**Name:** Het Piyush Sheth **GTID:** 903393644

CS 7641-HW4

**Introduction (Section 1)**

This assignment deals with the understanding of the concept of Reinforcement Learning, especially Markov Decision Processes. Reinforcement learning falls under the umbrella of Machine Learning where it is generally used to solve problems that involve an agent’s interaction with an environment. This interaction is more formally defined using two parameters: states and actions. Furthermore, Markov Decision Processes (MDP’s) are frameworks that are commonly used to solve these reinforcement learning problems. What they mean is that if we have information about the current state and the consequent action, then the future state relies only on them and is independent of any past state.

Majority of the MDP’s are stochastic in nature, i.e. taking an action *a* on a state *s* won’t guarantee that we reach a state *s’*. Basically, external factors could result in altering the assumed course of the action on the state. Alternatively, if the problem were deterministic in nature and not stochastic then we could just use something like a graph structure, where states are nodes and actions are edges, to solve the issue. In order to formally define a MDP, we use five different sets (or tuples) of values. They are as follows:

|  |  |
| --- | --- |
| **S** | It is a set of possible states in the environment that the agent can go to. |
| **A** | It is a set of possible actions using which the agent can go from one state to another. |
| **P*ₐ*(*s’, s*)** | It refers to the probability of reaching state *s’* from state *s* given that the agent performs an action *a*. |
| **R*ₐ*(*s’, s*)** | It refers to the immediate reward received after making a transition from state *s* to state *s’* by performing an action *a*. |
| **𝛾** | It is a discount factor. Its value always belongs in the range of (0, 1). It is multiplied to the future rewards in order to allow the model to focus more on immediate rewards. |

Now that I have established what MDP’s are, I want to elaborate further on the two MDP’s I have chosen to do my analysis over. For the purposes of this analysis, I have chosen to select two grid world problems. Basically, both the problems are based on a grid structure where I have *n* cells stacked horizontally to form a row and *n* such rows stacked vertically to form a 2-d matrix (grid). The difference between the two problems is the number *n* and how the various reward cells and walls are arranged around that grid structure. In both the problems, the starting cell is the bottom left one and the goal cell is the top right one. I have termed the grid with the smaller number of cells as the easy problem and the one with a greater number of cells as the hard problem.

In both the problems, there are four possible directions that an agent can move in from a given state. They are: *up, down, left, right*. When an agent has decided to take an action, there is a probability of 0.8 that the agent will go in that direction and the remaining 0.2 is uniformly distributed between the other three directions that the agent hasn’t decided to go in.

The reason I find grid world problems interesting is because of how seamlessly they can be modified to be interpreted as a real-world problem. It is quite analogous to saying how coordinates can be used to pin point locations. Now, obviously it might be a bit vague for the human mind but it has proven to work well for a machine. For example, this grid world problem can be assumed to be that of a path finding problem used by maps, where a shorter path can be rewarded more. Also, it can be used by autonomous robots to navigate within warehouses to find the right inventory and the walls can be seen as the other inventory occupying space in the grid. In the next section, I talk in detail about the easy and the hard problems that I have chosen.

The two model based methods I use to solve the MDP’s are Value Iteration and Policy Iteration. Both these models are based on the notion that an agent can plan its actions in advance, i.e. the agent already knows the MDP model of the world. (Like an autonomous robot would always know the model of the warehouse).

**Value Iteration**

In value iteration, we assign a value function (*V(s)*) to every state that denotes how good a state is for an agent to be in. In addition, we also define *Q(s,a),* which denotes how good of a move it is for an agent in state *s* to select an action *a*. With these two measures, we now devise an algorithm that starts with randomized values for V(s) and using Q(s,a), it iterates over the value functions of the various states. Thus, finally it converges to give the optimal values and it is worthwhile to note that this algorithm always converges.

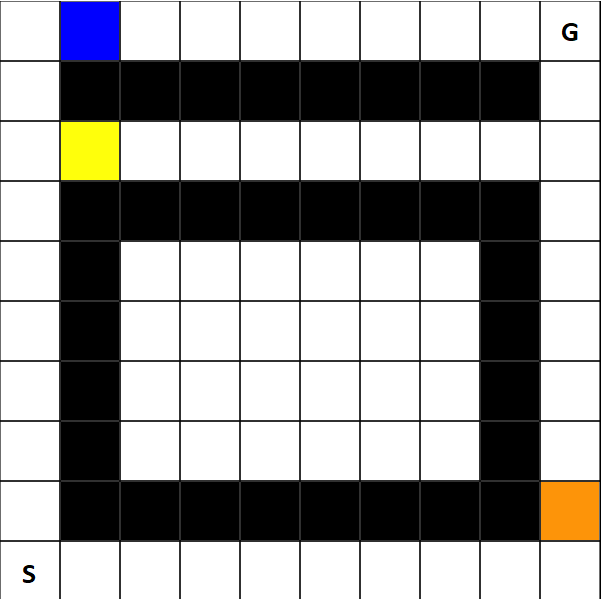
**Policy Iteration**

We know that the aim of the agent is to get to the goal in the best possible way, and to achieve this the most important metric that the agent should care about is the policy that it needs to follow when at a given state *s.* So the focus of our algorithm should be to find the optimal policy. Now while the value iteration approach does succeed, it is not the most efficient way as it improves the value function at each step and continues to do so until the values converge. An efficient way to do that would be to define the policy again at each step and then estimate the value after that. This way, our focus is more directed towards the policy itself. This method is known as policy iteration.

As for the value iteration method, the policy iteration method is also bound to converge. In fact, this method converges faster (i.e. less number of iterations) than the value iteration method.

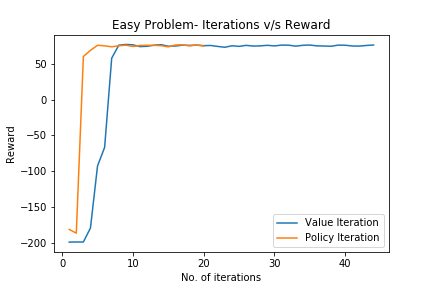
**Easy Grid World (Section 2)**

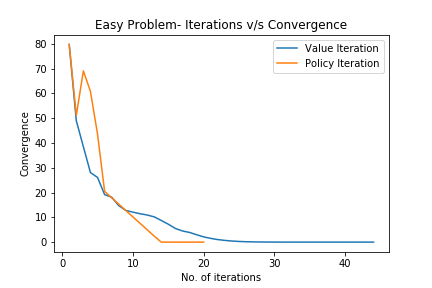
In this section, I first describe the problem that I am going to analyze, then I compare the various results obtained when the Policy and Value Iteration methods are applied to this problem. These results include the number of iterations needed to converge, the time taken, the total rewards and the number of steps required. In addition to that, I also show the grids obtained at two points, once during the 5th iteration and once during convergence for both policy and value iteration.



The black cells are walls, blue cell has a reward of +3, yellow cell has a reward of -1 and the orange cell has a reward of -3. This problem is interesting to analyze because it has three equally feasible paths to the goal from the start but all the paths have at least one cell with a different reward than normal. This will help me identify how the various methods behave when given a situation where the determining factor is not the length of the path but the reward along the way too. (Think of it as three paths to reach from point A to point B which are of the same distance but we have to choose the one with the least traffic, i.e. greatest reward.

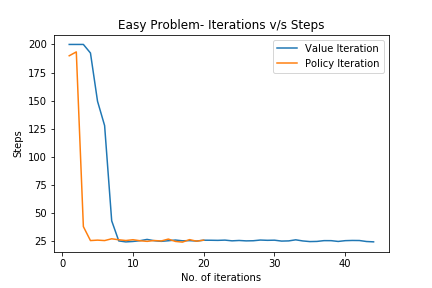
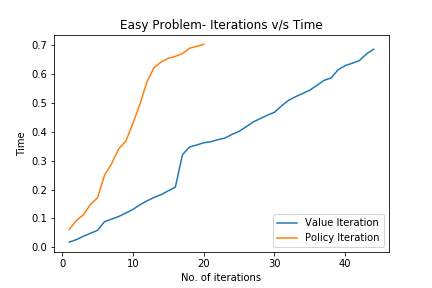
The first and the most obvious thing to notice from the below graph is that policy iteration converges in lesser number of iterations than value iteration. This is very much in sync with the explanation mentioned above (about how policy iteration converges faster). Also one interesting thing that we can notice from the convergence graph is that there is a slight uptick in the policy iteration plot and then it decreases steeply before converging. This uptick is actually due to the structure of the problem. As you can see above, I have, at all three paths, placed cells with varying rewards. Now, the moment the policy iteration step discovers that, the uptick arises. Soon after that, it adjusts its values to accommodate the cells with varied rewards and then the gird is pretty much free of any obstacles, so the method converges. Here, I define convergence as when the change in the values is lesser than 10-6.



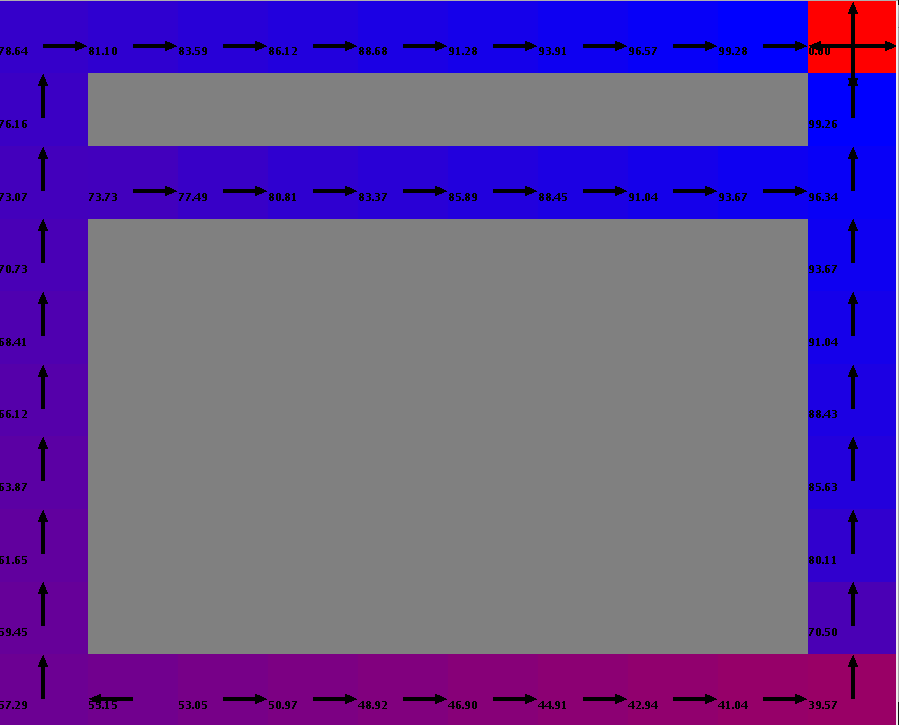


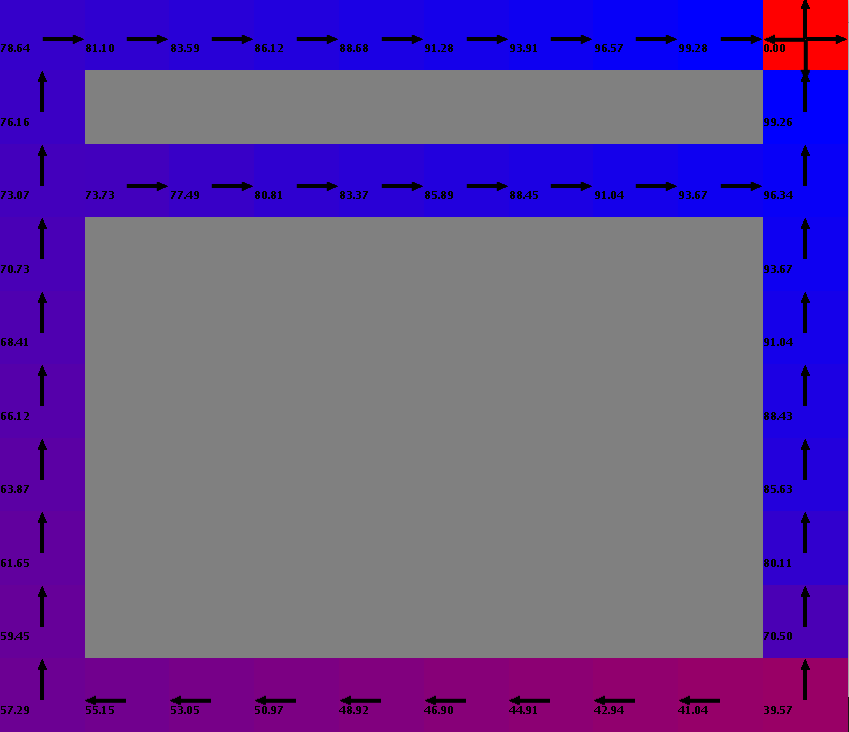
Looking at the second graph above, the reward v/s no. of iterations chart also follows a similar trend as the convergence graph. That is, the policy iteration method reaches the highest reward with fewer numbers of iterations than the value iteration method. As for the smoothness of both the line plots, we notice that the value iteration line has its reward value as constant for the first couple of iterations. This is because value iteration starts its algorithm by assigning random value functions to all the states and thus it takes a couple of iterations before the rewards start increasing.

Looking at the Iterations v/s time graph below, we can see that both the methods take approximately the same amount of time to run. But one thing worth noticing here is that policy iteration ideally converges in half as many iterations as value iteration. So, the time taken per iteration by policy iteration is much higher, and this makes sense because policy iteration involves solving a system of linear equations and calculating a new policy at each step. On the other hand, value iteration does not have to do that and thus takes a lesser amount of time.



**In addition, from the graph above, we can see that policy iteration requires the same number of steps as taken by value iteration. That is both of them provide the same kind of solution. Now, the policy iteration method finds it faster than the value iteration method. So if we wanted to find the best path to take from one point to another, we can get a similar answer by both the methods. Also, the value iteration method stays flat for the first few iterations. This is because the value functions are randomly assigned to the states for the first run and thus the value functions stay semi-stochastic for the first couple of iterations.**

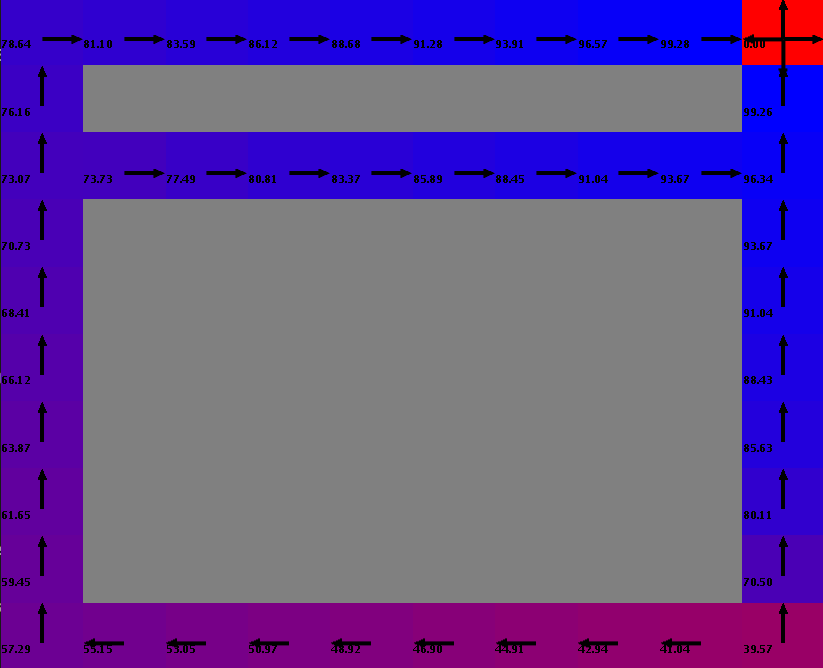
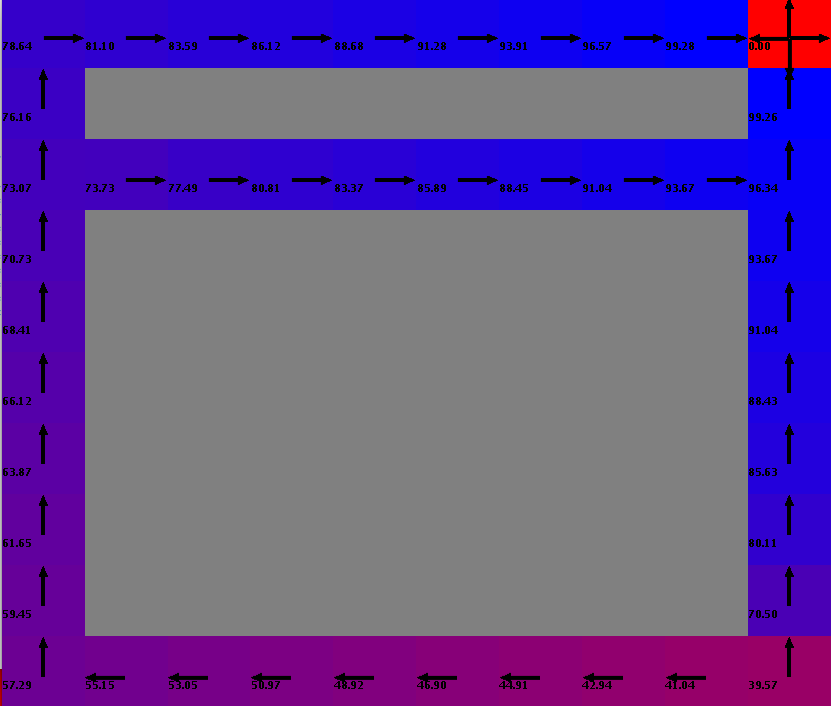
****



Value iteration (5th iteration) Value iteration (44th iteration)

Above are the grids for value iteration after 5 and 20 iterations respectively. In the above grids, a blue color means that the algorithm has visited is more often than the one with a lighter shade. In terms of the total reward, the topmost row has the greatest reward, followed by the middle row and finally the bottom row. This makes sense because the topmost row has a blue cell which has a reward of +3, whereas the bottom most row leads to the path through an orange cell, with a reward of -3.

One more thing that I noticed, which is a significant difference between the two grids, is the direction of the policies for the bottom row in both the grids. They point towards the right in the first one and towards the left in the second one. This is accurate because as the function iterates, it tends to go away from the cell that offers the least reward and towards the cell that offers the most. Thus, during convergence, it would prevent an agent from choosing a path with low reward when a better path exists with equal or lesser number of steps to the goal.



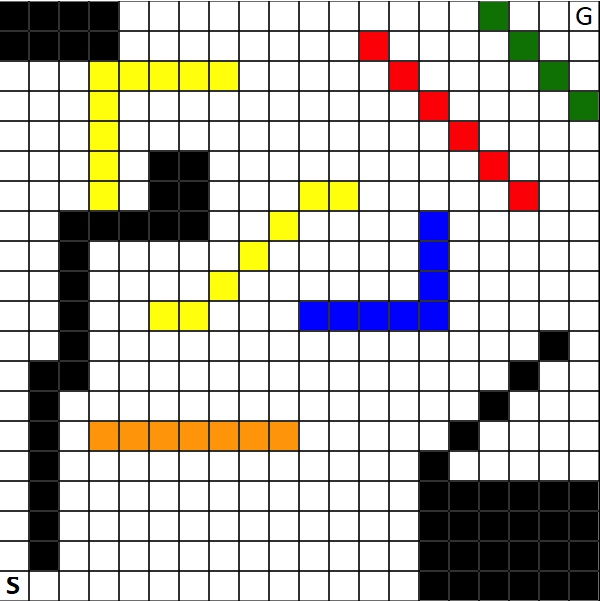
Policy iteration (5th iteration) Policy iteration (20th iteration)

From the grids above, we find out that the policy iteration approach has reached a near optimum solution in the 5th iteration itself, whereas, the value iteration was far from the optimum at this stage. Again, this can be validated by the fact that the policy iteration approach updates the policy and then reiterates over the updated policy as opposed to the value iteration method.

We can see that the final result (the structure of the grid at convergence) is quite similar for both the value and policy iteration approaches. This is accurate as both are converging towards the optimal solution and also from the line plots above, both converge towards a similar optimum. Now this happens because the problem under consideration is easy and thus both the approaches can solve it fairly quickly.

**Hard Problem (Section 3)**

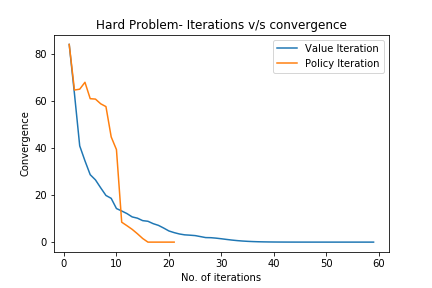
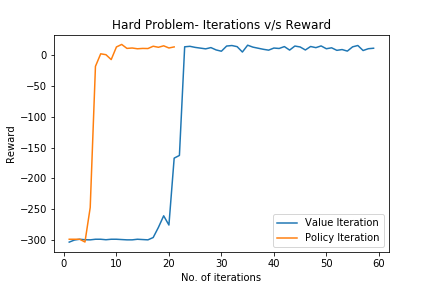
The structure of the analysis for this section remains the same as that of the previous section (I have intentionally done this, so that it encourages the reader to gauge the subtle differences between the easy and hard problem).



The black cells are walls, blue cells have reward of +3 each, yellow cells have reward of -1 each, orange cells have reward of -3 each, red cells have reward of -5 each and the green cells have reward of +5 each. This problem, based on its appearance might seem random at first but has a lot of subtle quirks that make it quite similar to a real life situation. For the purposes of explanation, let’s take the example of path finding. That is, an agent wants to find the best possible path from the start to the goal. Now, the bottom right ladder like wall structure is actually encapsulating the situation where the agent gets stuck in a dead end. Then, the yellow

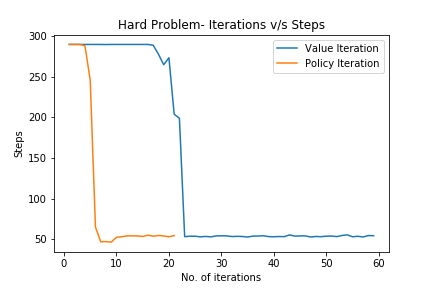
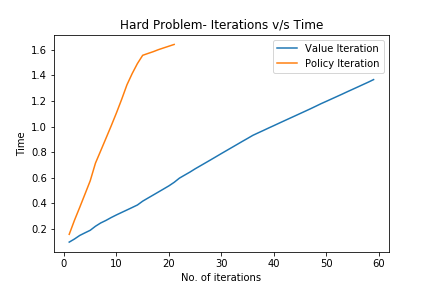
structure on the top right (surrounded by walls) is an example of a path that is short and feasible but slightly negatively rewarded (something like a shorter route but with some traffic). Then the red stretch is the most negatively rewarded region. This is analogous to most of the roads to a certain destination having some construction work going on. (What would be the best choice in that case? To go through the construction road or take a detour early on). Similarly, the blue and orange regions are analogous to obstacles a real-life path finding application might face.

In the iterations v/s convergence graph below, we can see a similar trend as that of the easy problem. That is, policy iteration converges faster than value iteration. In fact, the difference between the iterations required by both the algorithms is quite greater than that of the easy problem. This is expected because for the hard problem, the obstacles are quite strategically placed. So, just calculating value functions isn’t an optimal approach. In fact, the policy iteration method is a strategic approach to a strategic problem. Also, a similar uptick is observed in the policy iteration line plot as was seen for the easy problem. But here, the increment is more prominent that before. This is because there are many white cells that the agent encounters before finding the cells with varied rewards, and also there are much more cells with varied rewards than present in the easy problem. Thus, the uptick hits because of the first reason and it is prominent because of the second.

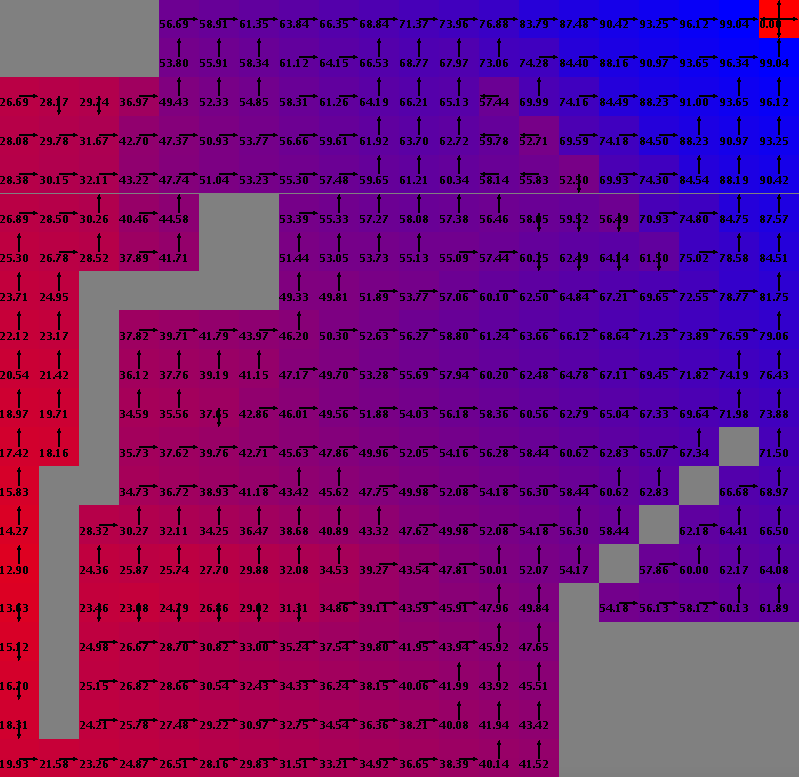
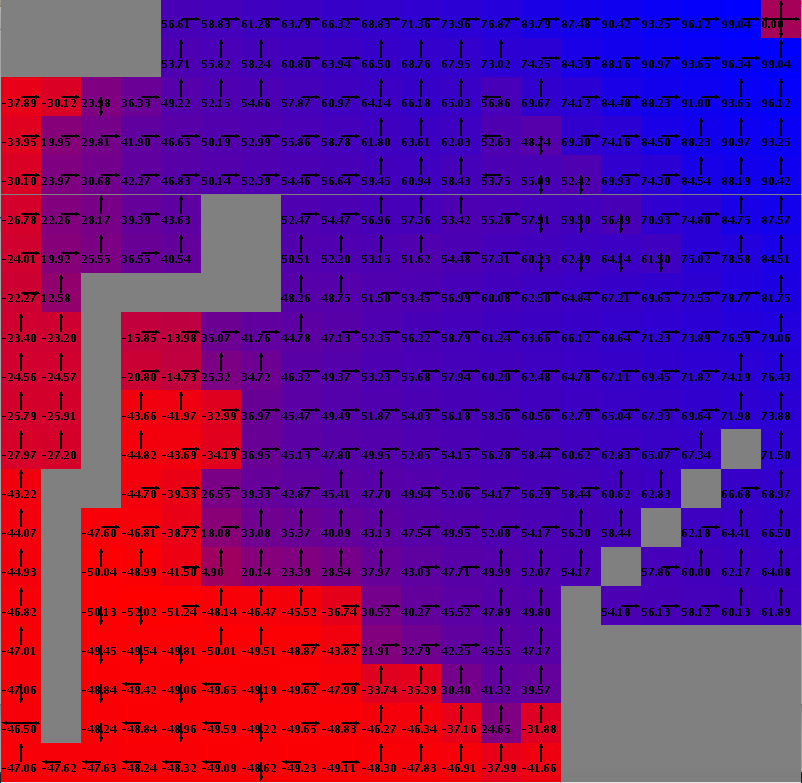
 

As for the iterations v/s rewards graph, the result is obvious as policy iteration would reach the optimal reward earlier than value iteration. (As it is a more strategic approach). And we can see that the value iteration line plot is flat for a greater number of iterations than before. This is due to the increase in the number of random value functions (because of the increase in the number of cells).

As for the iterations v/s time graph, the policy iteration method takes a longer amount of time required. This is because as we increase the number of cells, the number of linear equations that the algorithm has to solve also increase and this increment is more than that of the value iteration method.

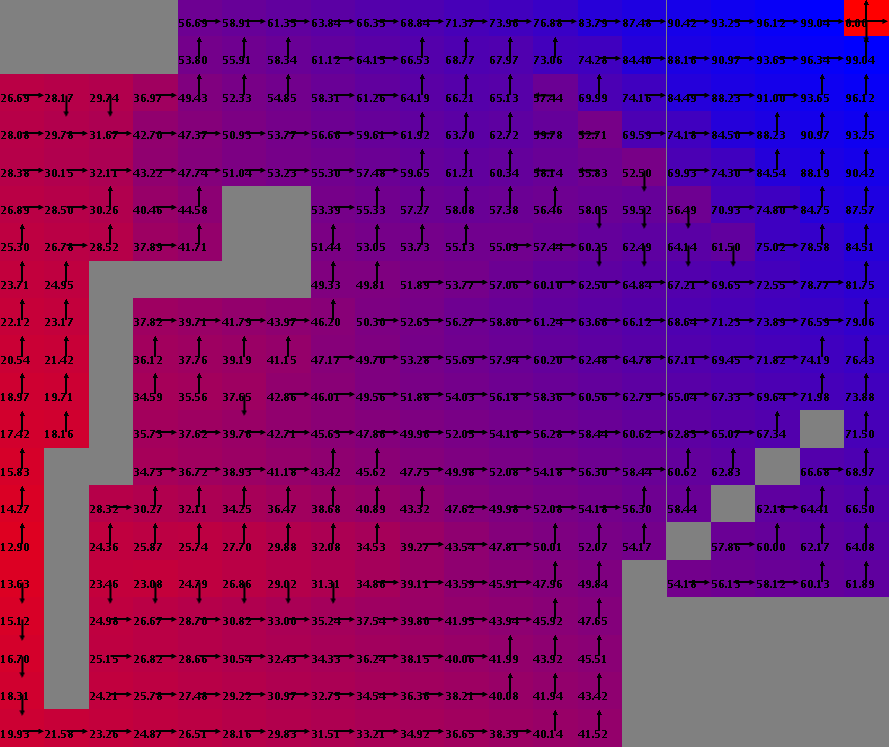


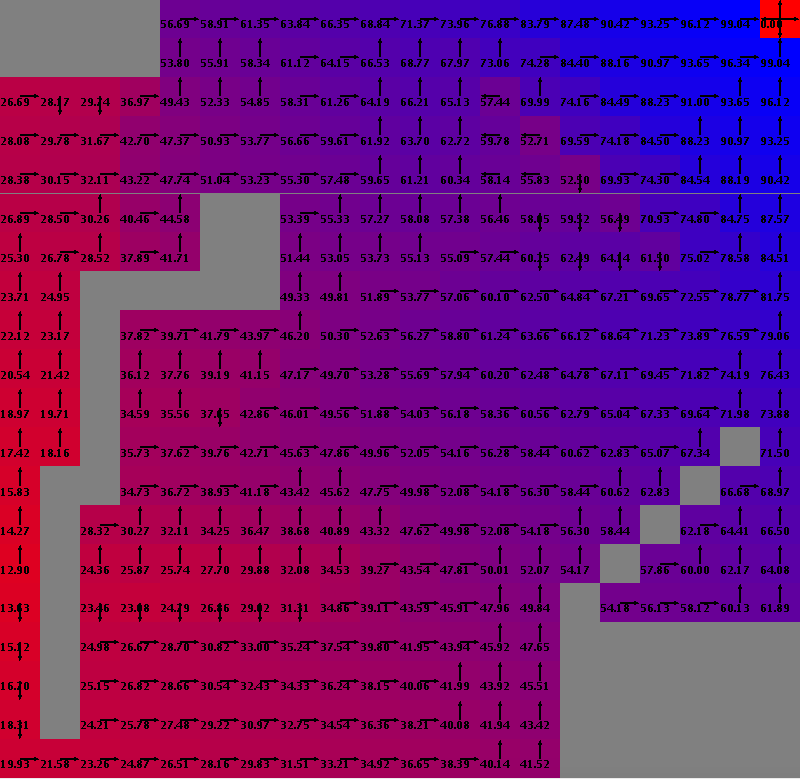
In the iterations v/steps chart, the policy iteration reaches the optimum no. of steps (minimum steps) faster. This is also because of the algorithm being strategic. One other thing to notice is that the policy iteration line is smooth but the value iteration line has a few upticks. Also the numbers of upticks are more than one. This is because there are several obstacles and as and when it encounters one, it gets an uptick, and then decreases when it factors in that obstacle in its calculation.



Value iteration (5th iteration) Value iteration (59th iteration)

In the grid above, we can see that the bottom right cells of the grid are dark blue in color (signifying that they have been visited more often) which makes sense because they are supposed to be a dead end. But once the algorithm figures that out, then the color for these cells change to purple (for the 59th iteration), which means that the agent should avoid that region. Also, we can see that the top right region is purple/red for both the 5th and the 59th iteration. There is a very little change in the color of these cells. First of all, the color makes sense because this region is nothing but walls and negative reward cells. Secondly, no change in the colors indicates that the algorithm has identified early on that the region needs to be avoided.





Policy iteration (5th iteration) Policy iteration (21st iteration)

As for policy iteration grids, we can see that they seem to have reached a way better state than value iteration for the 5th iteration itself. It correctly identifies the top left and the bottom right regions that are to be avoided by the agent. In addition, on a closer look, we can see that it has also factored in the red cells from the input grid and colored that stretch as purple/pink. Thus, it is taking into considering the fact that there is a negative reward region near the goal, and optimizing its results accordingly.

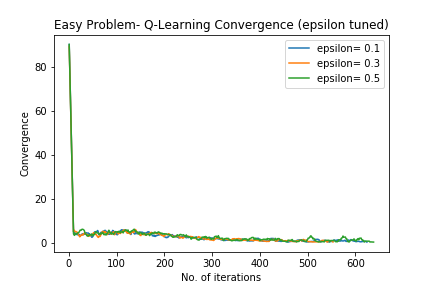
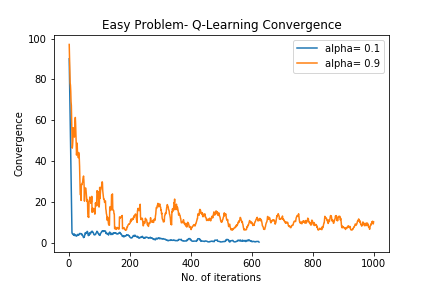
**Q-Learning (Section 4)**

This is a model-free learning method. Unlike policy and value iteration, in Q learning, the learner does not know what the effects of its actions would be on the environment. All the agent knows is the set of states and actions available. The main agenda of this learner at any given time is to maximize the expected reward of the future and base the current reward from this expected future reward.

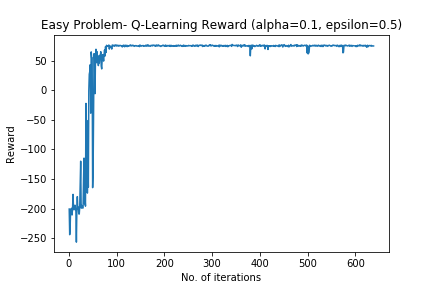
The reason I am interested in this algorithm is because it has tunable parameters such as epsilon (exploration v/s exploitation) and alpha (learning rate). Alpha signifies the learning rate and epsilon is a probability measure that indicates whether the agent will take a random action (explore) or stick to the given path. (*e* is a multiplier to the random action and *(1-e)* is a multiplier to the planned action. This way, we are not only sticking to the assumed best path but also exploring the surroundings to make sure that we aren’t missing out on any better path.

**Easy Problem**

In the below line plots, I have analyzed the variants in the convergence rate the values of alpha and epsilon. From the first graph, we can see that the value of 0.1 of alpha works better than 0.9. A lower learning rate signifies that the agent gives weightage to previous states along with the current states. And a higher learning rate means that it only cares about the current state. Thus we can see that a lower learning rate works better for our learner. This is because it is an easy problem and the old values itself can give us enough information to converge. We also observe that for both the alpha values that the line itself is highly variant that is not smooth. This has to do with the Q-learning approach which is a model-free approach and thus does not have enough information to create a smooth line and instead has variation at every step.

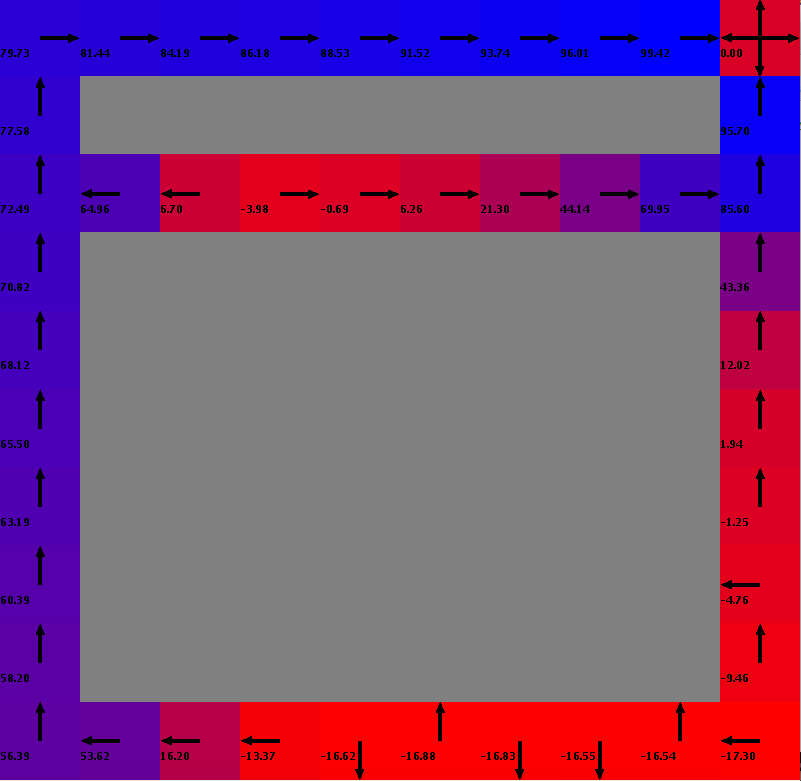


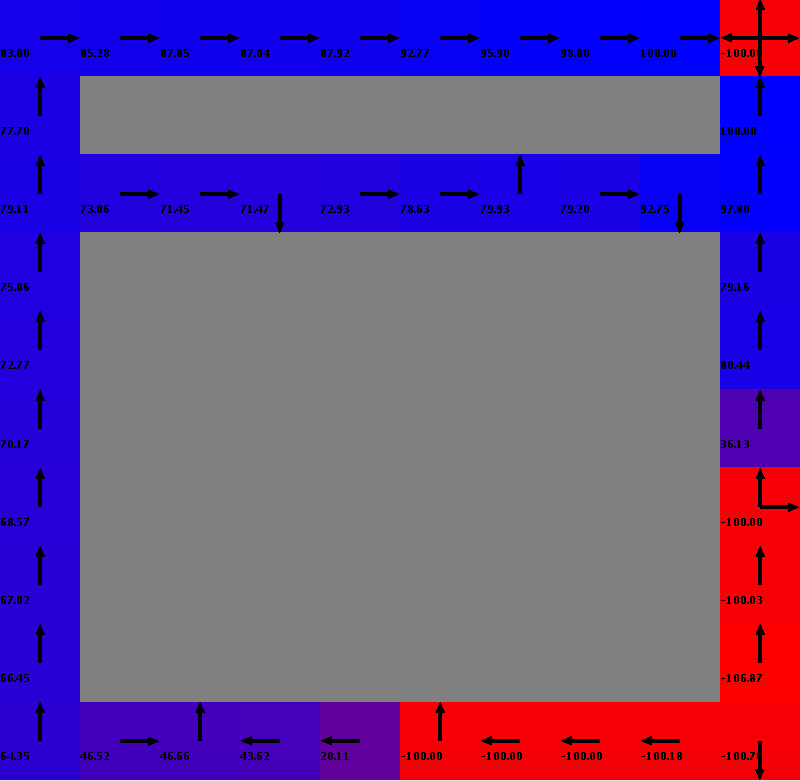
For the second plot, the graph remains the same for all the three epsilon values. This is pretty much expected because the problem we have taken has three equally short paths to the goal and increasing or decreasing the randomness of the actions will not affect the performance of the overall learner. Basically, most of the grid is covered during the search of the optimal solution and thus there is not much room for exploration.



For this third graph, I have selected alpha as 0.1 (because it gave better results) and epsilon as 0.5 (because it gives slightly better results than the other values, although taking another epsilon will not make a huge difference to this graph). We can see that the reward increases steeply and then stabilizes, indicating that the learner has converged. Also, the line is not smooth (as is the case for every Q-learner).

Below are the grids that I obtained after running the Q-learning algorithm. We can see that all the paths from the start to the goal are equally viable yet the grid obtained after the 50th iteration does not reflect that. The learner takes significantly higher number of iterations than the previous algorithms to converge. During convergence (i.e. at the 1000th iteration) the learner has encapsulated the grid to a good extent but still does not outperform the earlier approaches. This indicates that for a relatively simple path, the model based algorithms like value and policy iteration perform better than Q-learning.

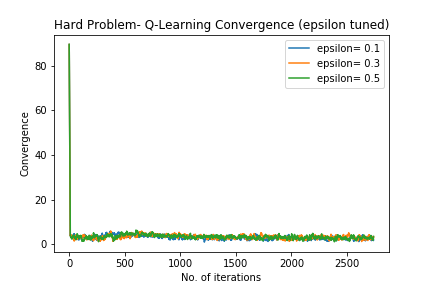
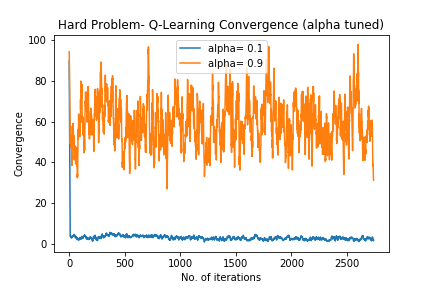
****



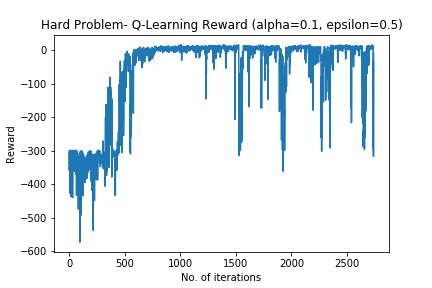
Q-Learning iteration (50th iteration) Q-Learning iteration (1000th iteration)

**Hard Problem**

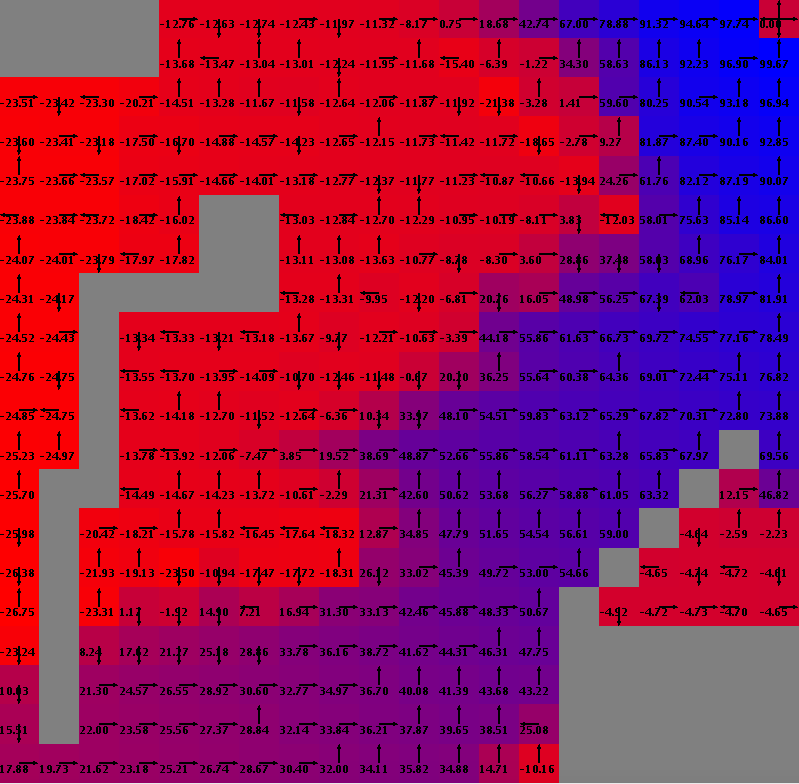
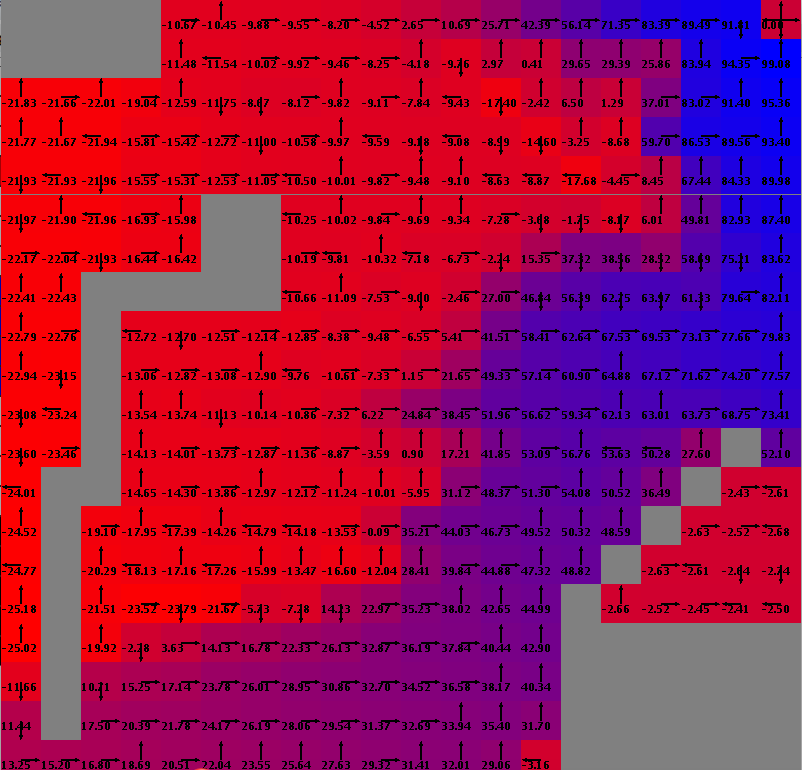
After analyzing the Q-learning parameters for the hard problem (i.e. 20x20 grid), I found similar results as that of the easy problem. A lower value of alpha is better. This is because the obstacles are strategically placed (at the corners or near the goal) and keeping track of the previous states result helps the algorithm converge faster. Also, the difference between the convergence of alpha = 0.1 and alpha = 0.9 is way higher than it was for the easy problem.



As for the epsilon values, the change isn’t that much. This is because taking random steps and exploring the surrounding area will lead the agent to the same path sooner or later because the obstacles cover a greater part of the grid.



For this third graph, I have selected alpha as 0.1 (because it gave better results) and epsilon as 0.5 (because it gives slightly better results than the other values). We can see that the reward has a flat line initially and then a steep increase. This is different from that of the easy problem (which had a steep increase initially itself). This is because this is a larger grid (with 4 times as many cells as the previous problem) and different kinds of obstacles (i.e. different kinds of rewards).



Q-Learning iteration (50th iteration) Q-Learning iteration (2740th iteration)

Above are the grids that I obtained after running the Q-learning algorithm. We can see that all the paths in the top left region aren’t viable as compared to the bottom left paths. The learner takes significantly higher number of iterations than the previous algorithms to converge. During convergence (i.e. at the 2740th iteration) the learner has encapsulated the grid to a good extent and slightly outperforms the earlier approaches (i.e. creates a clear distinction between the paths to choose and not to choose). This indicates that for a relatively complicated path, the model-free algorithms like Q-learning may perform better than model-based algorithms like value and policy iteration.

**Summary (Comparison between the three approaches)**

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria for comparison | Value Iteration | Policy Iteration | Q-Learning |
| Convergence | Converges with more number of iterations then policy iteration and less than q-learning. This is because it is a model-based but non-strategic approach. | Converges with the least number of iterations as it is a model-based strategic approach. | Converges with the greatest number of iterations as it is a model-free approach. |
| Rewards *(All three reach the optimum reward)* | Reaches the highest number of rewards with more iterations than policy iteration but less than q learning. | Reaches the highest number of rewards with the least number of iterations. | Reaches the highest number of rewards with the greatest number of iterations. |
| Time | Takes a similar amount of time as Policy iteration but less than Q-Learning. | Takes a similar amount of time as Value iteration but less than Q-Learning. | Takes the greatest amount of time. |
| Steps | The number of steps from start to goal is similar to that of Policy iteration but less than Q-learning. | The number of steps from start to goal is similar to that of Value iteration but less than Q-learning. | Takes the highest number of steps to reach from start to goal. |
| Grid at the time of convergence | Finds the optimal policy for each state | Finds the optimal policy value for each state | Policy values for each state aren’t as good as that of the model-based approaches. |

**References**

[1] *(Used to learn the concepts but did not take any statement from the article)* Deep Reinforcement Learning Demysitifed (Episode 2) — Policy Iteration, Value Iteration and Q-learning, <https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa>, Accessed April 14, 2019.

[2] GitHub- Jonathan Tay, <https://github.com/JonathanTay/CS-7641-assignment-4>, Accessed April 12, 2019.

[3] Brown-UMBC Reinforcement Learning and Planning (BURLAP) java code library, <http://burlap.cs.brown.edu/>, Accessed April 12, 2019.