Assignment 4: Markov Decision Processes

Ahmad Aldabbagh

November 24, 2016

1. Introduction

This assignment has several parts that explore Markov decision processes (MDP)s. The first part is related to coming up with two MDP problems and solving them using both value iteration as well as policy iteration and comparing both algorithms. The second part is to use a third algorithm to explore the same problems and see how the performance have changed in the new algorithm. The third algorithm of choice is q-learning.

2. Problems of Choice

The problems I decided to explore are grid world problems. These problems are essentially discrete maze problems with an (m by n) number of cells, where m is the number of rows and n is the number of columns. At any moment, the agent stands on one of the cells and is bounded by the boundaries of the grid (cannot stand beyond n or m). Also, some of the cells in the grid might contain a wall which block the agents path or in other words cannot stand on or jump over that blocked cell.

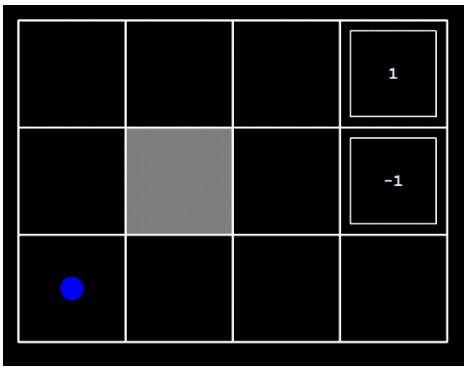
Furthermore, the agent can take four actions which are to move north, south, east or west. These actions are not always executed perfectly as sometimes the presence of noise prevents the taken action to be completed correctly. Typically, the selected action is executed with probability of (P). However, there is a (1-P)/2 probability to move right and (1-P)/2 probability to move left perpendicular to the action taken. Also, in the case where the agent took an action to move towards the block and the block is present in one of the cells next to the agent, it will simply stay in its place.

Each time the agent takes a step, it receives a small step reward (could also be negative or zero). Also, there could be one or more termination cells where the agent receives a large reward that could be either positive or negative. The main goal of this problem is to maximize the sum of the rewards until the agent reaches one of the termination points which in turn leads to finding the optimum policy for each state.

The actual size of the first problem is 3x4 where it contains 12 states. Whereas the size of the second problem is 14x14, containing 196 states. What makes these problems interesting is that they clearly capture the main components that comprise MDP problems and makes it easy to visualize the problem. This in turn helps in understanding the behavior of various solving algorithms and how changing the algorithm's parameters, changes the behavior or solution of the problem. In the following sections both problems will be analyzed using three different algorithms and a discussion of how both problems are different will be covered in more detail. All problems were implemented using UC Berkeley's CS 188 project files [1].

2.1 Problem 1: a small grid world problem

The first problem consists of 12 states, two termination cells and one block as shown in the figure. The gray square is the block and (+1), (-1) squares are the termination point with those values as the rewards for them. Also, the start state is the bottom left square with the blue dot that represents our agent. The introduced noise in the problem is 0.2 with a discount factor (gamma) of 0.9 and a living reward of 0.

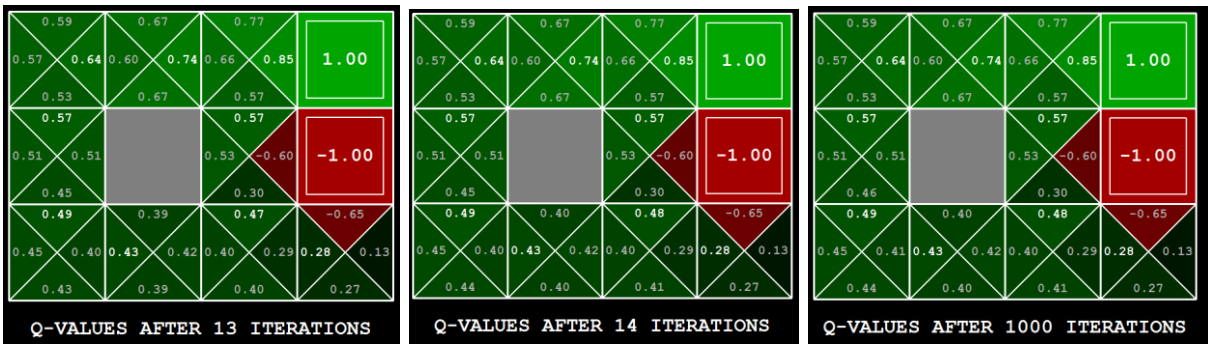


The first algorithm used to find the optimal policy is **value iteration** which recursively calculates the value function to find the optimal value function so that the optimal policy can be evaluated. The value function (V) is given by the following function:

$$V(s) \leftarrow \left[\max(a)\,\gamma \sum_{s'} P(s'|s,a)V(s')\right] + R(s)$$

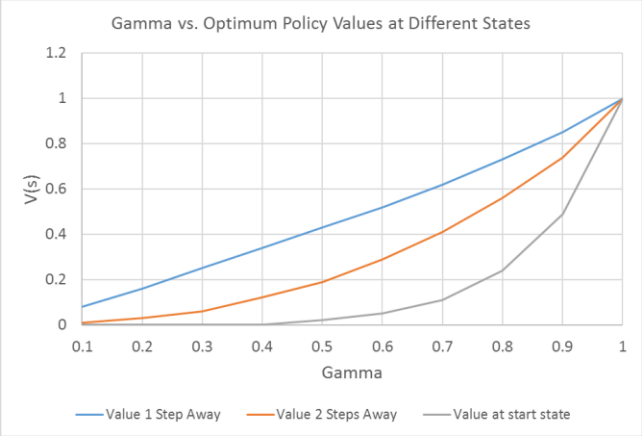
The value of state (s) is estimated recursively based on the value of successor state (s') where each action (a) is not deterministic so we need to go through all possible (s') and weigh each with the associated probability and this has a discount factor (gamma). Since there are different actions that can be taken at a given state, we need to take the maximum value of all possible actions. Finally, we add the reward (R) of the state (s).

To converge, I would need to iterate a number of iterations that is greater than or equal to the number of steps from the start state to either termination states. After running value iteration with different iterations, it was noticed that it takes 14 iterations for the values to converge to their max. The figure below shows the values at different iterations and how it stabilizes after 14 iterations where the best policy is highlighted with the brightest white color and has the highest value towards the direction of the best policy.



At 13 iterations, some values in some states haven't maxed out. This happens after 14 iterations.

To see gamma's role in our results, we will vary the value of gamma to see how that affects our results. It was noted that as gamma gets smaller and smaller, the expected values of states start getting smaller and smaller until they become equal to zero. The following graph shows value of the optimum policy of cells (states) several steps away from the +1 termination point vs. the value of gamma.



As shown in the graph above, when the state is close to the termination point, the value has a semi linear relationship with gamma. However, as we move to farther states the relation starts to decay exponentially. This is quite expected since the discount is recursively compounded when calculating V(s) and what this essentially means that for small gamma or discount values, we prefer immediate rewards rather than later rewards. To extend this idea even more, it is expected that for problems with very large number of states that the values far away will have small values; so to avoid having states with zero values, either gamma should be sufficiently large or the final termination state reward should be sufficiently large to be backed-up to far states. This gamma however is important to help the algorithm converge.

Upon inspecting the optimum policies, you notice that the policies that are near the -1 termination point could lead to falling in the -1 state. The reason is that the +1 reward is large enough to make these actions worth taking. For example, the right bottom state has an optimum policy to take action west, even if that action makes it prone to fall in the -1 point. This policy would change if the value of (1) was reduced or if for example negative rewards were introduced to the problem (as shown in the figure below).

As can be seen above, after introducing a state reward of -0.5, some policies have changed. Policies, that are far enough from +1 want to end the problem as fast as possible rather than

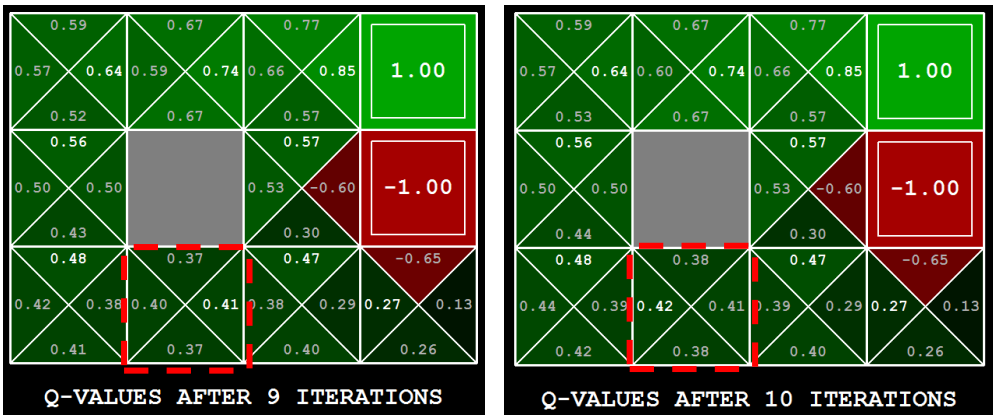
terminate at +1 and get penalized for the extra steps. In this case, only 2 steps are equivalent to the -1 termination state that was avoided in the old setup.

The second algorithm that was used is **policy iteration**. This algorithm builds on the ideas that are used in value iteration but we use the policy form of Bellman's equation:

$$V_{t+1}^{\pi}(s) \leftarrow \left[\gamma \sum_{s'} P(s'|s,a) V_t^{\pi}(s')\right] + R(s,\pi(s))$$

The value is calculated for some policy until convergence. The policy is then updated in each step by just looking one step to the future (s') by using the converged utilities as future values. This is repeated until convergence.

By using policy iteration on our problem, the policy converges before the values. It converges after 10 iterations as shown below:



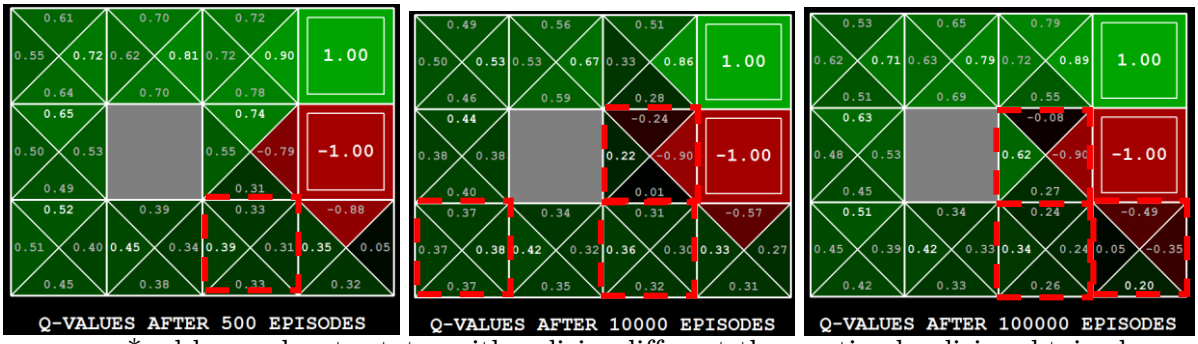
As seen above, most states converge to the optimum policy after 9 iterations (as a matter of fact after 6 but showed 9 here to demonstrate convergence after 10 iterations). However, the highlighted state does not converge to the optimal policy until 10 iterations where it should be west instead of east. A key difference between this and value iteration, is that policy convergence does not guarantee value convergence but the good thing is that we do not really care about the values rather than actually finding the optimal policy or best action to take. Another point worth mentioning is that when the policy converges, we do not know how good it is until we look at V(s) and this could be important to us if we were looking for an optimal policy that is significantly larger than other possible policies.

An interesting relation between convergence in policy iteration and the value of gamma is that as gamma gets smaller, less number of iterations are required for convergence; this however is bounded by the least number of steps between the start state and the terminating state with highest reward (which is 5 in this example). For example, when gamma is equal to 0.4, only 6 iterations are required to converge to the optimal policy instead of 10.

We see here that policy iteration converged in less iterations than value iteration (10 vs. 14) and convergence here was faster than value iteration.

The third algorithm is **q-learning** which differs from the two previous solutions in that it is an online approach where we do not know what actions do so we will need to do actions that will lead to -1 in order to know that this action in this state is not good. Whereas, in both value iteration and policy iteration we already know that we want to avoid -1 so we will not attempt an action that goes towards the -1 termination state.

For epsilon equals 0.3 and a learning rate of 0.5 after various runs with different number of episodes, we got the following q-values:

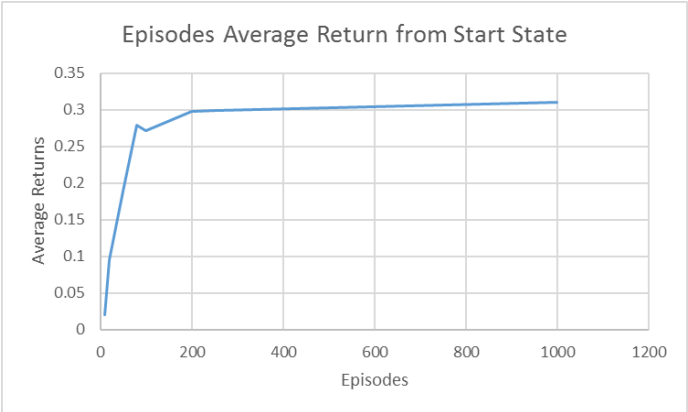


*red boxes denote states with policies different than optimal policies obtained using value iteration and policy iteration

It can be seen that after 500 episodes, we get policies almost identical to what we got in both value and policy iteration. However, by looking at higher episodes, we see that the policies change a bit. This is quite understandable, since q-learning converges if each action is executed infinite times on an infinite run with a decay in alpha but this is not practical so we need a metric to decide whether the current policy is close enough to the optimal policy or not. Based on the figure above, examining similarities in policies to the optimal policies in value iteration and policy iteration does not seem to be a good criterion as these do not seem to be consistent or related to the number of episodes. For example, as shown in the figure above, highlighted in red, the 500 episode run is closer to the optimal policy than the 100k run.

A possible metric to look at the change in q-values where we observe when the change in the maximum q-value of states starts to become small and is not fluctuating a lot. This can simply be calculated by looking at the average of the changes of max q-values in states for various episodes. (i.e. average($\Delta s1, \Delta s2, \Delta s3 \ldots$) and plot it vs. the number of episodes). Another approach is to look at the average returns from the start state and use that as a metric for convergence. It was noted that after a while the average returns start converging and this is the metric I decided to employ here.

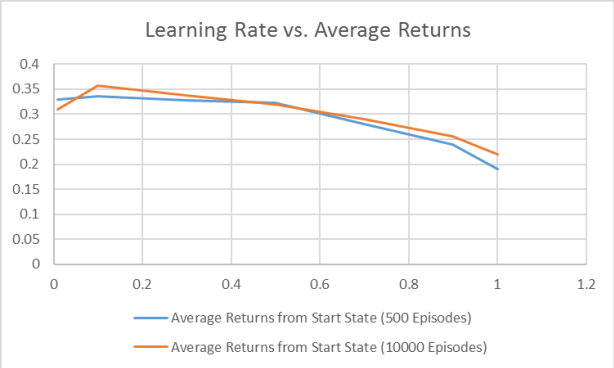
Each episode experiment is run several times and the output is averaged out. From the graph below, it looks like it is safe to assume that after 200 episodes, the average returns seem to somewhat stabilize and might be a possible way to decide to stop the episodes.

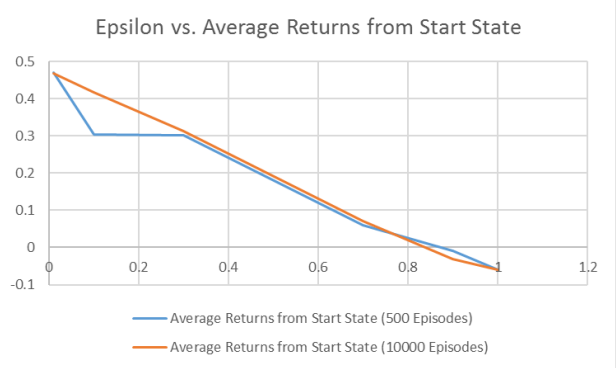


In general, after a fairly large number of episodes with respect to the number of (states*actions), the determined policies seem to be fairly acceptable since it is making sense of the environment and avoiding things that it should avoid without knowing that to begin with.

Another issue arises when choosing the parameters of the agent [epsilon and alpha (learning rate)]. These give rise to a trade-off between exploring and exploiting. The methods in [4,5] suggest a few techniques to improve learning such as: greedy strategies, randomized strategies and interval based techniques.

To try and visualize the effect of the learning rate, the learning rate and epsilon are varied and compared to the average returns from the start state at two runs (500 and 10000).





It was interesting to see that when both the learning rate and epsilon get larger, the learning becomes worse. In case of alpha (learning rate) it was somewhat expected since a high learning rate means that I am giving a high weight to the newly observed value vs. the old accumulated values and this newly observed value could be noisy.

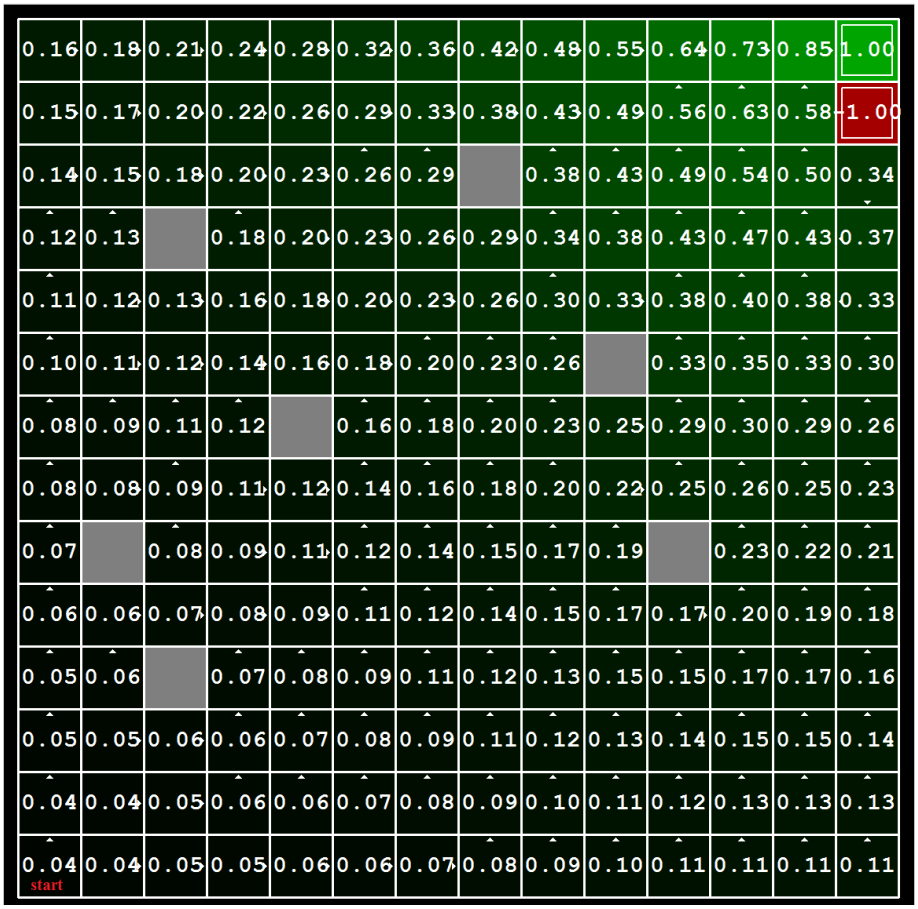
Learner	Iteration time(s)	Number of iterations until convergence
Value Iteration	0.00600	14
Policy Iteration	0.00599	10
Q-Learner	0.09100	200

It can be seen that both value iteration and policy iteration take around the same time to converge while q-learning takes a longer time due to the larger number of iterations.

2.1 Problem 2: a large grid world problem

In the previous sections the grid world problem has been described. Also, the main three algorithms have been mentioned in some depth along with some of the parameters affecting their behavior so I will not go in much detail here.

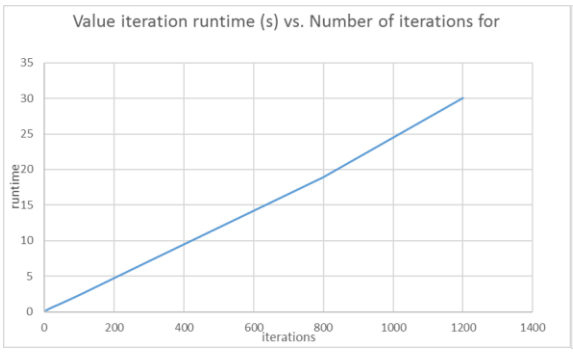
The second problem is also a grid world problem but with a larger number of states (196) instead of 12. All three algorithms have been applied to the problem and results give more insights about how each algorithm handles problems with varying number of states and how does scaling the problem affect performance, runtimes, etc.

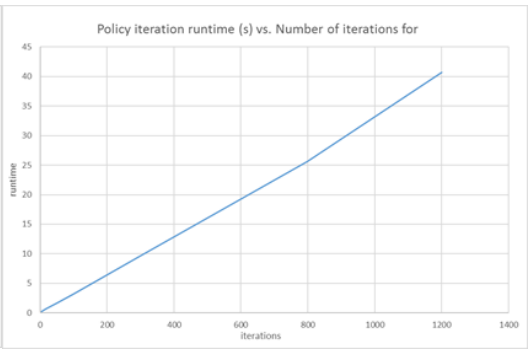


Upon running **value iteration** on the problem, it was found that it converges after 41 iterations (as shown above) based on the fact that the values stabilize so the number of iterations was approximately 1.578 the number of steps from start to end based on the optimum policy. This is smaller than the 2.8 ratio in the smaller problem but it is important to note that the iteration here takes would require approximately 267 times the number of operations required in each iteration in the small grid. Gamma was fixed at 0.9 and it was shown earlier how it gets compounded as the number of steps increase which obvious from the recursive nature of using bellman's equation in value iteration. Another thing noticed here, is that the value at the start state is really small due to the large number of steps and if the value of gamma is decreased even further, it would, surprisingly, require less iterations to converge since far away states become equal to zero and no changes take place as the number of iterations increase.

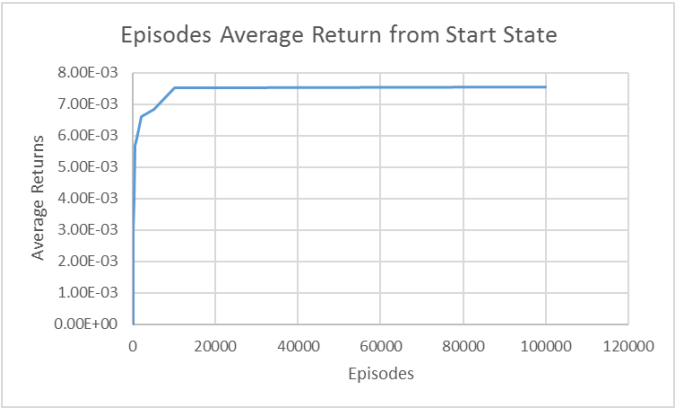
When **policy iteration** was run on the problem, it was found to converge after 28 iterations. This is approximately 1.037 times the number of steps between the start state and the termination compared to 1.28. Again, iterations in this problem still require more calculations than the smaller grid world due to the increased size of this problem. In both cases of policy iteration and value iteration, scaling the problem did not have an adverse effect on the number of iterations required for convergence vs. the size of the problem which I was expecting to be larger.

In terms of runtime, both value iteration and policy iteration increase linearly with the number of iterations. It can also be seen that they have a fairly similar run time.



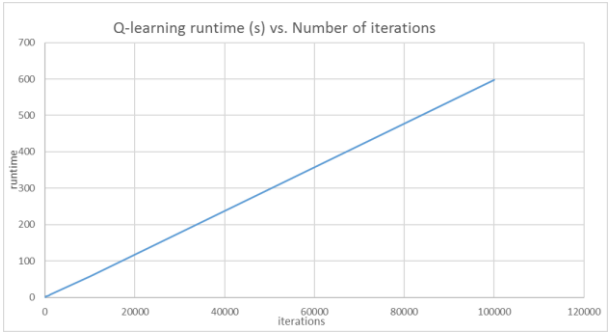


The third algorithm to test on the problem is **q-learning** which seems to be the most difficult to evaluate due to it needing infinite iterations to reach the optimum policy. Again, in this problem I decided to go with an elbow method approach to decide on convergence based on the average returns of the start state vs. the number of episodes as illustrated by the graph below.

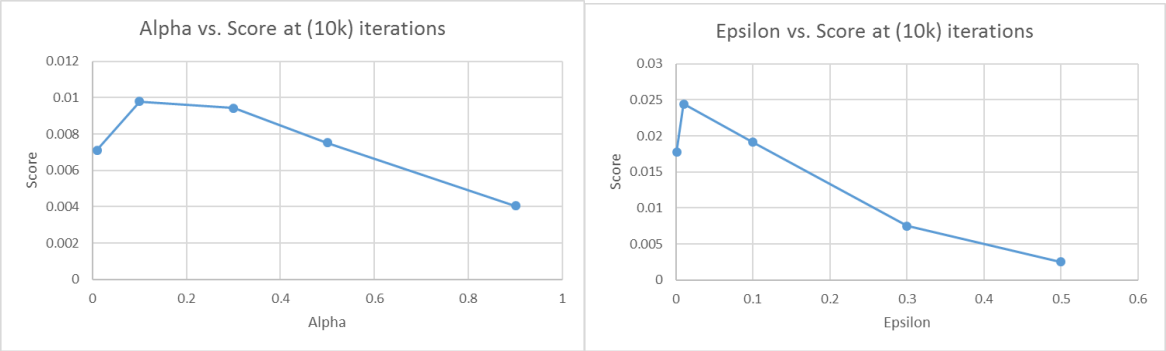


As shown above, the number of episodes required for convergence is around 10,000 episodes which is 51.02 the number of states vs. 16.67 times for the smaller problem so clearly the number of required episodes does not scale linearly with the size of the problem but it also depends on the configuration of the problem where a lot of states are away from the terminating states. This is expected to change had the terminating states been in the middle of the grid.

q-learning runtime also increases linearly with the number of iterations.



Finally, both epsilon and the learning rate have been varied to see the effect on the performance based on the average return from the start state. This has been calculated at 100,000 episodes



Again, the observed effect of alpha and epsilon on the scaled problem is similar to that observed in the smaller problem.

3. Conclusion

In this assignment three different algorithms have been used to solve two Markov Decision Problem (MDP)s and many metrics have been looked at when evaluating each algorithm. Two of the algorithms were found to be essentially the same algorithm with some differenced. Policy iteration efficiency was more efficient than value iteration with $O(s^2)$ per iteration vs. $O(s^2a)$ for value iteration. It was also noticed that the action with maximum q-value rarely changed after the first couple of iterations in both algorithms so we are always checking actions that are unlikely to affect our policy. However, both calculate optimal values. In value iteration, every iteration updates the values and the policy. Whereas, policy iteration only updates utilities and the policy is only allowed to change when a policy improvement step is done and this is why it converges in a smaller number of iterations.

The final algorithm was q-learning which is an online learner that needs to explore the different states and try actions to try and learn transitions and rewards to able to determine the optimal policy. This is different than both the previous algorithms in the sense that everything (rewards, transition, etc.) is known by those algorithms before finding the optimal policy. While in these problems, q-learning did not find the optimal policy for both problems, it was still able to generate a policy that is very close to the optimal policy. A reason for it not being able to find the optimal policy could be because the learning rate is not decaying (which if implemented carefully where it does not get too small to fast, can yield better results). It was also observed that q-learning iterations run faster than both algorithms. however, it takes much more iterations to reach the optimum policy.

References

[1] https://inst.eecs.berkeley.edu/~cs188/fa11/projects/reinforcement/reinforcement.html
[2] https://people.eecs.berkeley.edu/~pabbeel/cs287-fa12/slides/mdps-exact-methods.pdf
[3] https://s3.amazonaws.com/ml-class/notes/MDPIntro.pdf
[4] https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node25.html
[5] Sebastian B. Thrun. The role of exploration in learning control. In David A. White and Donald A. Sofge, editors, Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches. Van Nostrand Reinhold, New York, NY, 1992.