Assignment 4: Markov Decision Processes

Student Name: Sathish Sampath
GT Account Name: ssampath33

Abstract

This paper discusses two interesting MDP problems and explores two different methods for computing the optimal MDP policy and value – Value Iteration and Policy Iteration. Then, it explores how the Q Learning (Reinforcement Algorithm) performs compared to the aforementioned MDP solvers.

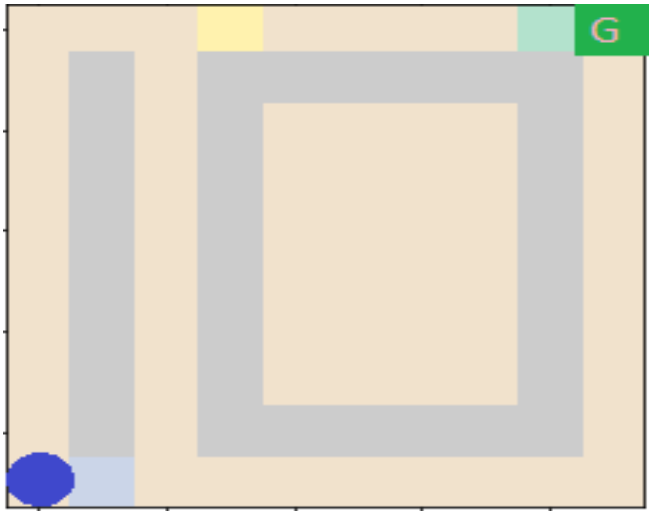
Markov Decision Process

Markov Decision Processes (MDPs) provide a mathematical framework for modelling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDP is based on Markov Property, "the effects of an action taken in a state depend only on that state and not on the prior history"

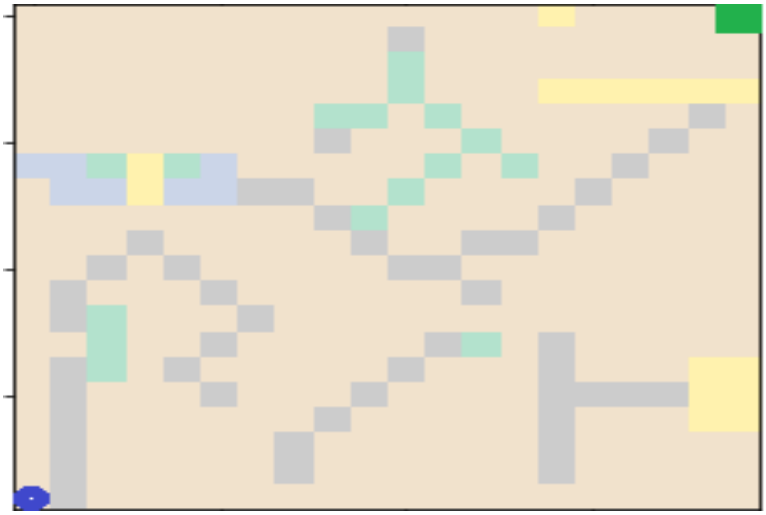
A finite Markov decision process can be represented as a 4-tuple M = {S, A, P, R}, where S is a finite set of states; A is a finite set of actions; P: S × A × S → [0, 1] is the probability transition function; and R : S × A → $\Re$ is the reward function. In this paper, we denote the probability of the transition from state (s) to another state (s ') when taking action, a by P and the immediate reward received after the transition. The Goal of MDP is to find the optimal policy that maximizes expected sum of rewards.

MDP Problems

For the purpose of comparison, two MDP problems are generated with a varying level of difficulty – an Easy Grid problem and a Hard Grid problem. The grid world comprises various stages, transitions, obstacles and varying rewards. The task for the agent is to traverse from the initial state to the goal state, avoiding obstacles(walls) and find an optimal path with maximum rewards(achievable).



Easy Grid (10 x 10)



Hard Grid (20 x 20)

The dimensionality of the Easy Grid is 10 X 10 and dimensionality of the Hard Grid is 20 X 20. Each Grid can have an obstacle, a positive reward or a negative reward, marked by a different color grid. The initial state is in the bottom right corner, marked by a blue circle. The Goal state is in the top right corner marked by a green cell. The grey cells are the obstacles(walls). The light-yellow cells have a lesser negative reward (-1), whereas the light blue and light green cells have a negative reward of higher magnitude: -3 and -5 respectively. The agent can move in 4 directions(actions): Up, Down, Left and Right. The agent has a target success rate of 0.8 and there is 0.067 probability for ending in any of the other three states.

The real values of the grid are shown below. The W represents Obstacle(wall) and there are a few cells with negative rewards (-1/-3/-5). The negative rewards are added to this maze-like grid, to make the grid more similar to the real-world problems.

```
[ 0,  0,  0, -1,  0,  0,  0,  0, -5,  0]
[ 0,  W,  0,  W,  W,  W,  W,  W,  W,  0]
[ 0,  W,  0,  W,  0,  0,  0,  0,  W,  0]
[ 0,  W,  0,  W,  0,  0,  0,  0,  W,  0]
[ 0,  W,  0,  W,  0,  0,  0,  0,  W,  0]
[ 0,  W,  0,  W,  0,  0,  0,  0,  W,  0]
[ 0,  W,  0,  W,  0,  0,  0,  0,  W,  0]
[ 0,  W,  0,  W,  0,  0,  0,  0,  W,  0]
[ 0,  W,  0,  W,  W,  W,  W,  W,  W,  0]
[ 0, -3,  0,  0,  0,  0,  0,  0,  0,  0]
```

Easy Grid

```
[0,0,0,0,0,0,0,0,0,0,-5,0,0,0,-1,-1,-1,-1,-1,-1]
[0,0,0,0,0,0,0,0,0,-5,-5,0,-5,0,0,0,0,0,0,0,W,0]
[0,0,0,0,0,0,0,0,0,W,0,0,0,0,-5,0,0,0,0,0,W,0,0]
[-3,-3,-5,-1,-5,-3,0,0,0,0,0,0,-5,0,-5,0,0,W,0,0,0]
[0,-3,-3,-1,-3,-3,W,W,0,0,0,-5,0,0,0,0,0,W,0,0,0,0]
[0,0,0,0,0,0,0,0,0,W,-5,0,0,0,0,0,0,W,0,0,0,0,0,0]
[0,0,0,W,0,0,0,0,0,0,W,0,0,W,W,0,0,0,0,0,0,0]
[0,0,W,0,W,0,0,0,0,0,0,W,W,0,0,0,0,0,0,0,0,0]
[0,W,0,0,0,W,0,0,0,0,0,0,0,W,0,0,0,0,0,0,0,0]
[0,W,-5,0,0,0,W,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,-5,0,0,W,0,0,0,0,0,0,W,-5,0,W,0,0,0,0,0,0]
[0,W,-5,0,W,0,0,0,0,0,0,W,0,0,0,0,W,0,0,0,-1,-1]
[0,W,0,0,0,W,0,0,0,W,0,0,0,0,0,W,W,W,W,-1,-1]
[0,W,0,0,0,0,0,0,0,W,0,0,0,0,0,0,W,0,0,0,-1,-1]
[0,W,0,0,0,0,0,0,0,W,0,0,0,0,0,0,0,W,0,0,0,0,0]
[0,W,0,0,0,0,0,0,0,W,0,0,0,0,0,0,0,W,0,0,0,0,0]
[0,W,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

Hard Grid

The real-world problems, especially the autonomous robots like warehouse robots and self-driving cars are the inspiration for the grids. The overall map in real-world will have both obstacles and rewards(negative/positive). The obstacles are usually the walls, trees and buildings in self-driving cars, and are walls, boxes in the warehouse scenario. The negative reward cells are representing the condition like uphill ramp, slippery surface or droppers. The agent can avoid those conditions to increase their rewards, but there are chances where it will take them, considering the future higher rewards. These conditions make these two grids interesting.

Implementation

The grid world and MDP are implemented using the BURLAP library, a Java Library. The implementation and exploration are done using the Jython, a Python version running on JVM. The Value Iteration and Policy Iteration are implemented with different Discount value(γ) and the best value is selected. The Q Learning is implemented with different values of learning rate(α) and Epsilon.

Part 2

The two commons algorithms implemented to solve MDP are: Value Iteration and Policy Iteration.

Value Iteration(VI)

Value iteration is a method of computing an optimal MDP policy and its value. Value iteration starts at the "end" and then works backward, refining an estimate of the value. Value Iteration is based on Bellman Equation. It calculates the value of a state based on the value from the neighbouring states. The process converges when the values of a state in two successive iterations are same or if the difference is too low.

Policy Iteration(PI)

Policy iteration is another method for computing optimal MDP Policy. It starts with a policy and iteratively improves it. It starts with an arbitrary policy π (an approximation to the optimal policy works best) and carries out the following steps. The Policy iteration method involves repeatedly evaluating a policy and improving the policy for each state, until the convergence is reached, where the policy remains almost same.

The Value Iteration and Policy Iteration are implemented to the Easy grid and Hard grid and the results are compared below. The VI and PI continues iterations till the Convergence delta value reduces below a threshold value(say 1e-06).

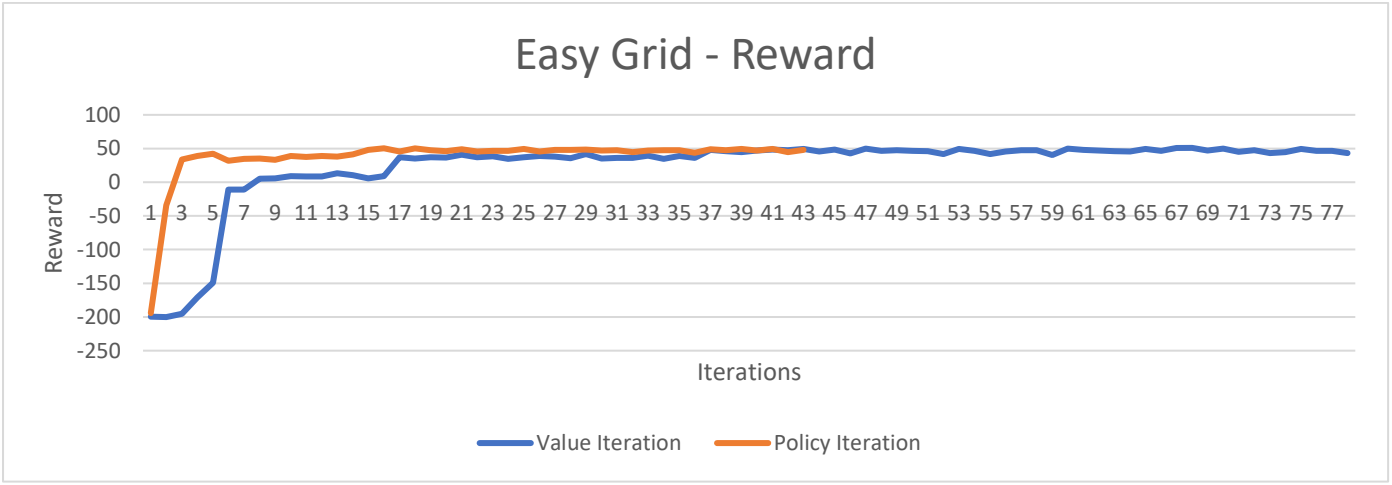
Easy Grid Problem

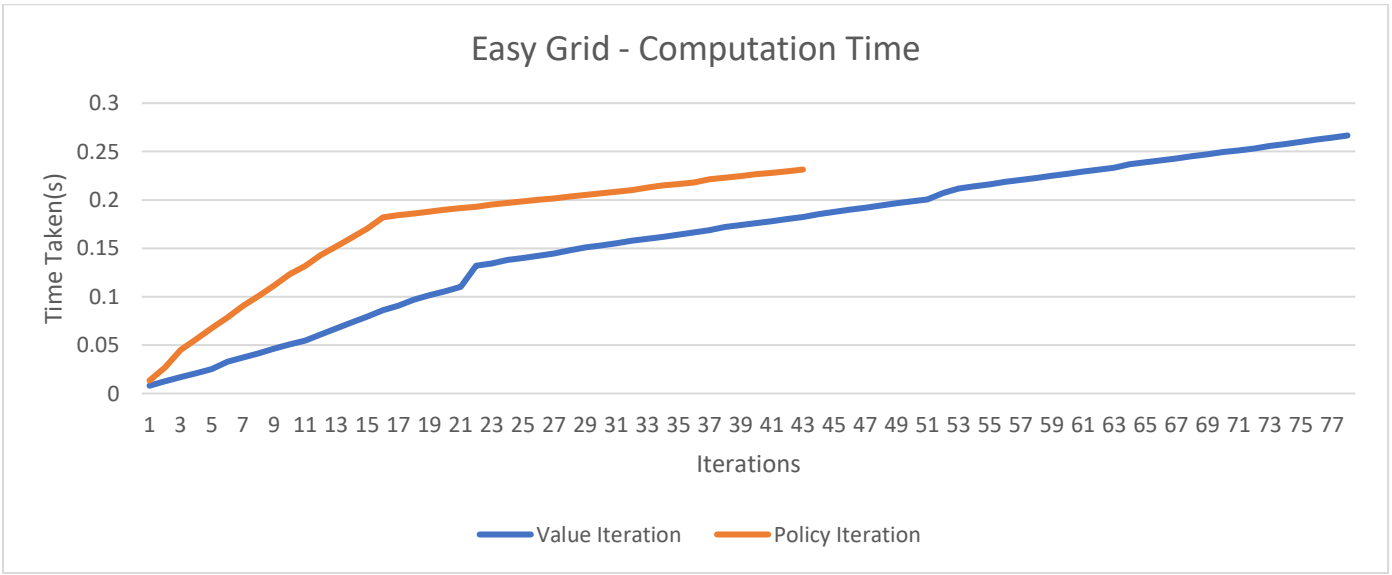
Initially, the Value Iteration and Policy Iteration were executed with varying discounts and the results when the condition(delta < threshold) is reached, are noted.

Discount	Value Iteration					Policy Iteration				
	Iteration	Time Taken	Reward	Steps	Convergence Delta	Iteration	Time Taken	Reward	Steps	Convergence Delta
0.1	7	0.033812	-200.16	200	8.38E-07	4	0.041864	-192.04	193.02	3.15E-07
0.3	13	0.028598	-167.1	175.86	3.47E-07	7	0.023105	-175	182.62	5.65E-07
0.5	21	0.044261	-189.06	192.94	4.98E-07	10	0.028786	-186.52	189.52	7.10E-07
0.75	46	0.089089	-199.58	200	8.52E-07	19	0.052259	-166.82	175.58	8.72E-07
0.99	78	0.154145	44.96	50.7	8.26E-07	43	0.19623	49.06	46.24	9.83E-07

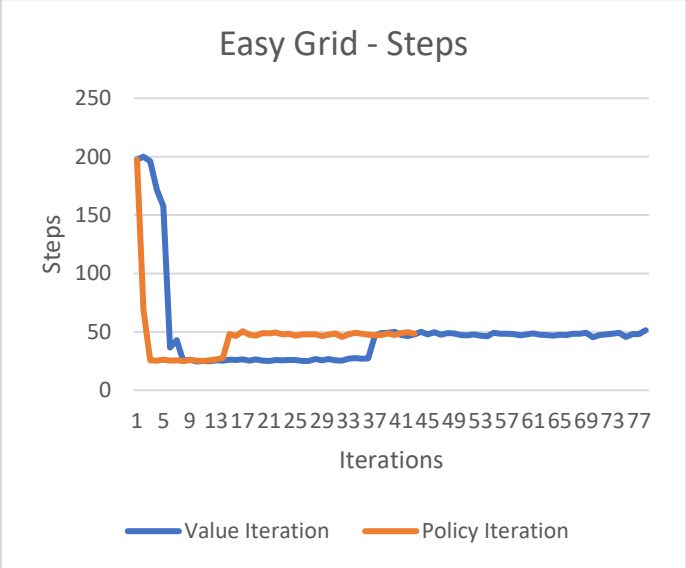
Even though the discount value = 0.99 takes longer to converge in both Value Iteration and Policy iteration, it has higher rewards when compared to other iterations. Also, some results have negative rewards, this might be because of the local optima. The algorithm would have got stuck in local optima and converged.

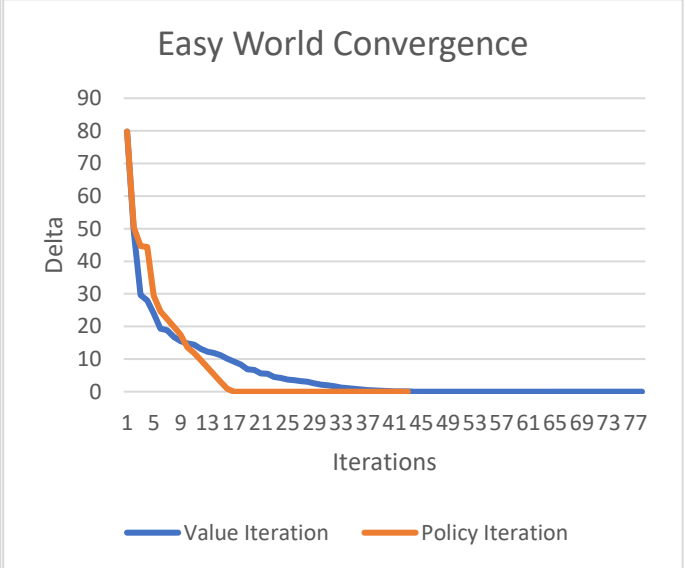
The Discount Value γ = 0.99 is selected and the algorithms are compared. The Policy Iteration has converged much faster than the Value Iteration. The rewards plot for both the VI and PI was increasing rapidly for lower iterations. As the iterations increase, the VI and PI plots stabilize and almost becomes a straight line. This indicates that the convergence has happened. The algorithm's iteration is programmed to stop when the convergence delta value reduces below 1e-06(threshold). The convergence delta was still higher than the threshold, so it continued till 43 iterations in PI and 78 iterations in VI. The VI converges to ~45 around 36 iterations and the PI converges to ~49 around 16 iterations.





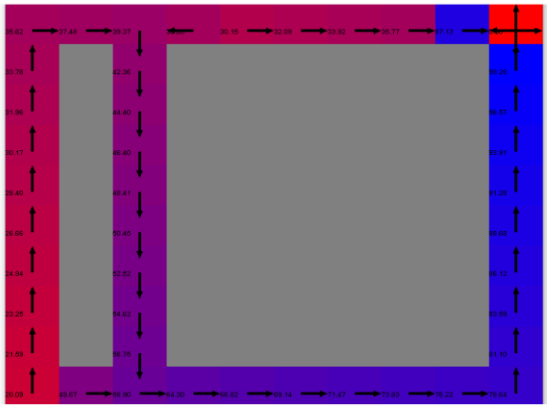
The total computation time taken by the VI and PI are compared below. The PI initially takes higher computational time with the increase in iterations, but around the 16 iterations, the slope reduces to a lesser value. Comparably VI takes lesser time for each iteration but PI converges at overall lesser time compared to VI.



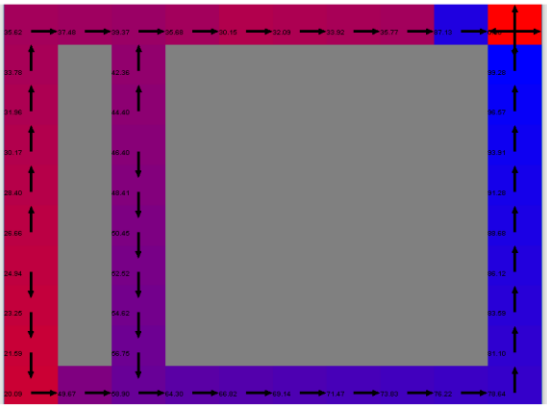


With respect to the steps, the PI drops to lower steps much sooner than the VI. Around 16 iterations, the PI takes a small rise in number of steps and remains ~46 steps. The number of steps for VI stabilizes only after 36 iterations and for higher iterations, it requires ~50 steps. With respect to convergence, both the iterations reduced to lower values at a faster rate. Around 16 iterations, the PI plot stabilized and around 36 iterations, the VI plot stabilized with very minor differences over the higher iterations.

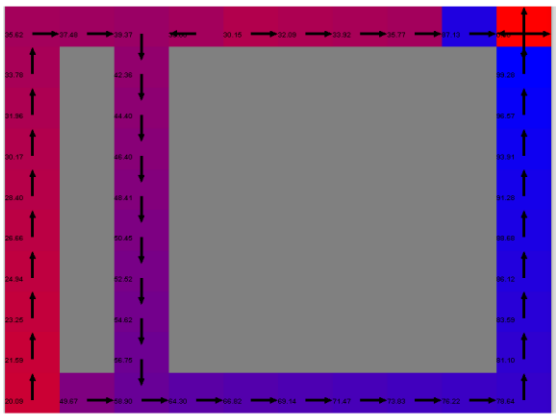
The optimal policy map output from both the VI and PI are visualized for 5th iteration and the final iteration (where delta value < threshold).



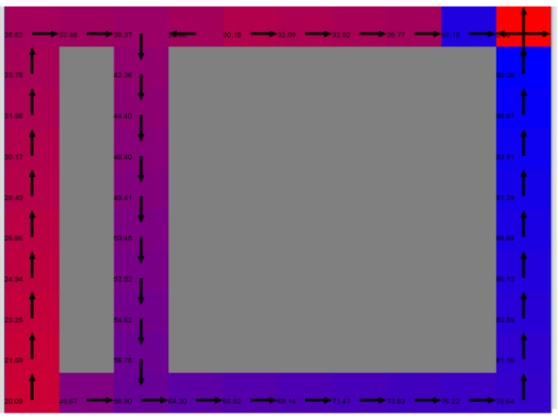
Value Iteration n = 5



Policy Iteration n = 5



Value Iteration n = 78



Policy Iteration n = 43

The red-blue pattern is used for indicating the amount of exploration done by the VI/PI. The dark red indicates very less or no exploration and the blue indicates a higher number of exploration. It is clear from the map that both VI and PI came up with the same final optimal policy map.

For Easy Grid: The Policy Iteration is considered better for this problem as it converges with a better reward for the problem.

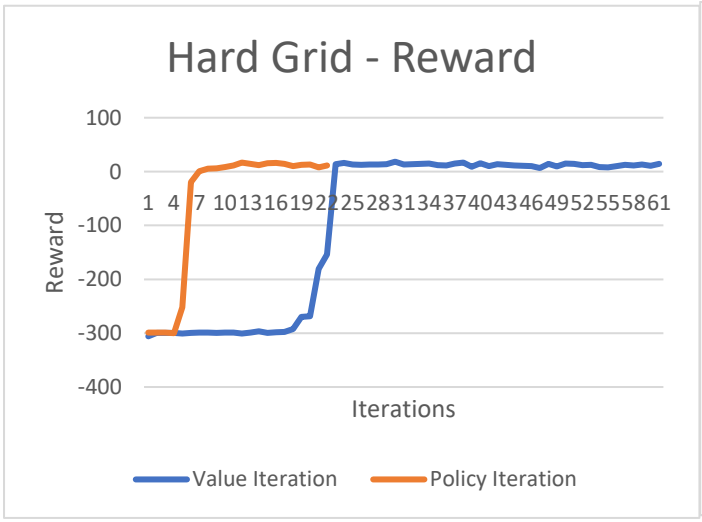
Hard Grid

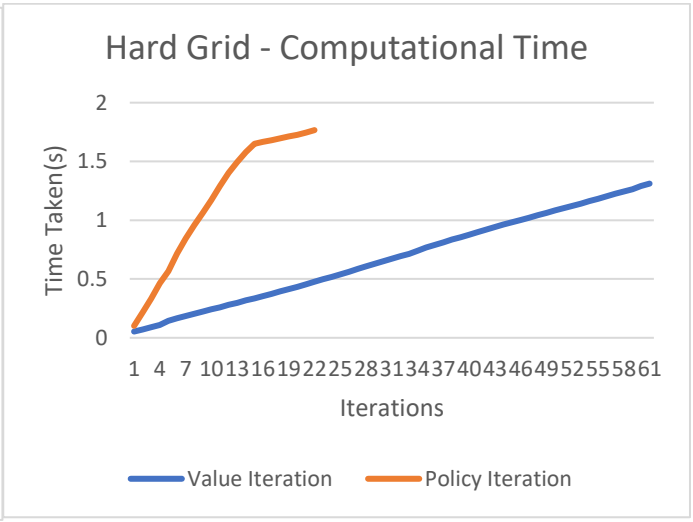
Similar to Easy Grid, the Value Iteration and the Policy Iteration was executed with varying discounts and the results when the condition (delta < threshold) is reached, are noted.

Discount	Policy Iteration					Value Iteration				
	Iteration	Time Taken	Reward	Steps	Convergence Delta	Iteration	Time Taken	Reward	Steps	Convergence Delta
0.1	5	0.161962	-299	300	7.28E-07	7	0.169493	-299	300	8.89E-07
0.3	7	0.216904	-299	300	5.47E-07	13	0.229672	-299	300	3.82E-07
0.5	10	0.432302	-299	300	5.71E-07	21	0.369674	-299	300	5.86E-07
0.75	16	0.66219	-299	300	3.25E-07	47	0.837819	-299	300	9.06E-07
0.99	22	1.526187	12.25	63.59	8.08E-07	61	1.105574	15.19	62.55	9.47E-07

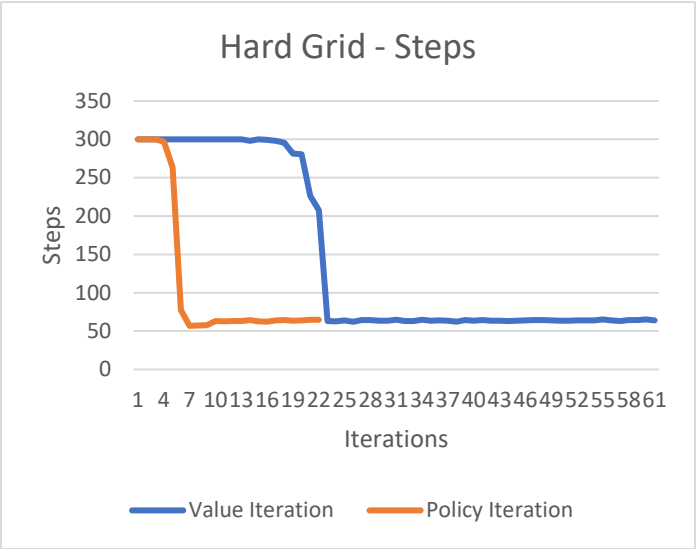
Even though the discount value = 0.99 takes longer to converge in both Value Iteration and Policy iteration, it has higher rewards when compared to other iterations. Also, some results have negative rewards, this might be because of the local optima. The algorithm would have got stuck in local optima and converged. The Discount Value γ = 0.99 is selected and the algorithms are compared.

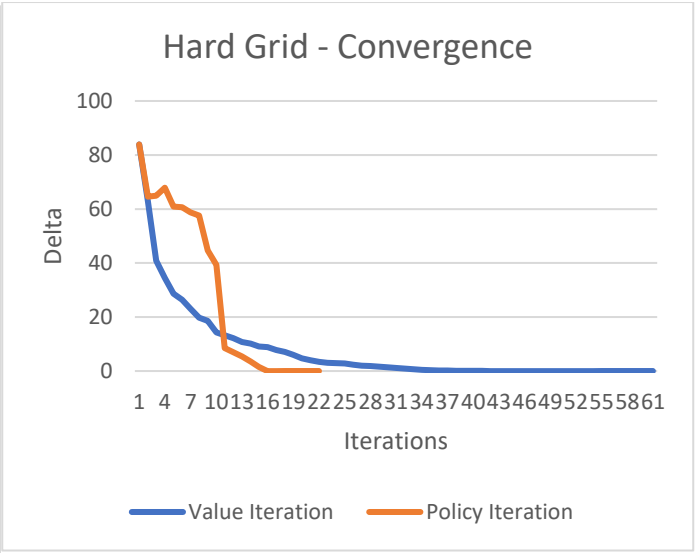
The PI executed for 22 iterations and VI executed for 61 iterations before satisfying the condition (delta < threshold). The rewards and computational time taken by PI and VI for different iterations are plotted. The PI converges much faster around 10 iterations to ~12 reward, whereas the VI converges around 24 iterations to ~15 reward. The Computational time taken for each iteration is much higher for PI when compared to VI. In VI, the computational time constantly increases.



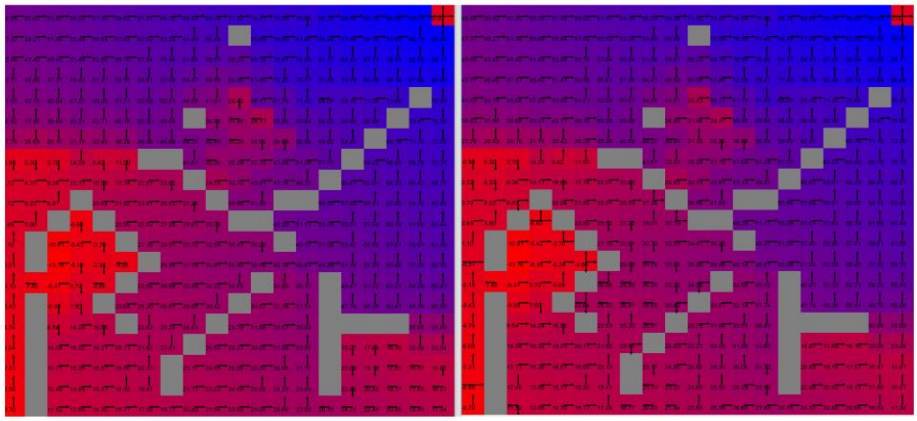


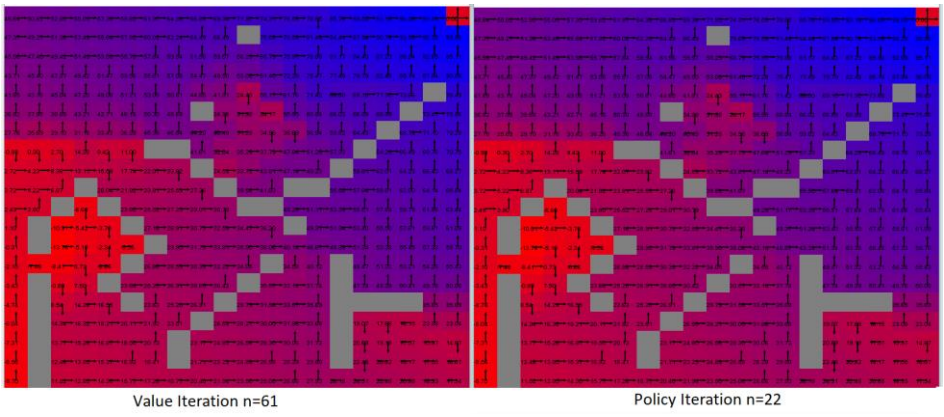
The number of steps taken by PI converges around 10 iterations to ~63 steps, whereas the VI converges around 24 iterations to ~62 steps. The Convergence delta of the PI is different from the Easy grid world. For initial iterations(<10), the delta value is much higher and it drops suddenly around 10 iterations.





The Optimal policy map for Hard grid by the VI and PI are visualized for 5th iteration and the iteration which satisfied the condition (delta < threshold). The same red-blue color code used in the Easy grid are used for the Hard grid visualization also. Both the PI and VI show the similar optimal policy map and both algorithms have explored similar levels.





For Hard Grid: Policy iteration converges at a lower iteration with similar rewards to Value Iteration, but the computational time is slightly higher for the Policy iteration.

Comparison - Policy Iteration and Value Iteration

- Policy Iteration converged at a lower iteration for both the problems.
- Policy Iteration resulted in slightly higher reward in both the problems than the Value Iteration.
- Policy Iteration takes higher computational time compared to the Value Iteration, for the similar number of iterations. Policy Iteration might have taken longer time, since it is finding the optimal policy for each state, which is computationally costly when compared to value computation.

- Policy Iteration for MDPs do not result in non-optimal local optima, because it is possible to improve an action for a state without affecting other states, whereas updating values directly can affect.

From the comparison, Policy Iteration seems to be a better planning algorithm for the MDPs.

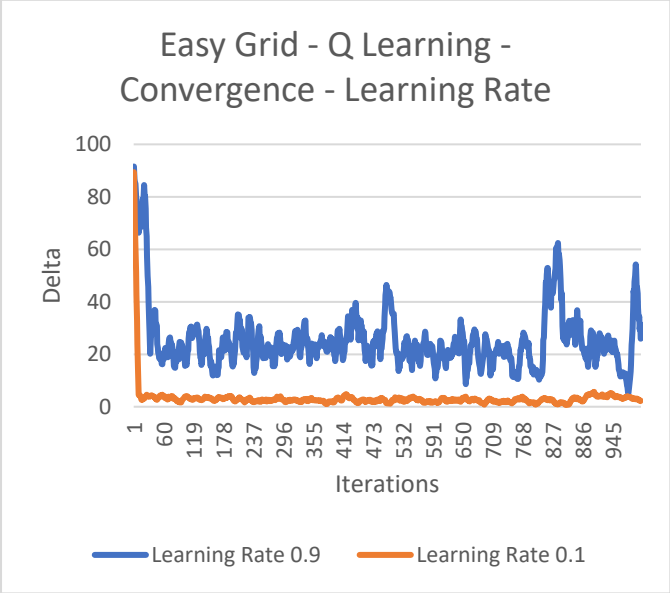
Part 3 Q Learning

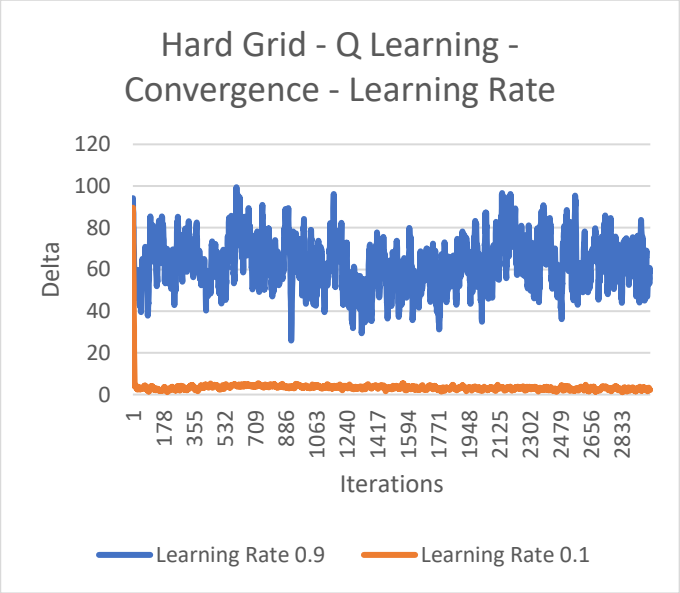
The Q-Learning is an off-policy, model-free RL algorithm based on the Bellman Equation.

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

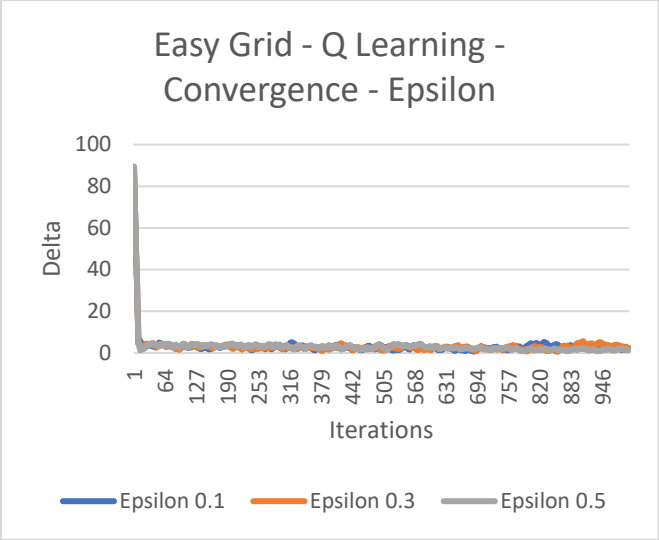
The Q Learning tries to maximize the Q value and get an optimal Q value for each state in the grid. This value is used to solve the MDP. The Q Learning is implemented to solved the above mentioned MDP problems. There are two main parameters in Q Learning: First, learning rate(α), which controls the importance of the future states against the immediate reward and second, Epsilon, which controls the probability of not taking the desired action from a state. Both the parameters are varied to determine the optimal set.

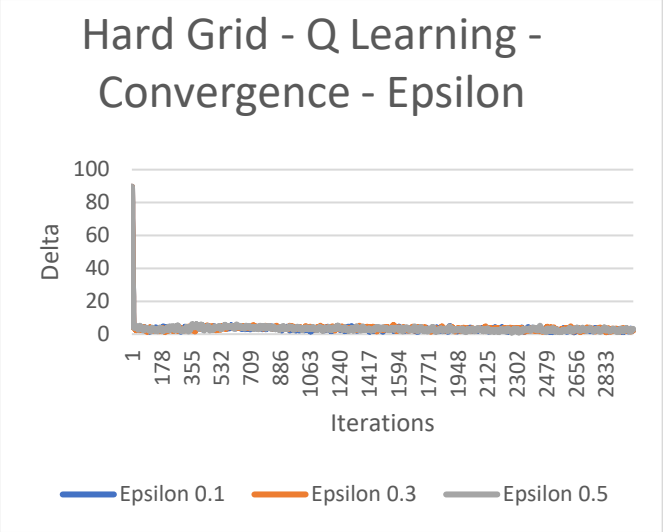
For determining the perfect Learning Rate for Easy and Hard Grid, the Q-Learning is executed with a fixed Epsilon(0.3) and the Learning rate is varied. The performance with respect to convergence is compared for both the problems. From the plot, it is clear that the learning rate 0.1 converges much faster than learning rate 0.9. This proves the fact that immediate rewards are more important than the future rewards and the optimal policy can be obtained faster using the greedy policy.



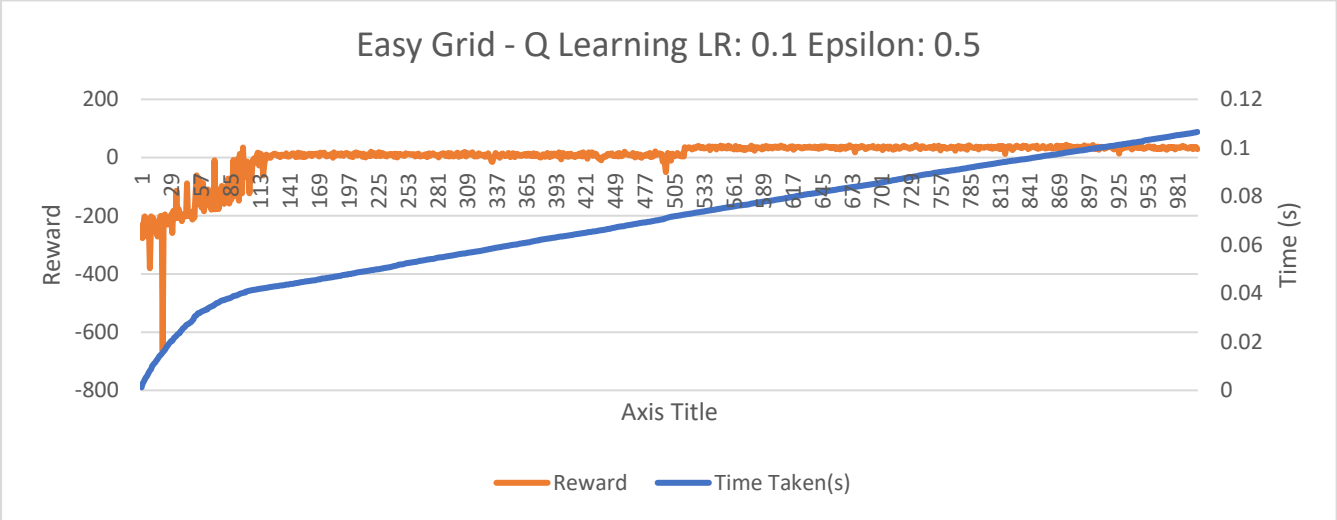


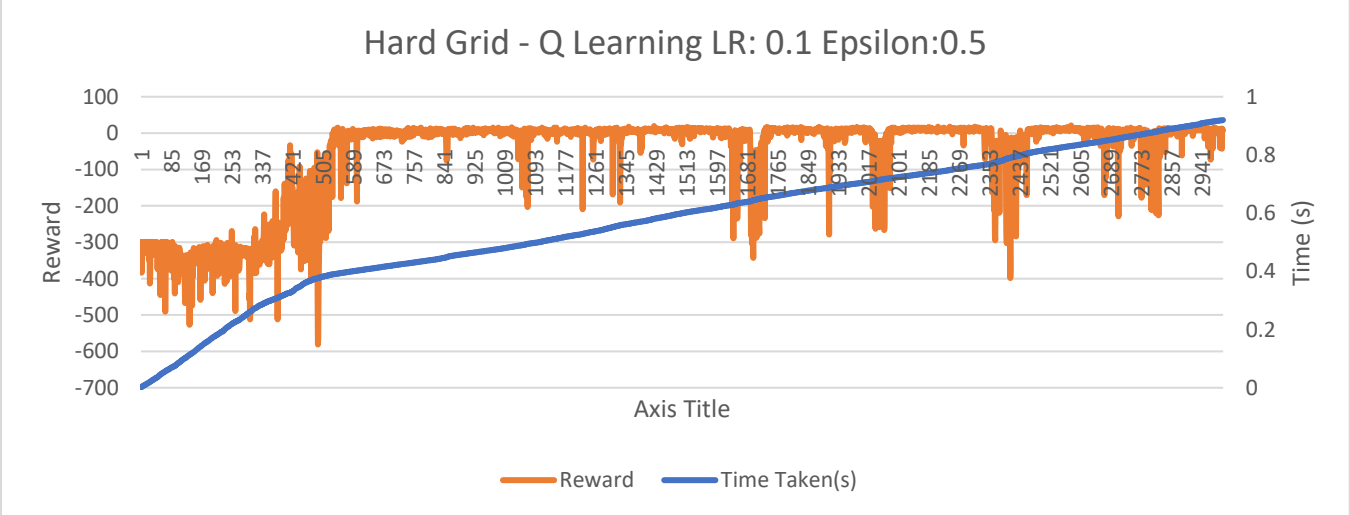
For determining the best Epsilon, the learning is fixed to previously selected value (0.1) and the Q Learning is executed with varying Epsilon values. The plots for all three epsilon values show a similar pattern. But the Epsilon 0.5 plot converges much faster for both the problems. The higher epsilon values, the Q Learning tends to select more random actions and explore. This leads to earlier convergence, compared to lower Epsilon values. So, it is selected as the optimal Epsilon value.





The Q Learning is executed over the Easy Grid and Hard Grid with parameters (Learning Rate: 0.1, Epsilon: 0.5)



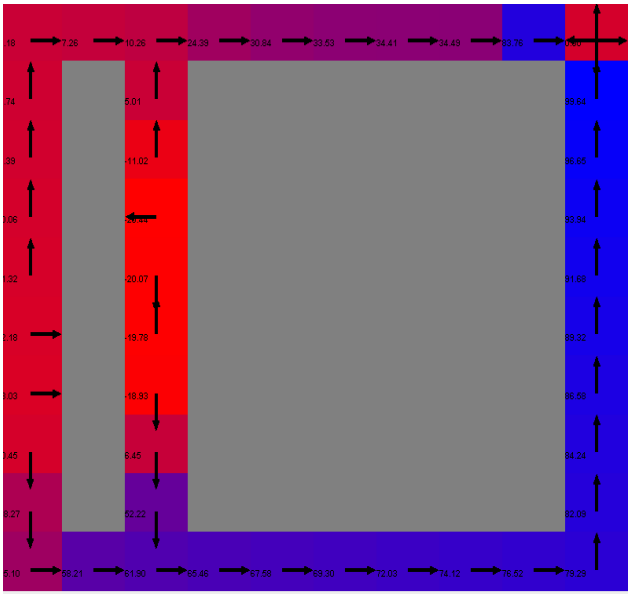


The Easy Grid converges much faster than the Hard grid, but there are few troughs in the reward plot. The Easy grid initially converges near 120 iterations, but there is a small trough near the 500 iterations and the reward plot re-stabilizes to a higher value after 550 iterations. This indicates that Q Learning explores all possible states with their future actions and there existed a combination with lesser reward. But after that, the algorithm was able to attain a new optimal Q from that conditions and re-converge to a higher reward value. For the Easy Grid, the computational time taken for each lower iteration (<500) is much lower than the time taken for VI and PI.

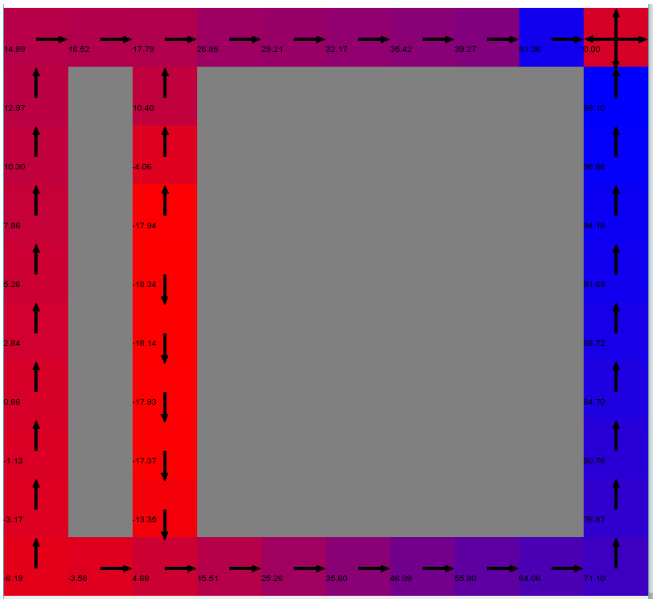
The Hard Grid has the reward plot with more number of troughs and after each trough, the rewards plot converges to a slightly higher value. As mentioned earlier, this could be an indication that Q-Learning is exploring all

states and their future rewards and correcting its Q values as it computes a new optimal policy. This way, Q-Learning tries to avoid getting stuck in the local optima. The computational time is lesser compared to VI and PI. After converging, the computational time remains at a constant slope with respect to the iterations for both the problems.

The Policy visualization for Easy Grid and Hard Grid are given below. The same red-blue color coding is used for the map visualization.

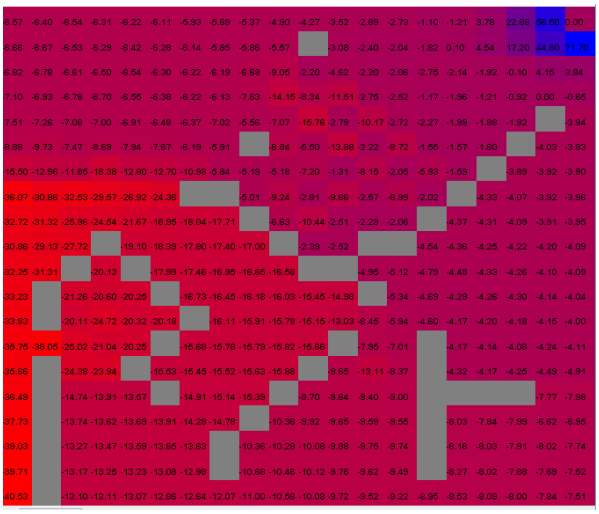


Easy Grid n = 5

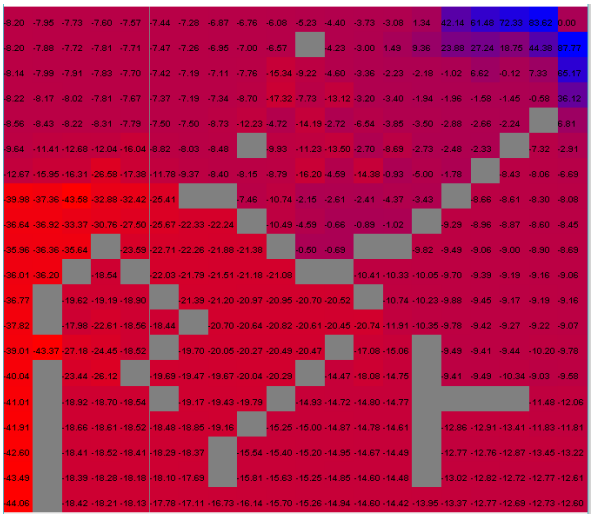


Easy Grid Optimal Policy

The Q learning was able to obtain an optimal policy similar to the VI and PI for the Easy Grid. But for Hard Grid, the optimal policy is not obtained, this might be because of lower number of iterations for that dimensionality. The Q Learning rewards plot showed small improvement as the iterations increased. So the Q Learning might be converging for higher value of iterations.



Hard Grid n = 5

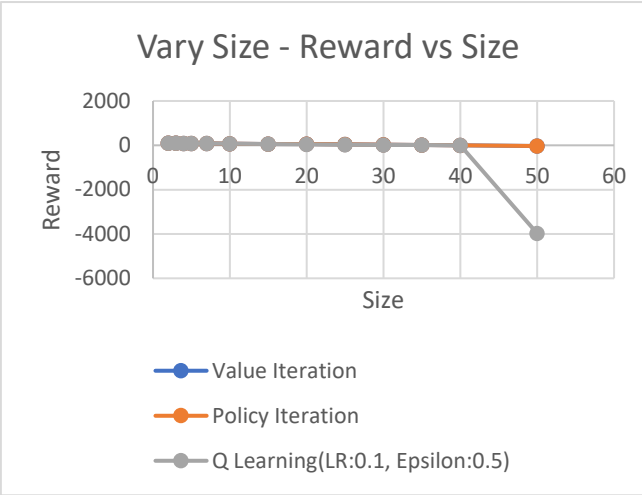


Hard Grid Final Policy

Other Experimentation Results

The Q Learning initial Q matrix was filled with varying values from negative(-100) to positive(100), instead of zero and the performance is compared. For negative initial Q values, the Q Learning was converging in a lower iteration for very low reward: ~ -200 for Easy Grid and ~ -300 for Hard Grid. As all the states have a negative high reward (-100), instead of finding the global optima, the algorithm gets settled with local optima. For higher positive(+100) Q initial matrix, the agent converges but not with the highest optimal reward. This is also an indication that it did not reach the global optima.

The Size of the grids were varied and the performance of Q Learning, VI and PI are compared.



The Q Learning is executed with parameters Learning Rate 0.1 and Epsilon 0.5. From the plot, the following inferences are made

- The VI and PI's reward remains almost same with the change in the grid size. The PI takes higher computational time and steps compared to the VI.
- The Q learning performs poorly with higher value of dimensions(size). Either the iterations should increase or parameters has to be altered to get higher reward.

Conclusion

- The Q Learning is the fastest of all three methods. But it did not converge much easily as the dimensionality increases.
- The Immediate reward has much more effect and makes the Q-Learning converge faster than the future rewards.
- The initial states/rewards affect the optimal policy and it might even make the algorithm get stuck in local optima.
- The Policy Iteration is much better for both the problems together, as it converged much faster(with lesser number of iteration). The Value iteration takes lesser amount of time with each iteration.
- Policy Iteration is much better in avoiding local optimal than the Value Iteration.
- Q Learning performs with lesser reward for higher dimensions and lower iteration. The number of iterations has to be increased with the dimensions.

References

1. http://artint.info/, Artificial Intelligence: Foundations of Computational Agents, second edition, Cambridge University Press, David Poole, Alan Mackworth.
2. https://www.ntu.edu.sg/home/boan/papers/AAMAS16_MDP.pdf, Measuring the Distance Between Finite Markov Decision Processes Jinhua Song, Yang Gao, Hao Wang, Bo An.
3. Code Source: https://github.com/JonathanTay/CS-7641-assignment-4, CS-7641-assignment-4 by Jonathan Tay
4. http://burlap.cs.brown.edu/, Brown-UMBC Reinforcement Learning and Planning (BURLAP) java code library.