, \epsilon = 0.1$ (c) $\alpha = 0.9, \epsilon = 0.1$

Figure 14: Varying α

Further, we will now vary the ϵ and observe how the randomness increases as ϵ increases. This trend can be observed in Fig 11



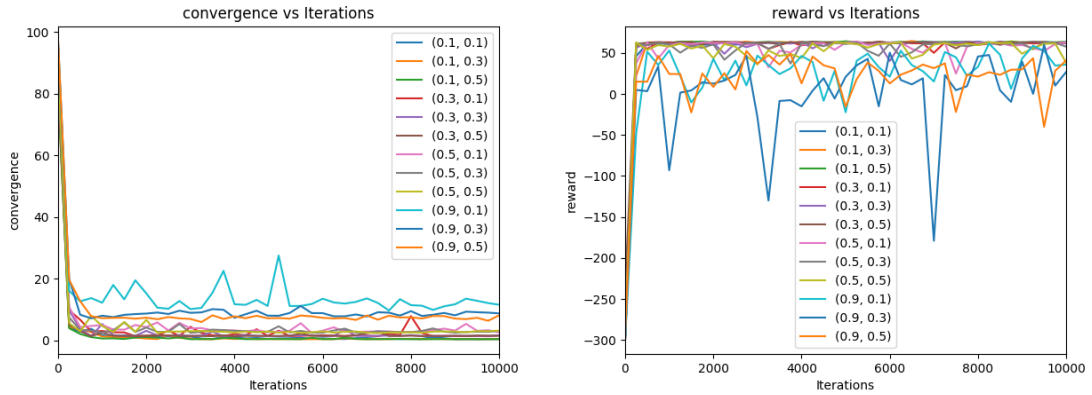
(d) $\alpha = 0.1, \epsilon = 0.1$ (e) $\alpha = 0.1, \epsilon = 0.3$ (f) $\alpha = 0.1, \epsilon = 0.5$

Figure 15: Varying ϵ

Further, we plot convergence, rewards, steps, time plots for the Q learning algorithms with different α, ϵ values across different iteration size, as can be seen in Figure 12. We observe that results with low α values tend to converge better, attain higher rewards in fewer steps Figure 12 g, h, i . However from the time comparison graph Figure 12 j we notice that higher α values perform better. This would seem intuitive as low α value implies very slow learning. Hence to strike a balance, our selection for α should be 0.3 or 0.5 and not 0.1 (which can be concluded from the convergence and rewards graphs). The choice ϵ is dependent on how much we are willing to explore and exploit. We can choose a value of 0.5 to strike a balance between exploitation and exploration.

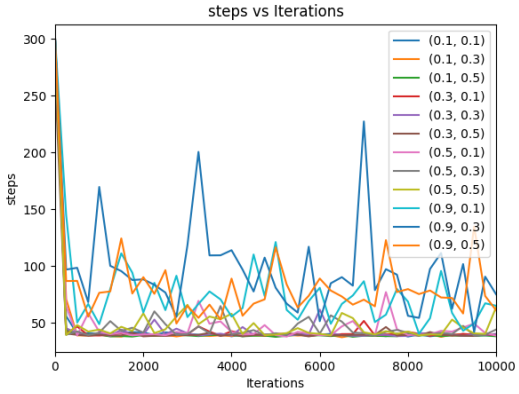
3.3.2 Hard Grid

In hard grid, the convergence is much more distinct across different α values. The variation in rewards, steps is much more haphazard and is difficult to draw conclusions from. However, we can still conclude that $\alpha = 0.1$ preforms well in terms of high reward and few number of steps. Coming to the time plot for the hard graph, it is evident that increasing ϵ value steadily increases the slope of the time lines. Best performance is observed at $\alpha, \epsilon = 0.3, 0.1$, closely followed by 0.1 and 0.1. Hence the best choice of the hyper parameters for the hard grid would be 0.1, 0.1. Indicating slow learning and high belief on the learning.

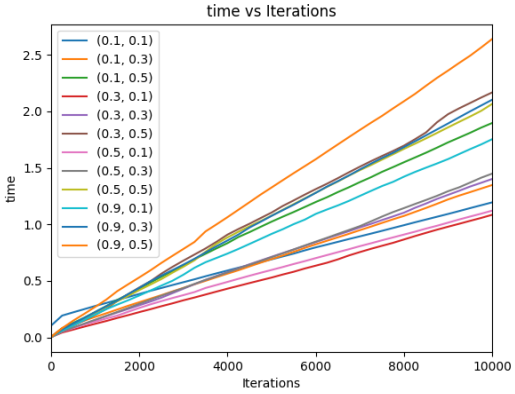


(g) Convergence Plot (h) Rewards Plot

Figure 12 Q learning on easy grid

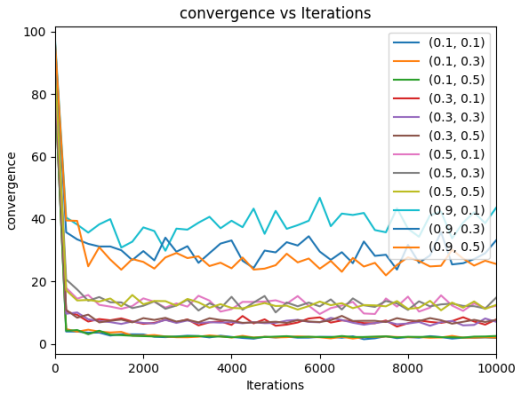


(i) Steps Plot



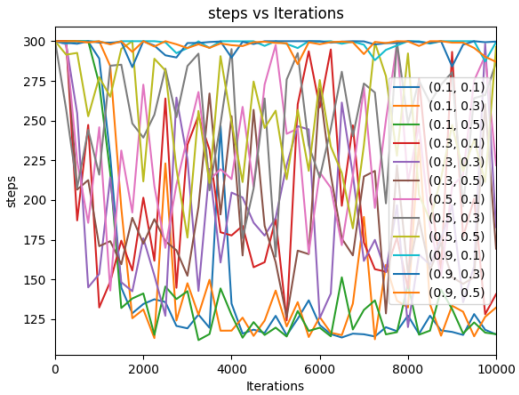
(j) Time Plot

Figure 16 Q learning on easy grid

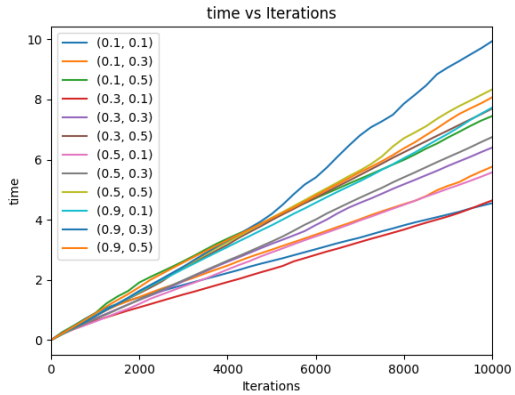


(k) Convergence Plot

(l) Rewards Plot



(m) Steps Plot



(n) Time Plot

Figure 17 Q learning on hard grid

4 Conclusion

We observe that policy and value iteration perform fairly equally. Q learning is a better choice, since we don't require any prior knowledge of the system. In most realistic application of MDPs, transitions T or the rewards R of the environment are unknown.