Markov Decision Processes

Darsh Thakkar (dthakkar33)

Code and Readme.txt: https://github.com/darshthakkar/Markov-Decision-Processes-and-Reinforcement-Learning

Introduction

Two different Markov Decision Process(MDP) problems are analyzed with various learning techniques and discussed in this report. The framework utilized to perform these experiments is the BURLAP framework [1] which contains helper functions and modules, designed to implement and test reinforcement learning and MDP. These techniques are compared with each other and their behavior is analyzed, i.e. what happens and why? Part 1 discusses about two interesting MDP problem. Part 2 discusses about the implementation and analysis of two popular algorithms used to solve MDP problems – Value iteration and Policy iteration and how they behave while solving the two problems discussed in Part 1. Part 3 of the report analyzes how another Reinforcement Learning algorithm (in this case Q Learning) solves the same two problems discussed in Part 1.

Markov Decision Process

A Markov decision process (MDP) is a discrete time stochastic control process which means that it divides the state space into discrete states and gives a mathematical foundation to model decision making for an agent in that environment [2]. The environment referred here has 'S' states from which an agent can pick an action 'A' from predefined list of actions in such a way that the reward "R (S, A)" is maximized for that particular state-action pair. The transition function decides which action to take in a given state. The solution to a MDP problem is to determine something called as a "Policy" which is denoted by 'π' and π (S) or Policy (S) denotes which action the agent should take when it is in state 'S'.

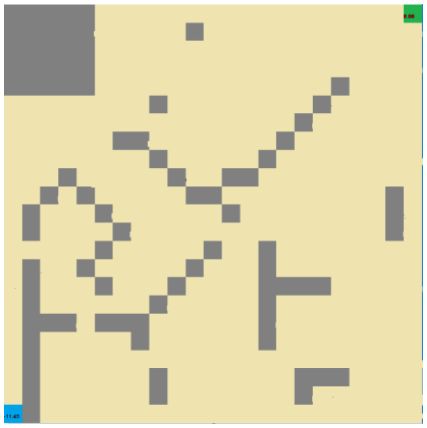
Part 1 – Two MDP Problems

Two MDP problems I have selected are name as Easy Grid and Hard Grid problem. The goal in both the problems is for the agent to start from the blue square and reach the goal in the green square. Both grid worlds possess various different obstacles, states, transitions and rewards. The solution also requires that the agent follows the optimal path and provides optimal policy as there can be more than one way to reach from start to goal. The dimensions of the grids are 8x8 and 23x23 for easy and hard grid problems respectively.

These problems are interesting as they relate to real life scenarios, for instance it is very close to navigation of paths from one position to another, finding safest path to reach from one point to another (reward comes to play in this case) which can also be implemented for navigation around the Home Park area near campus if relevant data is available. It also resolves to the famous problem of navigation for a robot in a maze as well as navigating between the heavy traffic of Atlanta and choosing the best path to save time (saving time is reward in this case).

Though all the problems mentioned are complex, their Machine Learning concept foundation can be tackled using the easy and hard grid problems.

The images shown below are a representation of the grids utilized in the analyses. Grey area represents walls in the grid world, blue square represents the starting position and green square the ending position.



Follow are the real values utilized for making the grid world, easy and hard grid respectively. 1 represents wall, 0 as valid path and the values which are negative are also valid path but with negative rewards.

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

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

Part 2 - Value and Policy Iteration on MDP Problems

Value Iteration

“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.” [3] Value Iteration is based on Bellman Equation. It calculates the value of a state based on the value from the neighboring 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

Policy iteration is another method for computing optimal MDP Policy. It starts with an arbitrary policy and iteratively improves it. 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.

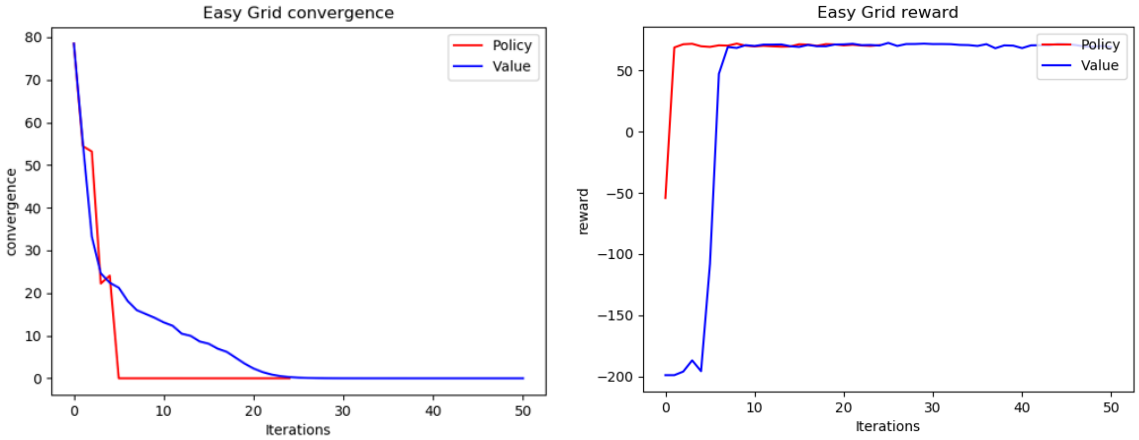
Both the algorithms are allowed to run until the convergence/delta value goes below the threshold, that is the value difference between consecutive states becomes too small.

Easy Grid Problem

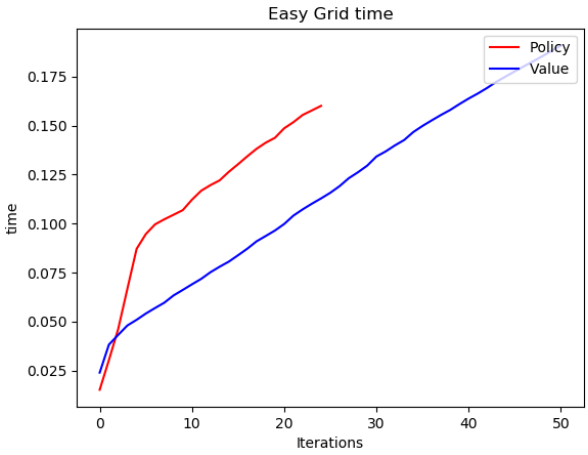
When both algorithms are applied to this problem, they iterate relatively very fast to provide an optimal path because the number of states are not that many. The output metrics that the algorithm provides are number of steps taken by the agent, convergence, reward and time take. On comparison of different discount values it was found out that

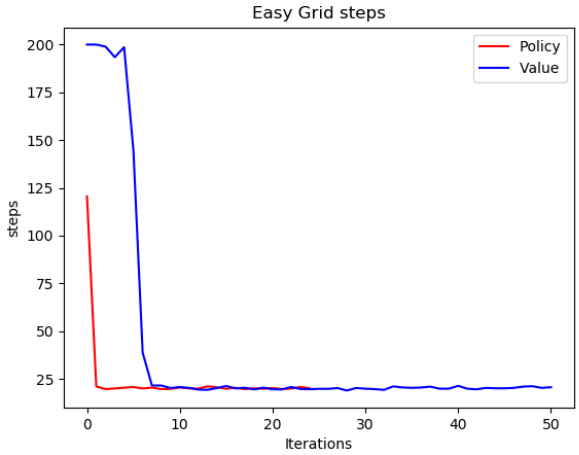
Following is the comparison of convergence values of the two algorithms on the easy grid problem. From the graph it is evident that policy iteration converged much faster compared to value iteration, ultimately value iteration also reaches convergence. Convergence is considered one of the most important metric in comparing performances of the algorithm and policy iteration seems to be the better way to go for this. This is probably because policy iteration performs two steps – policy evaluation as well as policy improvement in each iteration, while value iteration directly applies Bellman operator to converge it to an optimal value.

Next up, the cumulative reward between both the algorithms is compared which is just a bit above 50 and this is also obtained by policy iteration faster.



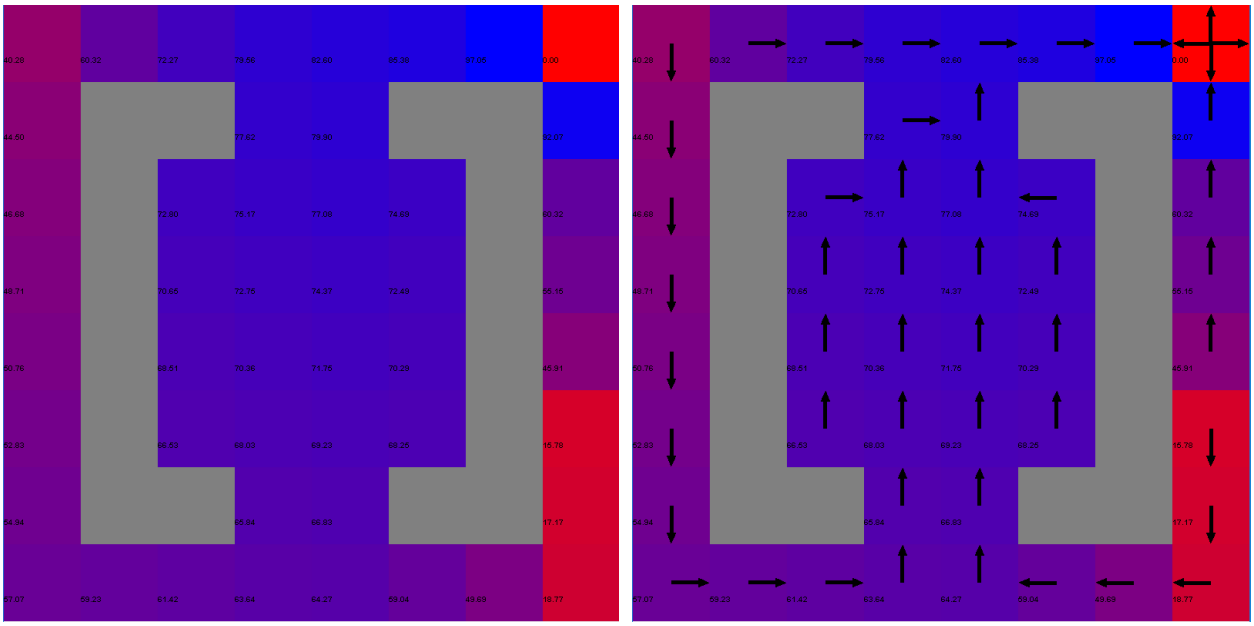
On comparing the time and number of steps taken by the agent in both the algorithms is compared. This shows that policy iteration takes more time per iteration compared to value iteration but also on the contrary it takes less number of steps than value iteration. Thus, this seems to point out that policy iteration is learning more per step which explains the more time that it takes.





The policy learned by policy iteration and corresponding q values are visualized and shown in the following picture. The color is encoded with q-values, with red being a lower value and blue being a higher value of q. The arrows in each square show which direction the agent should take when in that state/square. There wasn't any change in the visualization between 5 and 25 iterations since for the easy grid the algorithm arrived at the solution very quickly and thus only results after 25 iterations are shown.

After 25 iterations:

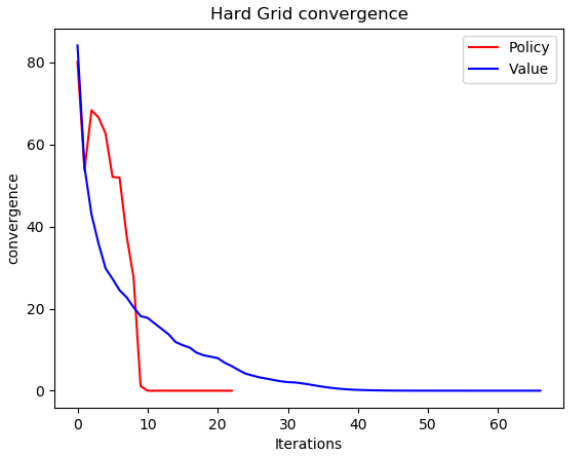


Hard Grid Problem

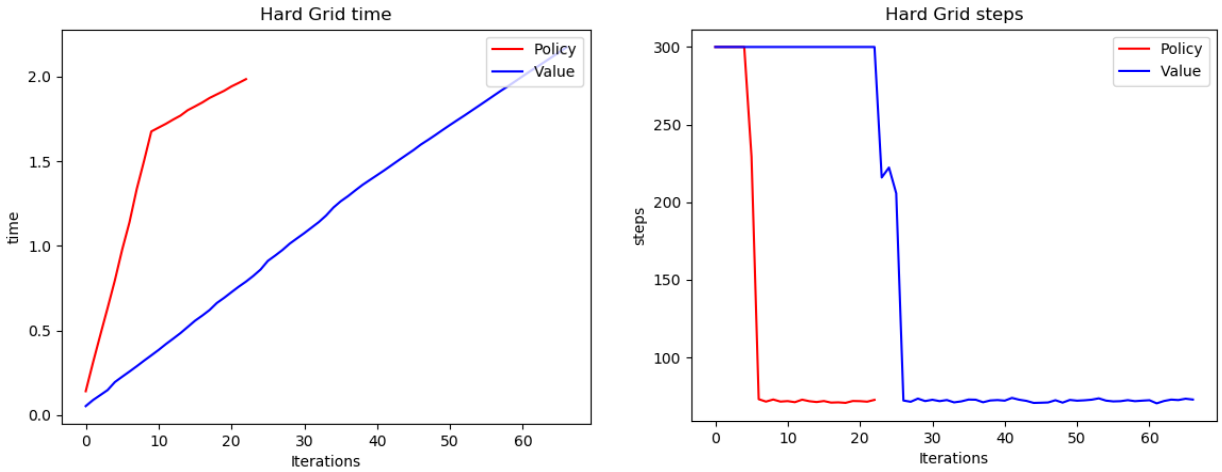
On applying policy and value iteration algorithms on hard grid problem, they take significantly much more time compared to the previous case as the number of states have increased. For this problem also both the algorithms are able to find out the optimal path that the agent should take along with cumulative reward, time, number of steps and convergence. The end point is when the algorithms converge that is the difference between consecutive iterations is zero or very small.

The graph below, compares convergence between both the algorithms. Inference from this graph is that initially the convergence for policy iteration is increasing and then decreasing, while at the same time for value iteration it is constantly decreasing. This is similar to the behavior seen in the easy grid solution, however, in this case it is very evident that the convergence difference for policy iteration is much higher initially and then it suddenly falls down after 10 iterations. This behavior can be explained by the policy evaluation and improvement in each iteration, meaning the arbitrary policies chosen at the start were very wrong but the algorithm learned the correct policies quickly and converged.

Next, comparing the cumulative reward between both algorithms shows that they achieve the cumulative reward, which is around 20 in this case compare to ~50 in previous case, eventually. However, because of the same reasoning as provided earlier, the policy iteration achieves this faster in around 10 iterations that is when it just reaches convergence. One thing to notice here is that when both algorithms initially start, reward for policy iteration after 1st iteration itself becomes higher than value iteration.

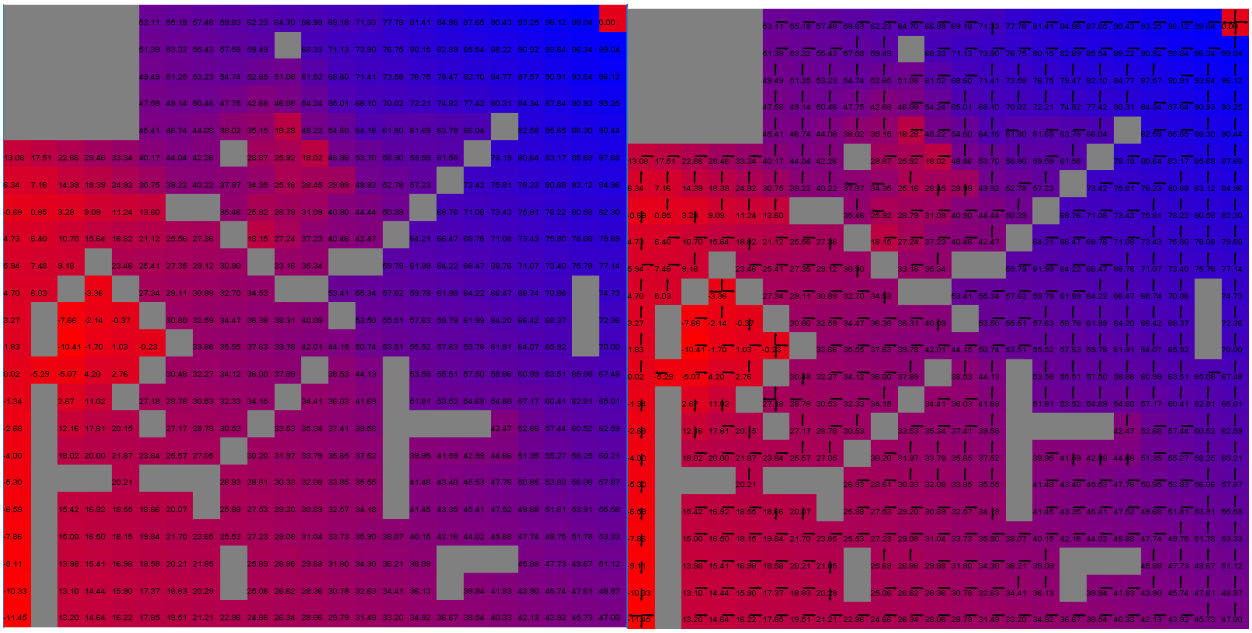


Lastly, comparing the time and number of steps taken for both algorithms are compared. The result here is similar to the one observed in easy grid problem, that is, policy iteration takes more time compared to value iteration to complete at the same time the number of iterations are less. Also, as seen previously, number of steps initially taken are same for both but as policy iteration reaches conversion, number of steps also reduce.

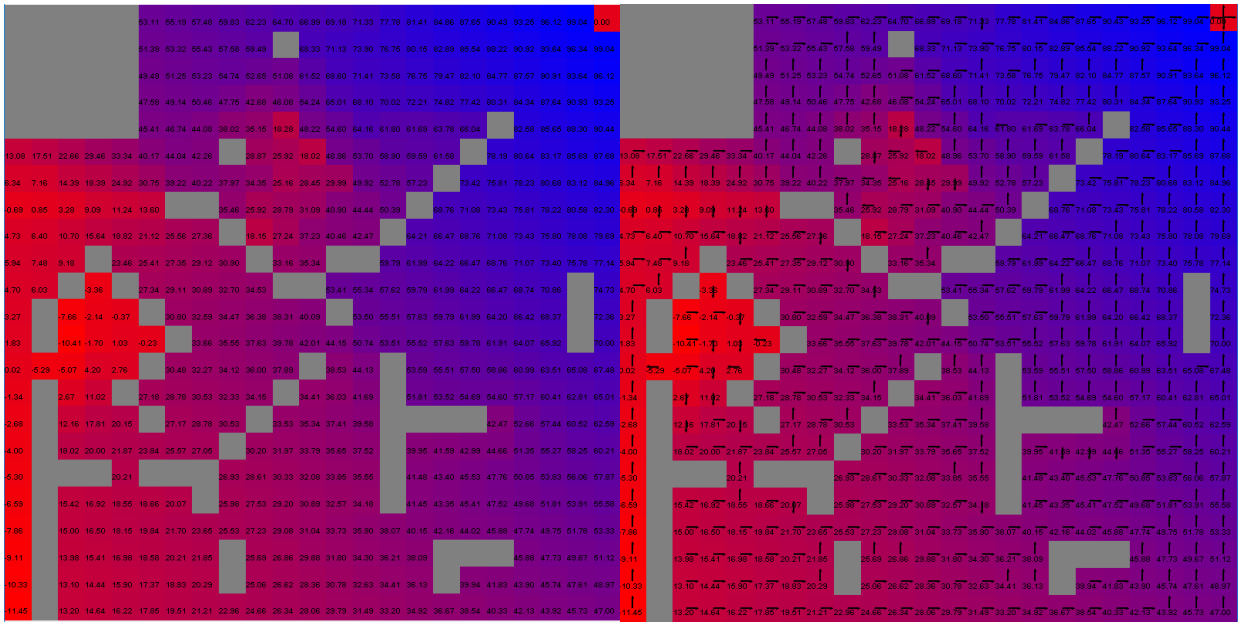


Below, is the comparison of 2 visualized mappings of hard grid problems, one after 5 iterations and another after 23 iterations. The legend for these mappings is same as in the previous case.

After 5 iterations:



After 23 iterations:



Comparison between Value and Policy Iteration

- Overall, in these two problems, policy iteration turns out to be better.
- In terms of computational time, policy iteration takes more time compared to value iteration. This is because it finds optimal policy for each state, which turns out to be computationally costlier when compare to value iteration.
- Policy iteration reaches convergence quickly as in a sense it is able to learn quickly even if the initial arbitrary policy is very bad.
- Policy iteration does not lead to a non-optimal local optimum as it can improve an action for a state without affecting another state, wherein updating values can affect other states.

Part 3 – Reinforcement Learning Algorithm (Q Learning)

Q Learning is a model free reinforcement learning algorithm where Q values are updated using the Q-function and the policy is learned without any prior training based on those Q values. [4] The updated Bellman equation on which Q-learning works is as follows:

$$Q(s,a) := Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

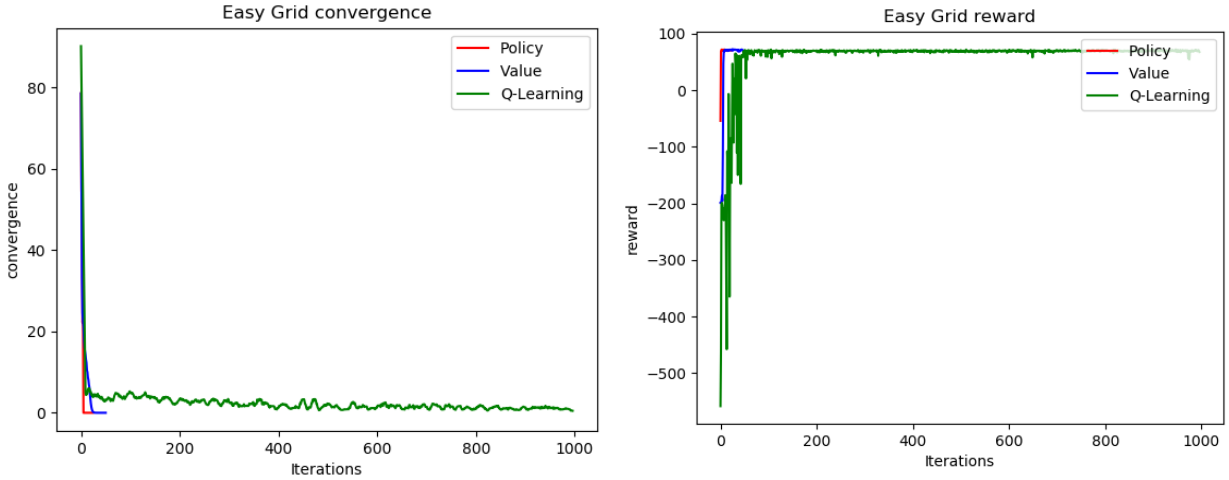
Here, Q(s,a) is the q value corresponding to action a and state s and s' is the next state. The following section compares the Q learning algorithm with policy and value iteration algorithms for the same two MDP problems discussed earlier – Easy and Hard Grid Problem.

The algorithm is run with different hyper parameter values of learning rate and epsilon to determine the best possible combination and the metric used to select the hypermeter values

is by looking at how fast the algorithm converges. The final combination selected was LR=0.1 and Epsilon=0.5. There is an observation here that the algorithm performs well with less learning rate, thus it prefers having more immediate rewards compared to future rewards and therefore has a greedy approach. A higher epsilon value signifies that the algorithm will try more to take random actions and explore.

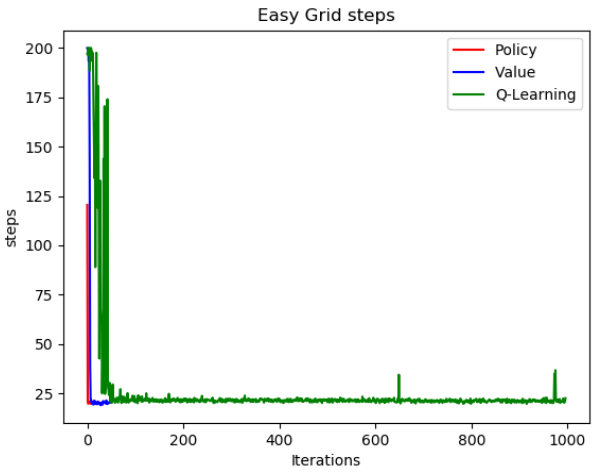
Easy Grid Problem

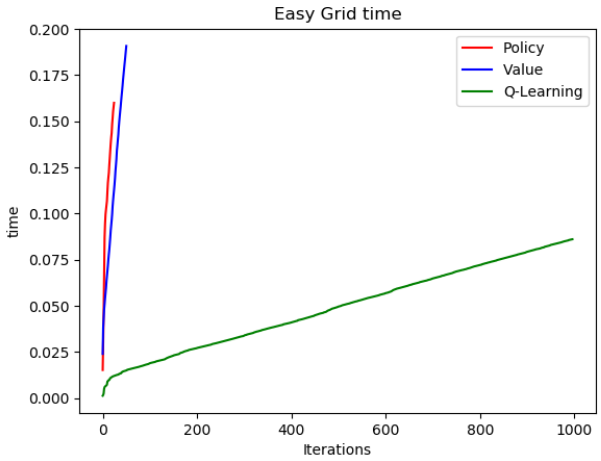
Q Learning algorithm is compared using the same metrics as policy and value iteration, i.e. convergence, reward, time and steps taken. Here, Q leaning is plotted alongside of the previous two algorithms to allow effective comparison. In case of convergence, the Q Learning completely converges after 1000 iterations while the other two converge around 50 iterations.



The graph above makes it evident that Q learning takes a lot more time to converge, however the convergence value just stays above the threshold most of the time after 50-60 iterations. The probable reason behind this slow convergence is the low learning rate of the algorithm and also as it's an easy problem, the other two algorithms converge even faster. Another reason that why Q learning takes more time can be because its exploring all possible options until it finds the optimum. In case of the cumulative reward metric, the Q learning algorithm behaves similar to value and policy iteration algorithms.

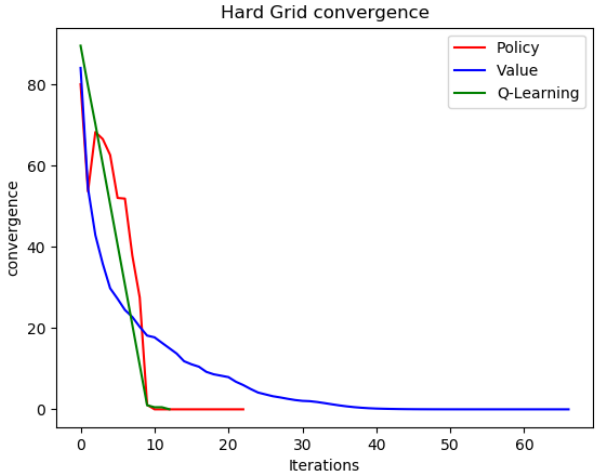
The graphs below show comparison of Q learning with other two algorithms in terms of total steps taken and time. All algorithms behave very similar to each other for number of steps taken, Q learning just takes few steps more to reach the minimum number of steps. For time graph, it's pretty evident that Q learning takes significantly less time for initial and even overall iterations when compare to the other two algorithms.

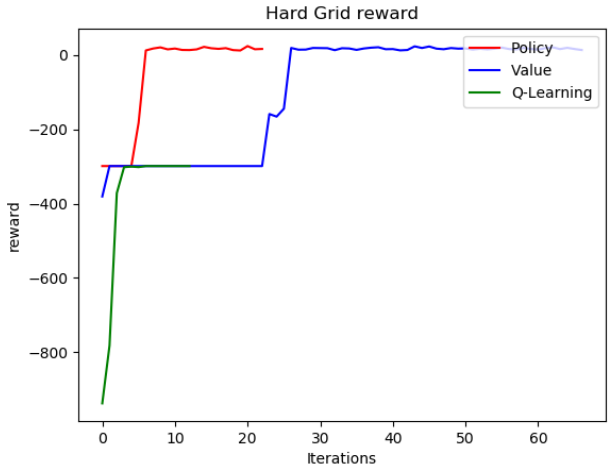




Hard Grid Problem

Same pattern as above was followed for the Hard grid problem. The graph produced for convergence and reward are given below. In this case the Q learning algorithm converges very soon, compared to the other two and therefore its line also disappears after just few more than 10 iterations.

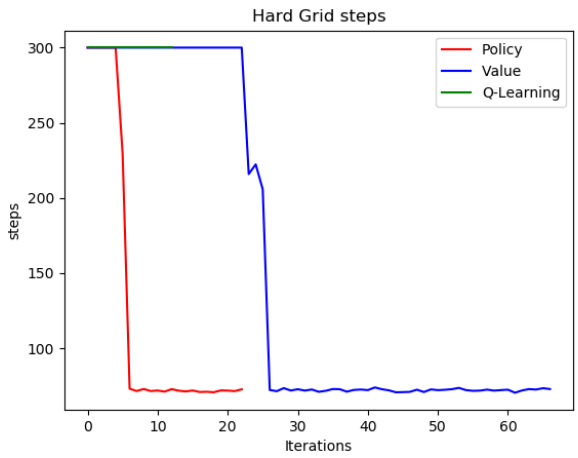


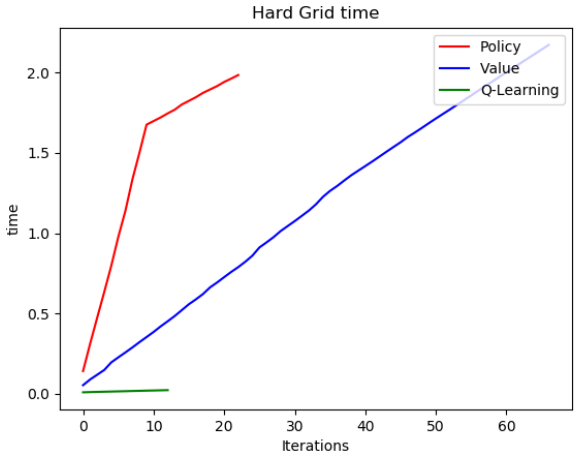


For convergence graph, the main change from previous scenario is that the Q learning converges very fast, almost similar to policy iteration but here the convergence does not go in the wrong direction initially, as in the case of policy iteration.

On comparing the reward gained from all three algorithms, there is a change from the easy grid problem. Here, the Q learning algorithm was not able to produce high reward or comparable reward to other two algorithms. This is related to convergence here, since the algorithm converges near 10 iterations, it does not allow to iterate more for better reward gain. The reward actually stabilizes around -300 for Q learning algorithm here.

The graphs below are for comparing Q learning with other algorithms in terms of number of steps taken by the agent and time.





From the graph above we can infer that forr the Q learning algorithm the computational time required is much less compared to valu and policy iteration. Moreover, for number of steps taken the algorithm behaves similar to Value iteration, except the steps are not reduced afterwards.

Overall, it seems like the Q learning algorithm does not wor will the Hard Grid problem. In a sense the convergence is achieved but the corresponding cumulative reward is low compared to the other two algorithms and also the number of steps are not reduced. Probably, there might be a change if convergence is not achieved so early and algorithm is able to iterate further. This may also be the impact of dimensionality and thus it requires more iterations.

Conclusion

- Q Learning seems to be the fastest algorithm; however, it doesn't work efficiently when dimensionality increased.
- Immediate reward has more effect in Q learning than long term reward.
- Policy iteration is much better than Value iteration and it is able to avoid local optima.
- Initial actions/states affect the algorithms ability to overcome local optima.

References

[1] http://burlap.cs.brown.edu/ Brown-UMBC Reinforcement Learning and Planning (BURLAP) java code library.
[2] https://en.wikipedia.org/wiki/Markov_decision_process Wikipedia, Markov Decision Process
[3] https://artint.info/html/ArtInt_227.html Artificial Intelligence: Foundations of Computational Agents, second edition, Cambridge University Press, David Poole, Alan Mackworth.
[4] https://en.wikipedia.org/wiki/Q-learning Wikipedia, Q-Learning
[5] Code Source: https://github.com/JonathanTay/CS-7641-assignment-4, CS-7641-assignment-4 by Jonathan Tay