Mikey Ling
mling7
903278121

CS7641 Assignment 4: Markov Decision Processes

Introduction

The purpose of this assignment is to explore the performance of three very well known reinforcement learning algorithms, namely value iteration, policy iteration, and Q-learning, in the context of solving Markov Decision Processes (MDPs). An MDP is a discrete time stochastic control model that contains:

- A set of possible world states **S**
- A set of possible actions **A**
- A real valued reward function **R(S, A)**
- A description **T** of each action's effects on each state

We also assume the Markov Property: the effects of an action taken in a state depend *only* on that state and *not* on the prior history.

In this assignment, we implement MDPs in the context of grid world problems. Grid world problems contain an agent that traverses a grid-like environment. The agent is rewarded (or punished) based on the actions it takes in certain states. We put our agent in two different grid worlds. The first grid world features a 5x5 state space (we call this world the "Easy World"), and the second world features a 15x15 state space (we call this world the "Hard World").

The agent begins its traversal from the bottom left and attempts to reach the goal state in the top right in both grid worlds. The reward for any state other than obstacles and the terminal state is -1. The agent cannot occupy a state space that's designated as an obstacle, and the reward received when the agent reaches the terminal state is +100. The stochastic nature of the problem is as follows: there is an 80% probability the agent moves in the desired direction and a 20% chance it moves in one of the three other directions. For example, if the policy tells the agent to move up, there's an 80% chance the agent moves up and a 20% chance the agent moves down, left, or right (the agent cannot stand still).

The two grid worlds featured in this assignment are shown on the next page. The Easy Grid World's dimensions are 7x7, so the state-action space contains 49 states each with 4 available actions. The Hard Grid World's dimensions are 20x20, so the state-action space contains 400 states each with 4 available actions

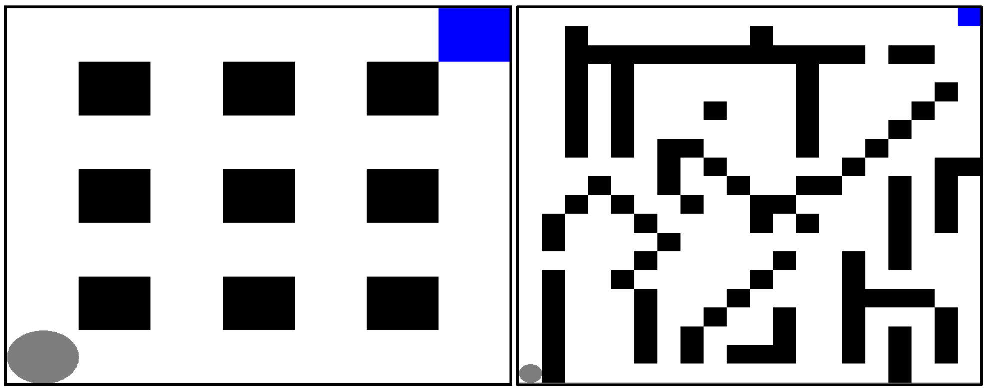
Figure 1 - Easy and Hard Grid Worlds

Why Are These MDPs Interesting?

The grid world is a classic and popular MDP because it's a wonderful abstraction of the real-world. An obvious analogy would be how humans navigate mazes in real life. When first attempting a maze, a human has no prior knowledge of the maze (the environment) and the actions that need to be made to successfully reach the exit (the policy). The human will arrive at the optimal policy (the fastest route to the exit) by choosing different actions at different stages of the maze. This is nearly identical to how our algorithm traverses the maze environment to arrive at the optimal policy. Simply put, the maze grid world is interesting because it's a near-perfect simulation of how us humans would solve an identical problem in real life. The motivation behind many of today's artificial intelligence efforts is to develop machines to perform tasks as well as, if not better than, humans would, and these grid world problems are a wonderful step in that direction.

Reinforcement Learning Algorithms

Three reinforcement-learning algorithms are used to solve the grid worlds mentioned above: value iteration, policy iteration, and Q-learning.

Policy Iteration attempts to converge to an optimal solution by repeatedly manipulating the policy directly, rather than finding it indirectly via the optimal value function. It does so by choosing an arbitrary policy, computing the value function of that policy, and improving the policy at each state until the policy stops changing.

Value Iteration attempts to converge to an optimal policy by finding the optimal value function (the value function that maximizes utility). A value function is arbitrarily initialized and is iteratively improved upon using the Bellman equation to evaluate the given utility at any given state-action pair. This process continues until values of the utilities stop changing by a specified hyperparameter value often referred to as epsilon, ϵ.

Q-Learning is different from policy and value iteration because it does not utilize any prior domain knowledge of the environment to calculate the transition function. Therefore, the algorithm is considered model-free. A

Q-table is initialized (often to zero). This table maps the realized reward of all state-action pairs in the environment. The agent visits each state and chooses the action that yields the highest Q value. This process is repeated until the Q-table stops changing.

All three algorithms and grid worlds were implemented using the Burlap Library. The library's source code is written in Java, and the code used to conduct the experiment is written in Python. The project itself is run using Jython.

Implementation

The Easy Grid world was solved with all three algorithms; however, the number of iterations performed by Value and Policy differed from Q-Learning (100 for Value and Policy Iteration and 1000 for Q-Learning). The number of iterations is greater for Q-Learning because it is a model-free algorithm and requires more interaction with the environment to discover state-action-reward relationships and converge to an optimal policy. Also, Value Iteration and Policy Iteration do not require hyperparameter tuning like Q-Learning does. So, an epsilon value of 0.5 and a learning rate of 0.1 were used in the next two sections to compare the Q-Learning's performance against that of Value and Policy Iteration. It's important to note both of the algorithms converged to a solution well before the maximum number of iterations was reached, as shown below.

Easy Grid World

Figure 2 shows the arbitrary policies initiated by the three algorithms. At the beginning of time, we should not expect the algorithms to maintain similar policies; however, we do expect them to converge to similar, perhaps identical, policies after an adequate number of iterations are performed.

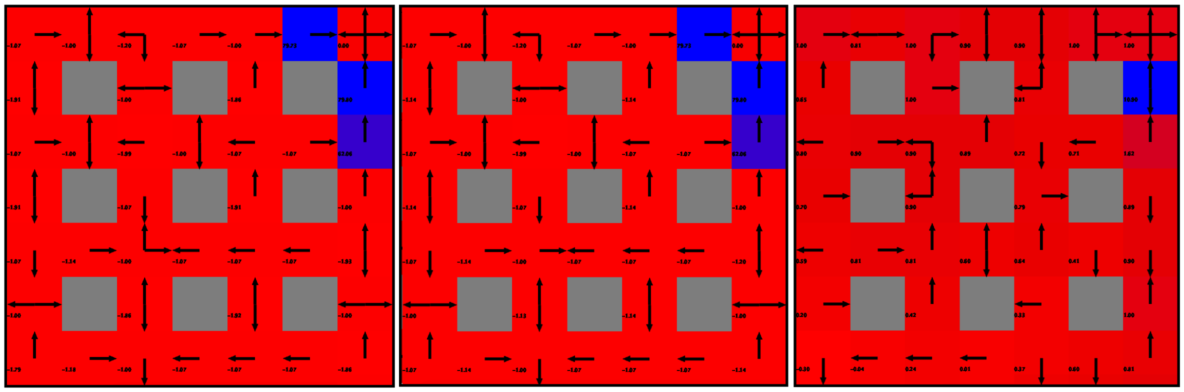

Figure 2 – Policies of Policy Iteration (left), Value Iteration (middle), and Q-Learning (right) After 1 Iteration

Figure 3 shows the amount of time and the number of iterations it takes for the three algorithms to converge to an optimal solution. The graph demonstrates how the number of iterations does not always lead to a longer convergence time. Q-Learning does hold this pattern. It took roughly 23 iterations to converge to an optimal policy. This is less than 10 iterations larger than Policy and Value Iteration, yet the amount of time it took to converge is *much* longer (by about 900ms). On the other hand, notice Value Iteration took *longer* to converge to an optimal policy than Policy Iteration despite requiring less iterations. Compared to each other, Policy

Iteration is often considered to be more efficient because it requires fewer iterations to converge despite each iteration being more computationally expensive. This notion is supported by the bar graphs featured in Figure 3.

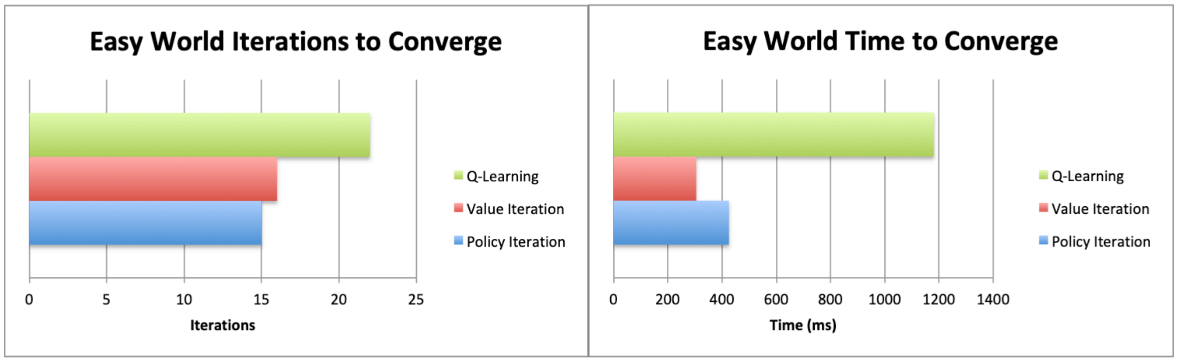

Figure 3 - Time and Number of Iterations to Converge for Easy World

Figure 4 reveals the optimal policy the algorithms converge to. As hoped, all three algorithms converged to identical policies despite having completely different initial policies.

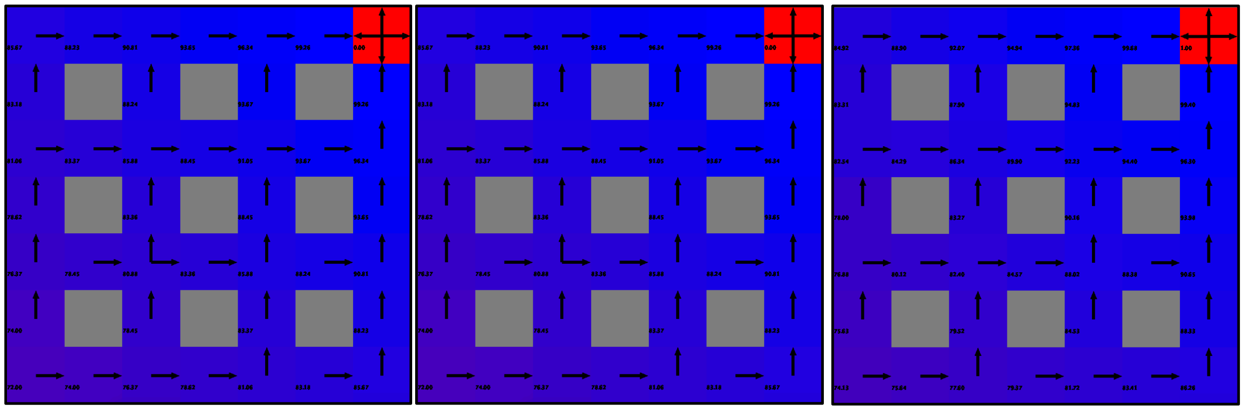

Figure 4 - Policies of Policy Iteration (left), Value Iteration (middle), and Q-Learning (right) After Convergence

Figure 5 expands on the 'Time to Converge' idea presented in Figure 3, which was created by taking the sum of the iterations executed to converge to an optimal solution. Here, the amount of time *per iteration* is plotted. The total reward of the iteration is plotted as well to further demonstrate convergence (as one would expect the optimal policy to have the highest reward). As you can see, the amount of time it takes per iteration for Policy Iteration and Value Iteration increases monotonically as the iteration number increases. Again, the belief that Policy Iteration is more computationally expensive per iteration compared to Value Iteration is supported by the left graph in Figure 5. Notice how Q-Learning initially takes *much* more time per iteration in the beginning of time, and as convergence approaches, the time per iteration drastically decreases and maintains a relatively steady value after convergence. Q-Learning maintains a Q-Table that takes constant time to query (follow the optimal policy). The perturbations in the Time vs Iterations graph is most likely due to the stochasticity of the environment. If the environment were completely deterministic, we would expect the Time vs Iterations graph of Q-Learning to be absolutely constant. The right graph in Figure 5 depicts the rewards per iteration as the algorithms work towards convergence. The Q-Learning takes the most iterations to reach its maximum reward, and it's interesting to see its value seems a little higher than that of both Policy

Iteration and Value Iteration. We would expect the algorithms to obtain the same maximum reward value because they all converged to the same policy, so I believe this discrepancy can be attributed to the differences in the randomly initiated values of the Q-Table.

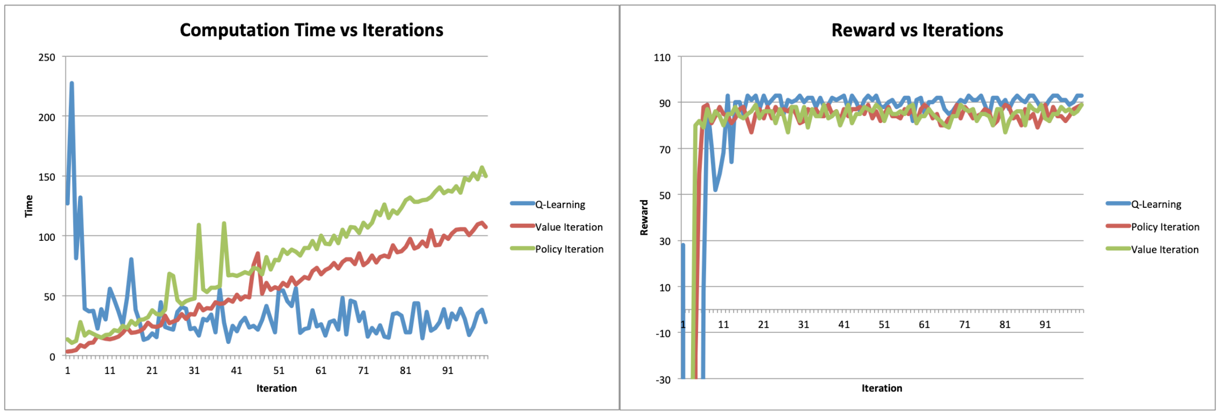

Figure 5 - Computation Time and Reward vs Iteration Graphs for All Three Algorithms

Hard Grid World

The Hard Grid World problem was implemented to see the influence the environment had on the three algorithms under study. The Easy Grid World problem contained 49 different states (7x7 grid), while the Hard Grid World Problem contained 400 (20x20 grid).

With a state space that is nearly eight times larger than the Easy Grid World problem, it's extremely likely we will see an increase in the number of iterations it takes to converge to an optimal solution, the amount of time it takes to converge to an optimal solution, and a decrease in the total reward when the world is complete by the agent.

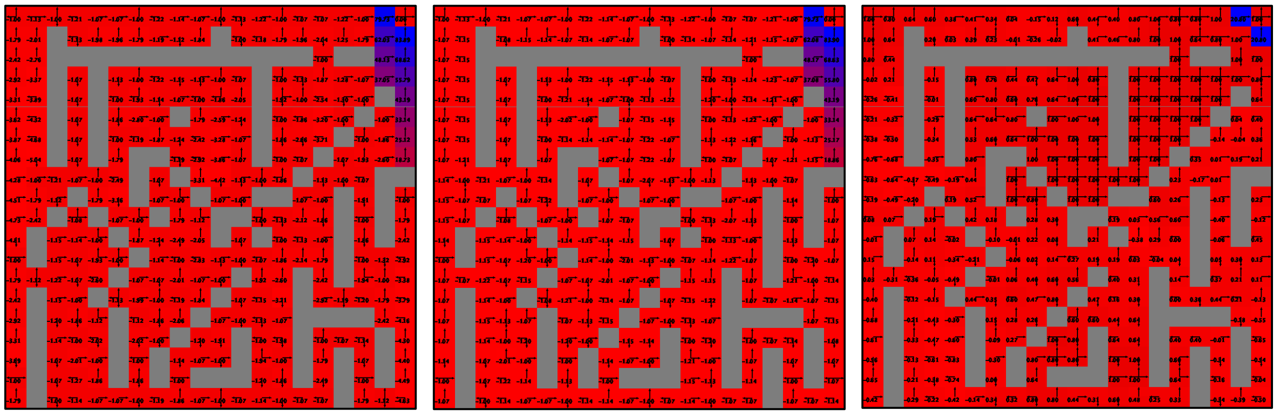

Figure 6 - Policies of Policy Iteration (left), Value Iteration (middle), and Q-Learning (right) After 1 Iteration

Figure 6 shows the initial policies of the three algorithms for the Hard Grid World, just like Figure 2 for the Easy Grid World. Again, we can see the algorithms arbitrarily choose policies that will be improved as the agent iteratively traverses the environment.

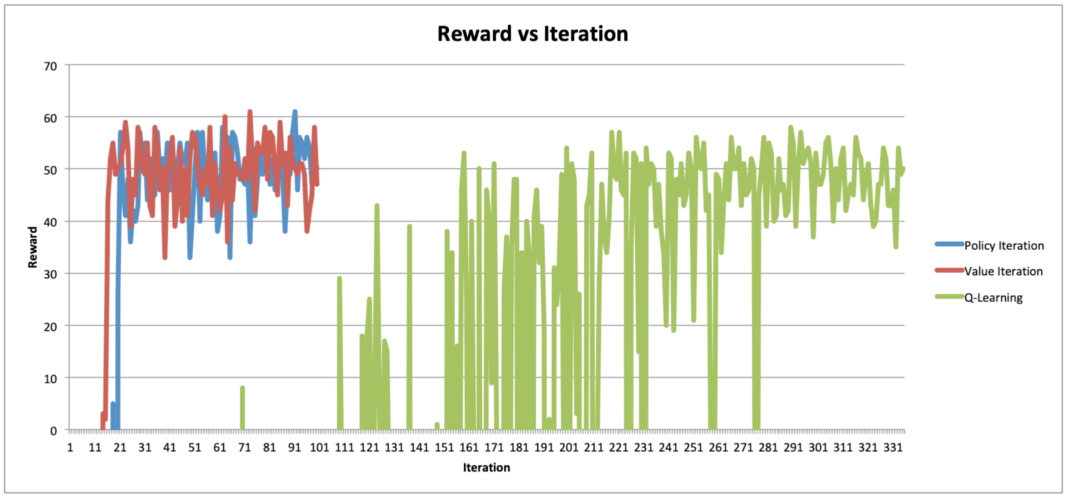

Figure 7 - Rewards vs Iteration for Hard Grid World

We witnessed how Policy Iteration and Value Iteration converged faster than Q-Learning when solving the Easy Grid World problem. This relationship is exaggerated when we solve the Hard Grid World problem because of the exponential increase in the number of states in the environment. Figure 7 shows the reward per iteration of the three algorithms as time goes on. Policy Iteration and Value Iteration converge *must* faster than Q-Learning (around 20 iterations). 100 iterations of these two algorithms were performed. 1000 iterations were performed in our Q-Learning (though Figure 7 only shows a little over 300) implementation because we know it will take longer to converge to an optimal policy. As we expected, Q-Learning took *much* longer to converge than Policy Iteration and Value Iteration because it has no prior knowledge of the state-action-reward interactions of the environment.

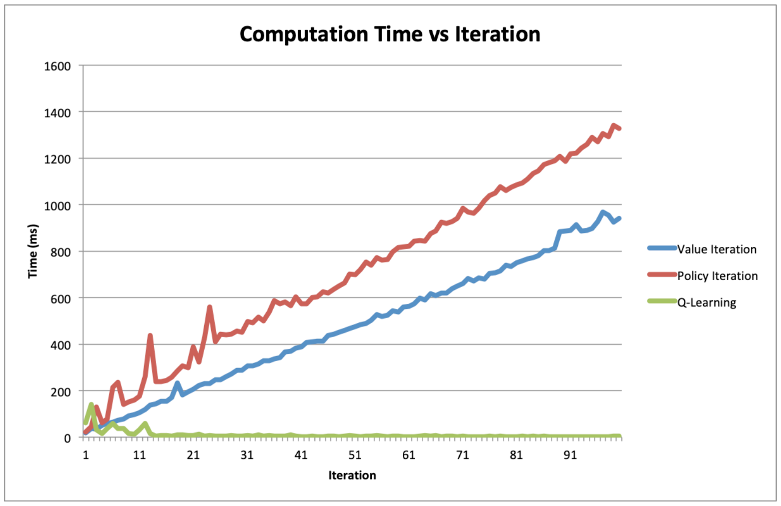

Figure 8 - Time vs Iteration for Hard Grid World

Figure 8 shows the amount of time it takes to complete an iteration for each of the algorithms when solving Hard Grid World. Much like in Figure 5, we see Q-Learning takes *much* less time to execute an iteration than Value Iteration and Policy Iteration. This is because both Value and Policy Iteration must compute the value function and evaluate the policy for current and past iterations, whereas Q-Learning simply follows the optimal policy (which is constant in time complexity).

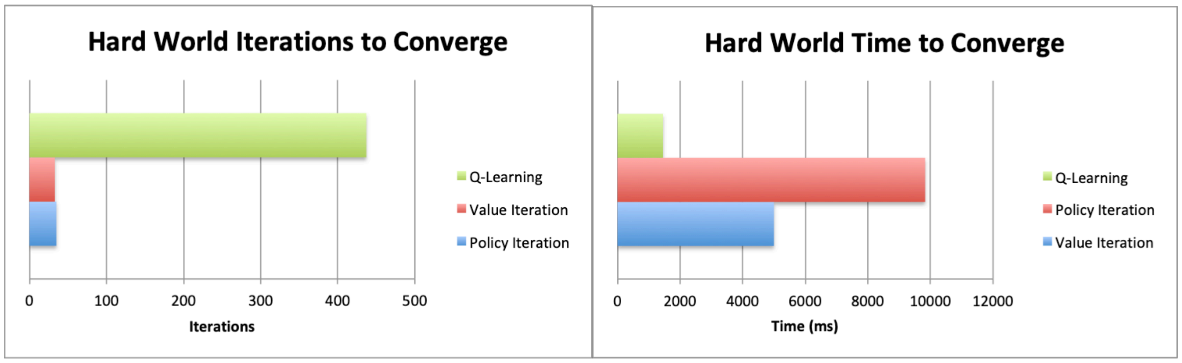

Figure 9 - Iteration and Time to Converge for Hard Grid World

Figure 9 further demonstrates the relationship between time and computational complexity of the three algorithms. Again, due to Q-Learning's extremely fast iteration execution speed, its total time to converge is much lower than Policy and Value Iteration despite having no prior knowledge of the environment. We can also see Policy and Value Iterations converged in nearly the same amount of iterations, yet Policy Iteration takes much more time to converge due to its higher computation complexity per iteration. This higher computational complexity is a result of Policy Iteration having to evaluate the entire policy while Value Iteration evaluates the value function. As we suspected, increasing the size of the state space led to an increase in the time it took for an algorithm to converge to an optimal policy. The effect was not as prominent in Q-Learning compared to Value and Policy Iteration; however, our intuition is correct in this case.

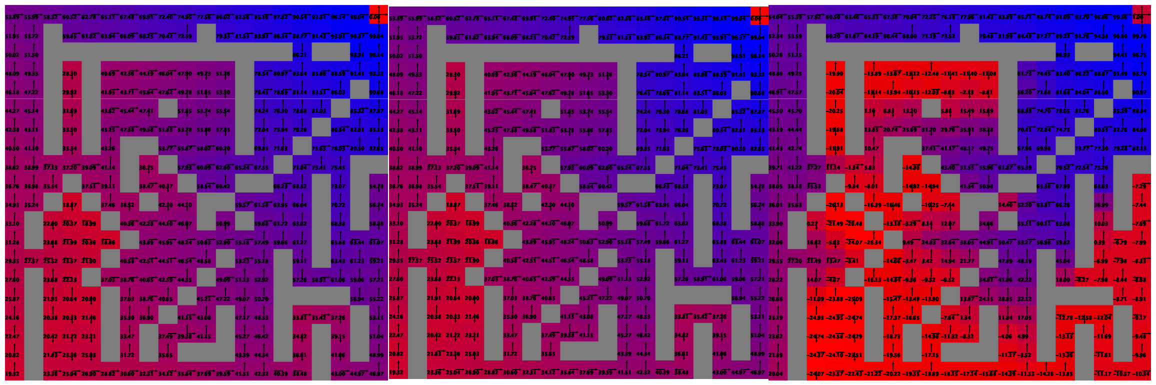
Figure 10 - Policies of Policy Iteration (left), Value Iteration (middle), and Q-Learning (right) After Convergence

Figure 10 shows the optimal policies the three algorithms converged to. Compared to Figure 4, it's interesting to not that the three algorithms *do not* converge to identical policies! Policy Iteration and Value Iteration converge to very similar policies; however, Q-Learning's policy looks significantly different than the other two. My reasoning behind the differing policies is Q-Learning does not guarantee all the states will be visited given a *finite* number of iterations. In theory, Q-Learning is guaranteed to converge to an optimal policy if it is allowed an infinite number of iterations to traverse the environment with its agent. In our experiment, we constrain the algorithm from performing more than 1000 iterations. In this case, it is very likely the algorithm does not get to explore all of the state space.

Q-Learning: A Closer Look

This section of the assignment sheds light on the effects of hyperparameter tuning when solving the Hard Grid World problem. In the Easy Grid World and Hard Grid World sections, an epsilon value of 0.5 and a learning rate of 0.1 were used. In the context of Q-Learning, the epsilon value determines when the agent will act randomly. So, if the epsilon value is 0.5, the agent will act randomly 50% of the time. This is known as an epsilon-greedy strategy.

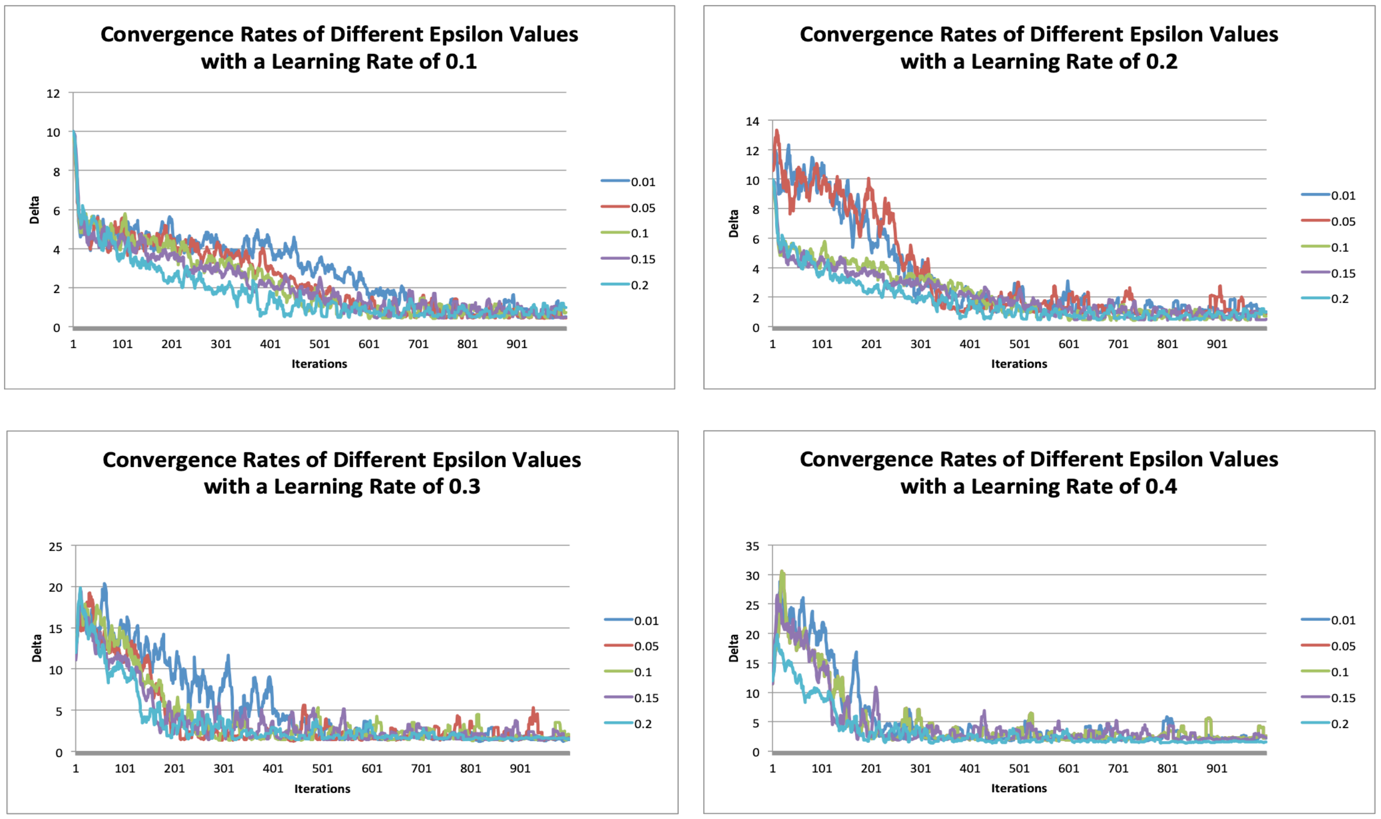

Figure 11 - Q-Learning Convergence with Different Learning Rates and Epsilon Values

An important concept in the world of reinforcement learning is the tradeoff between exploration versus exploitation. Exploration is the attempt to gather more information about the environment, and exploitation is making the best decision given the information currently available. Figure 11 shows the performance of Q-Learning when different exploration rates and learning rates were used to solve the Hard Grid World problem. The learning rate dictates the maximum magnitude the values of the Q-Table can be changed. Larger learning rates mean the learner has the potential to converge to an optimal policy faster with the risk of

“overshooting”. On the other hand, lower learning rates lead to much slower convergence times with the risk of not converging within the given number of iterations.
As you can see in Figure 11, the rate at which the delta between iterations is directly correlated to the learning rate. The figure features four graphs that depict the change in the Q-Table between iterations, so the smaller the change in the Q-Table (the y-axis in the figure) the closer the learner is to converging to an optimal policy. Notice how the general slope all the lines (which represent different epsilon values) is much steeper when the learning rate is 0.4 compared to when the learning rate is 0.1. The agent converges to an optimal solution after about 800 iterations when the learning rate is 0.1, while only 300 iterations were required to converge to an optimal solution when the learning rate is 0.4.

The bottom two graphs in Figure 11 show very similar (and good) performance when solving Hard Grid World, so our analysis will focus on these two visualizations so we can further discuss the effects changing the epsilon values has on the performance of the Q-Learner. If we take a closer look at the performance of the models that have epsilon values of 0.01 and 0.2, we can see the exploration versus exploitation paradigm at work. The models with epsilon values of 0.01 in both graphs take *many more* iterations to converge compared to the models with epsilon values of 0.2. This discrepancy is the result of the different rates at which the models attempt to gain new information about the environment. Remember, the Q-Learner has *no prior knowledge* of the environment; so exploring theRef environment plays a key factor in converging to an optimal solution quickly. The model with epsilon = 0.01 attempts to find new information (act randomly) at a rate 20x *less* than the model with epsilon = 0.2. In other words, the model only deviates from the arbitrarily initiated policy once over 100 moves, whereas the other model deviates from the policy every 5 moves. This leads to a *much* slower convergence.

This begs the question, “which has more influence over a model’s ability to converge to an optimal solution? The learning rate or the epsilon value?” Unfortunately, the answer to this question greatly depends on the problem we are attempting to solve. Undoubtedly, both hyperparameters have large amounts of influence on a model’s performance. But in the context of solving the Hard Grid World problem it seems like the learning rate affects the performance more than the epsilon value.

Conclusion

Policy Iteration and Value Iteration solve MDPs much faster than Q-Learning because they have the advantage of having prior knowledge of the model, rewards, and so on. Q-Learning must explore the environment and figure these relationships out on its own. To help improve the performance of Q-Learning, different exploration strategies can be implemented to help answer the following questions:

- How can learning time be minimized?
- How can computation costs be minimized?
- What impact does the exploration strategy have on the speed and cost of learning?
- How does one trade off exploration and exploitation?

In our case, we implement an undirected exploration strategy called Random Exploration, where actions are generated based on some random distribution and does not take into consideration the learning process itself.

In the context of Q-Learning, this distribution is dictated by the epsilon value of the model where higher epsilon values correspond to a higher probability a random action is selected.

We have found that selecting random actions at a relatively high rate (epsilon value = 0.2) causes the Q-Learner to perform *much* better than a Q-Learner with a relatively low epsilon value of 0.1.

Policy Iteration is commonly considered to be more efficient than Value Iteration because it requires less iterations to converge to an optimal policy. It is able to do so because it evaluates the entire policy during each iteration whereas Value Iteration evaluates the value function during each iteration. And though Policy Iteration converges to an optimal solution in fewer iterations than Value Iteration, its individual iterations are more computationally complex than iterations of Value Iteration. It's for this reason that Policy Iteration can be *slower* than Value Iteration when it comes to time complexity. This notion was confirmed in Figure 5 and Figure 8.

It would be interesting to implement more exploration strategies, like semi-uniform distributed exploration, Boltzmann-distributed exploration, or basic directed exploration, to improve the performance of Q-Learning; however, we believe we were able to analyze the influence of the epsilon hyperparameter on the model's performance without the use of extraneous strategies. Overall, this assignment was extremely insightful, and the I am sure to explore the effectiveness of different exploration strategies in the future.

References

1. *A Survey of Exploration Strategies in Reinforcement Learning* by Roger McFarlane
2. *Deep Reinforcement Learning Demysitifed (Episode 2) — Policy Iteration, Value Iteration and Q-learning* by Moustafa Alzantot
3. *Machine Learning* by Tom M Mitchell