CS4641 MDP Analysis
Colby Tobin
Professor Isbell
April 14, 2019

Introduction

The purpose of this paper is to discuss 3 different reinforcement learning algorithms. More specifically, this paper focuses on Markov Decision Processes (MDPs). MDPs consist of a model that is made up of a set of states that make up the world, a set of possible actions, a reward function, and a transition model. This paper will discuss 2 MDP problems, and the 3 reinforcement learning algorithms will be used to solve the MDP problems where the performance will be compared.

Problem Description

The 2 MDP problems being discussed in the paper are Grid World problems. A grid world problem consists of an agent that can move through a grid map moving either north, south, east, or west. While moving, the agent can encounter obstacles such as walls. To find a solution to the problem, the agent must traverse the grid map to one of the terminating states that exist within the world.

For this paper, the two MDP problems differ in their difficulty and size of the problem. The first problem is an Easy Grid World Problem where the grid has a size of 7x7 with 41 reachable states. To setup the MDP problem, all states are given a reward of -1 except for the terminating state, which has a reward of 100. In the map, the walls are represented as black tiles. In addition to this, for the MDP problem, the agent has a 70% chance of successfully completing an action. The other 30% chance is uniformly distributed across the other 3 possible actions the agent can perform. The easy grid world map is visualized below:

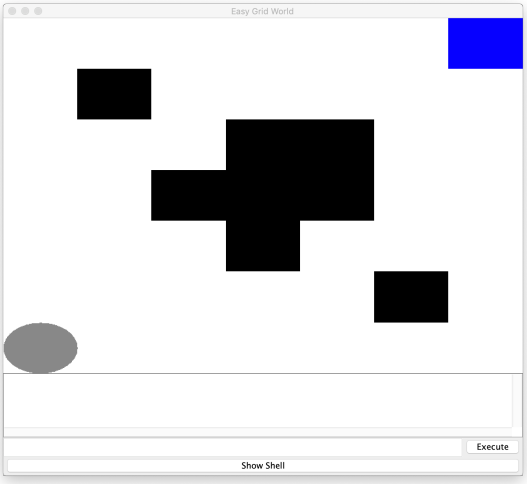

Figure 1: Easy Problem Map

The second MDP problem has a grid with a size of 21x21 with 382 reachable states. The probability of completing an action is the same in this problem as the previous one. In addition to this, for both problems, the agent starts in the bottom right corner and the terminating state is in the top left corner. The hard grid world map is visualized below:

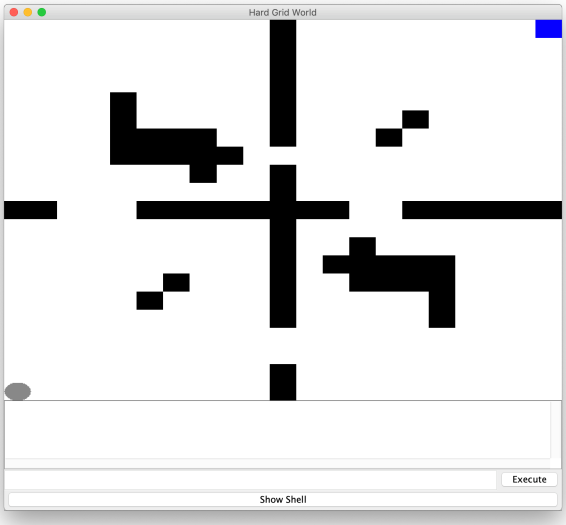

Figure 2: Hard Problem Map

With these two problems, we can compare how various reinforcement learning algorithms perform differently when the size of the problem and number of states increases greatly. In addition to this, Grid World problems are very applicable for these algorithms. Grid World problems are essentially a simplified version of the real world. In the real world, the state space is not always known and the desired actions might not always be performed, which is applicable to how the Grid World problem operates. These problems are a very simplified versions of real life situations and real world problems, which is why these are a good set of problems to test these algorithms.

Algorithm Testing

This section will be split into 2 subsections. First, policy and value iteration will be compared, and then the Q-learning algorithm will be analyzed. For the case of the MDP problems, the optimal performance is characterized by time to convergence, iterations to convergence, number of actions required, and total reward generated.

MDP: Policy and Value Iteration

The value iteration algorithm works by finding the optimal utility for each of the states within the world. It uses the Bellman update equation to iteratively update the utility based on the neighbor states using the transition model. The optimal policy can be derived from the optimal utility of the states. Policy iteration works by updating the policy directly. At first, an arbitrary policy is chosen. Based on that policy, the values of surrounding states are calculated. Based on these new

values, a new policy is chosen, which means that with every new iteration, the new policy is better or equal to the previous policy.

For the algorithms themselves, a discount factor of 0.99 was used. A high discount factor means that the algorithms will result in values that are based on the discounted future reward that the agent will see based on the current policy. With the grid world problem, because non-terminating states have a reward of -1, the problem will find a policy that results in the shortest path to the terminating state in order to maximize the future reward.

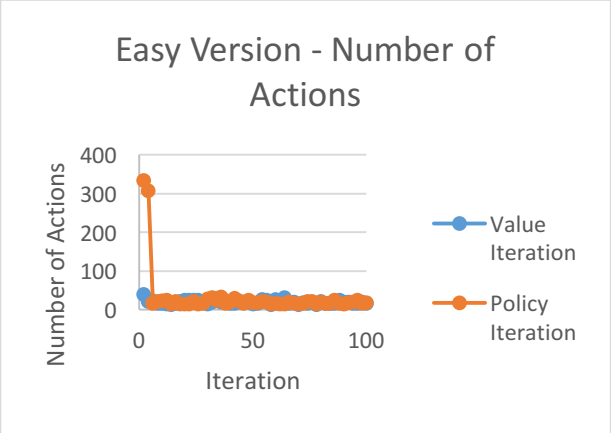

Figure 3: Easy Version # of Actions

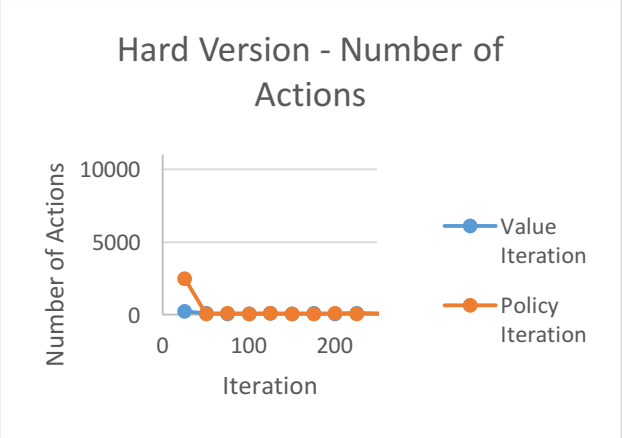

Figure 4: Hard Version # of Actions

Above, the number of actions the agent should take to reach the terminating state given the current policy was plotted against the number of iterations used to converge to that policy. 100 iterations were chosen as the maximum for number of iterations because, in both the hard and easy problem, no more than 100 iterations were required to reach the optimal number of actions. Based on the graphs, it can be seen that the number of actions required converges to the optimal value in around the same number of iterations for both iteration policies. This makes sense that it will find the optimum number of actions quickly because each state has a -1 reward, which means that it will try to minimize the number of actions to maximize reward.

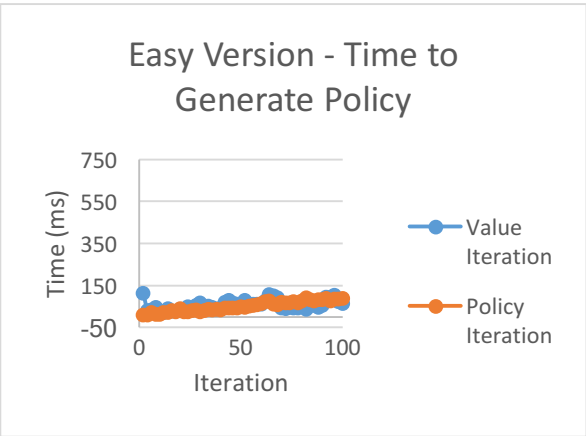

Figure 5: Easy Version Time for Policy

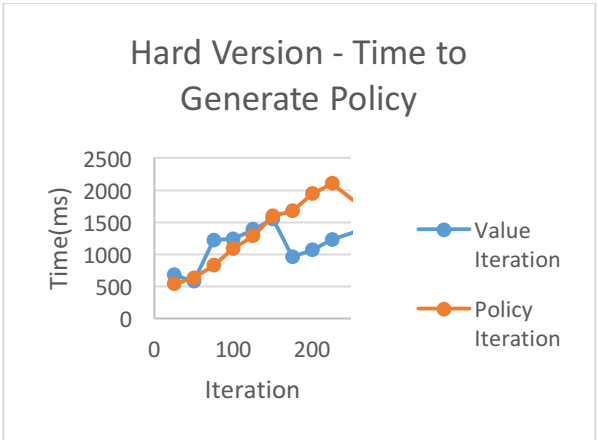

Figure 6: Hard Version Time for Policy

The above graphs show a representation of the wall time required to perform the iterations for each specific algorithm. Based on the above graphs, it can be seen that the value iteration and

policy iteration algorithm both took relatively similar times to generate the policy for the easy version. However, based on the hard version, it can be seen that the policy iteration algorithm took a longer time to generate a policy. This makes sense because policy iteration requires the algorithm to solve a set of system of equations, whereas value iteration just updates values based on the neighbors. The time difference between policy and value iteration is more prevalent in the hard version as well, and this is due to the scaling of the problem. The number of neighbors doesn't really change as the number of states increase, but the size of the system of equations scales faster with more states, which explains why the policy iteration algorithm takes much longer in the hard version.

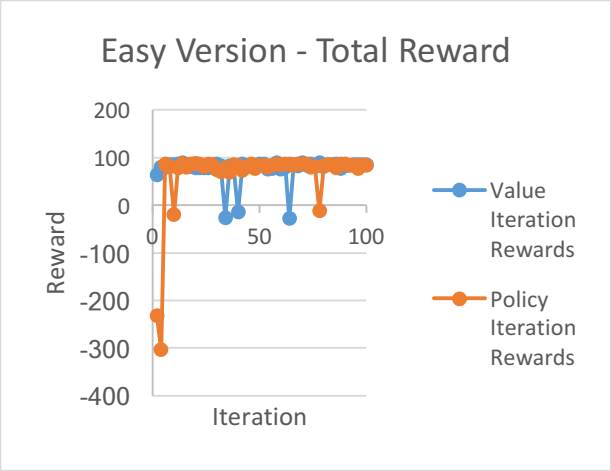

Figure 7: Easy Version Total Reward

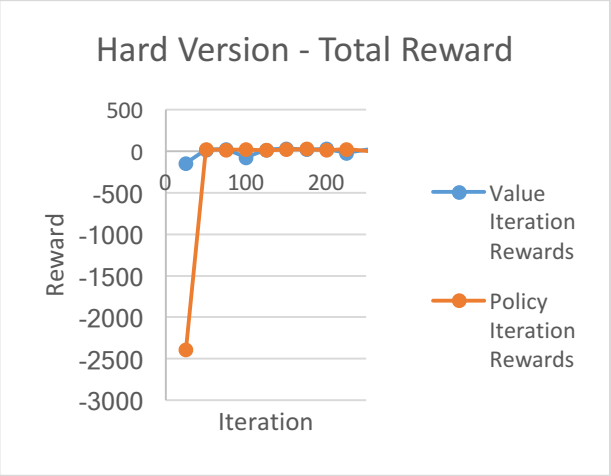

Figure 8: Hard Version Total Reward

The above graphs depict the reward for a given policy at the current iteration number on the x-axis. The algorithm was considered to converge when the reward leveled off and did not change much when the number of iterations increased. In the easy version, the value iteration algorithm converged to the reward of 85 at around 6 iterations, whereas the policy iteration algorithm converged to the same reward around 14 iterations. In the hard version, the converged reward value was around 22. The value iteration algorithm converged to 22 at around 50 iterations, and the policy iteration algorithm also converged at around 50 iterations as well. This follows the pattern of the number of actions required to reach the terminating state graph as well due to the fact that in order to maximize reward, the agent must take as few actions as possible to get the most reward.

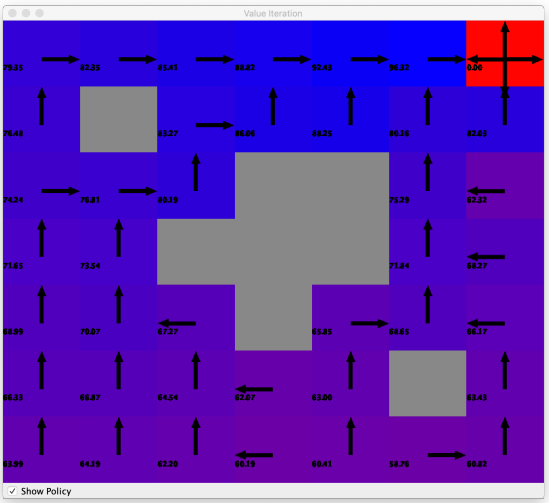

Figure 9: Easy Value Iteration

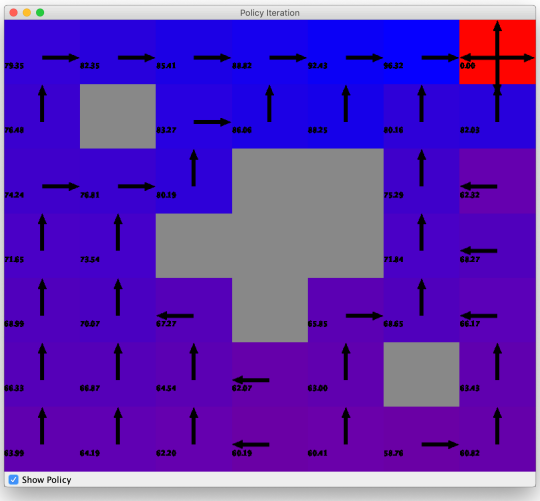

Figure 10: Easy Policy Iteration

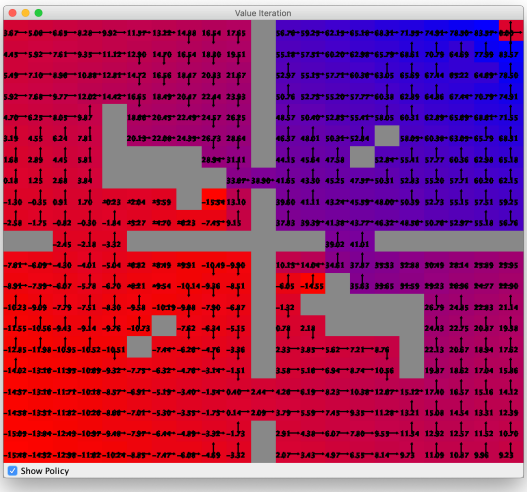

Figure 11: Hard Value Iteration

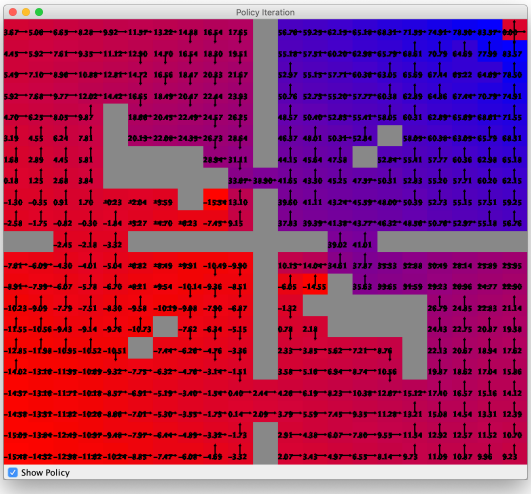

Figure 12: Hard Policy Iteration

The above figures are a visual representation of the policy that each of the algorithms generate. Based on the above figures, both algorithms generate correct policies where they are very close to the optimal policy for the problems. Because value iteration converges in less iterations and takes less time to perform each iteration, value iteration can be considered the better algorithm for these problems. In addition to this, based on the time graphs, the value iteration algorithm is much better for the harder problem than the easier problem. This is likely due to the fact that value iteration scales better for more states because the number of neighbors stays the same, whereas the policy iteration algorithm scales worse due to the fact it must solve a system of equations. In the easy version of the problem, the value iteration performed slightly faster, but in the hard version of the problem, the value iteration algorithm performed almost twice as fast.

Reinforcement Learning: Q-Learning

The previous 2 algorithms are very useful when there is a transition model that exists within the world. If there is no transition model, the algorithms would not be useful, so a different reinforcement algorithm must be used. In this case, the q-learning is a good algorithm to use when there is no transition model. This algorithm learns over time without a model backing it up. Similar to the other algorithms, q-learning has a discount factor associated with it. 0.99 was chosen as the discount factor because this will optimize the policy based on future reward, which will result in the least amount of actions due to the -1 reward system. The other hyperparameter for this algorithm is the learning rate. A low learning rate means that the algorithm will learn slowly versus a high learning rate, means that q-values are updated quickly, so learning happens quickly. For this algorithm, 0.1 was selected as the learning rate because that is the industry standard. For the q-learning algorithm, the other important thing is the exploration strategy. For this analysis, there were 7 different variations of exploration strategies that were used for these MDP problems. The 3 main strategies were greedy deterministic, epsilon greedy, and the Boltzmann exploration strategy. In the epsilon greedy and Boltzmann exploration strategies, the epsilon value and temperature values were set to 0.1, 0.4, and 0.7 respectively to test how these variations impact the performance of the algorithm. With regards to the nature of the exploration algorithm, the greedy deterministic exploration strategy always takes the most optimal step for exploration. The epsilon greedy strategy performs similarly to the greedy deterministic, except that it will perform a non-optimal step with a (1 – epsilon) chance. The Boltzmann strategy works similarly to simulated annealing where the algorithm will first explore more to non-optimal steps, but as the number of iterations increases, it will only choose optimal steps to reduce exploration. Because of the computational complexity of some of the algorithms, the maximum number of iterations was set to 200. To compare the performance of each of these algorithms, the total reward of the policy and the time to create that policy were compared. Below is the graph of the total reward of each of the policies for the various exploration strategies of Q-learning:

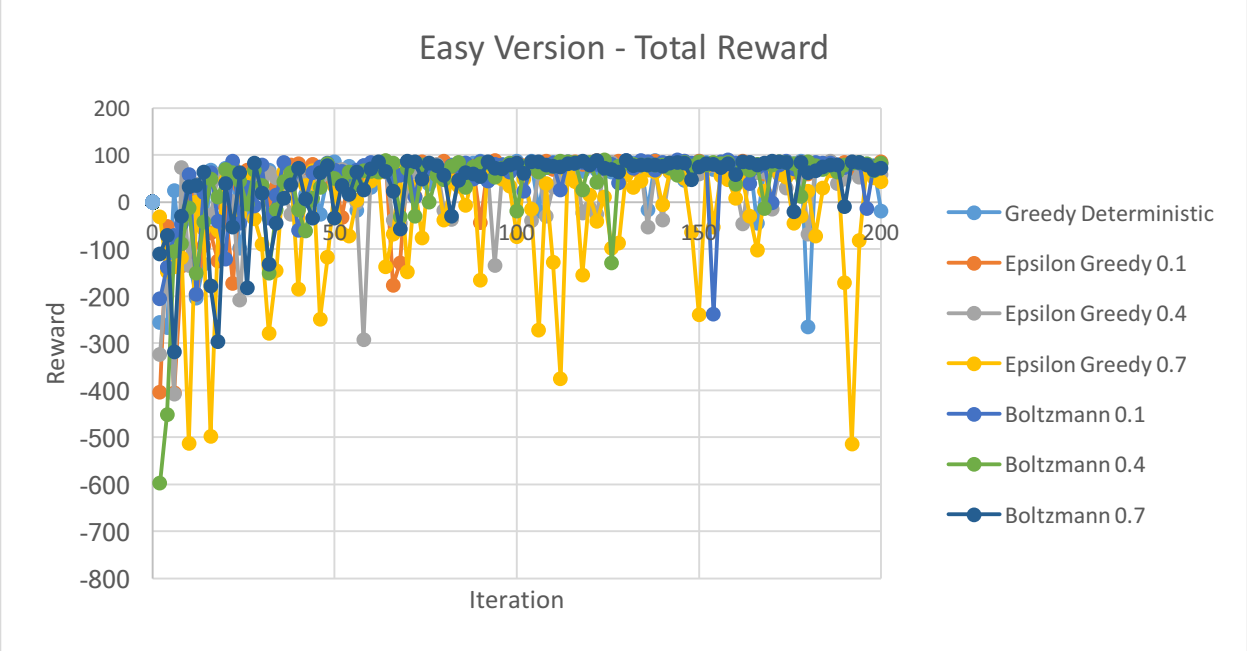

Figure 13: Q-Learning Easy Total Reward

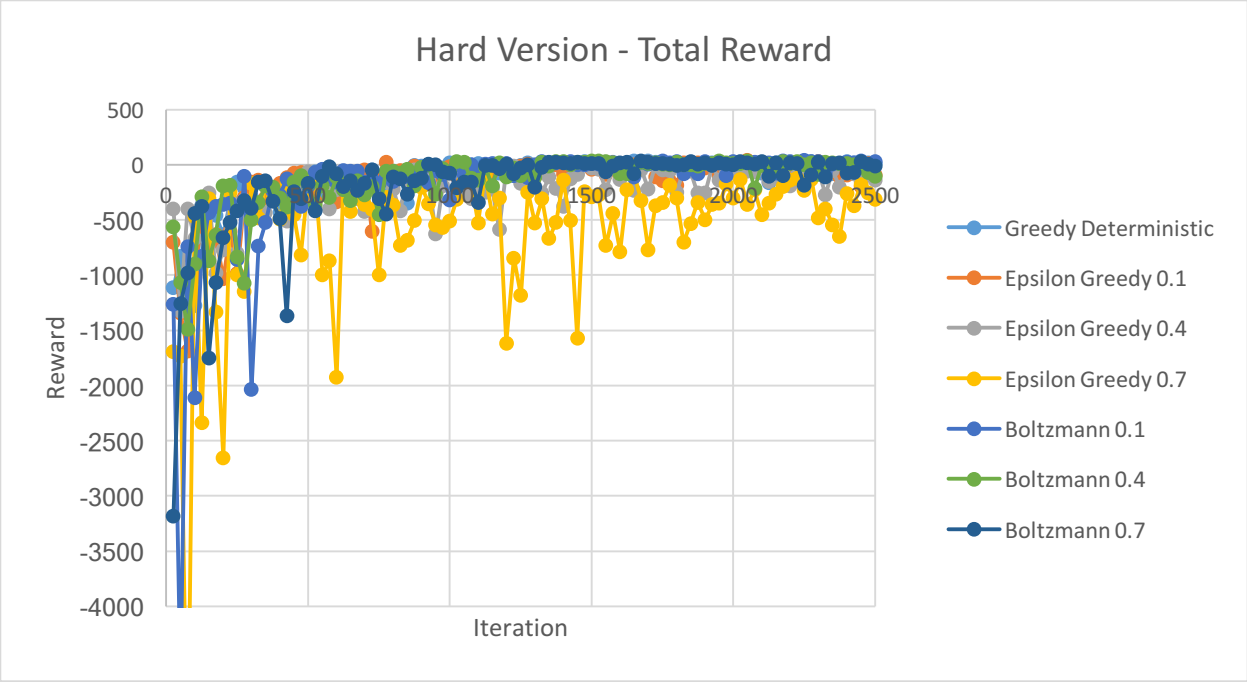

Figure 14: Q-Learning Hard Total Reward

When looking at the above graphs, it can be seen that all 3 algorithms converged fairly quickly. However, it looks as though the greedy deterministic and Boltzmann exploration strategies converged much quicker than the Epsilon greedy algorithm. In addition to this, the epsilon greedy algorithm showed a great number of outliers in the graph especially when Epsilon was set to 0.4 and 0.7. When looking at the easy version of the problem, it can be seen that the q-learning algorithm does eventually converge to a policy for the Boltzmann exploration strategies that is very similar, but slightly less reward than the value and policy iteration algorithms. With the other exploration strategies, the algorithm does not truly converge to a true policy in the easy problem. In the hard problem, based on the number of outliers in the graph, it seems as though the Q-learning algorithm does not really converge to an optimal policy, as the policy performs worse than the value and policy iteration algorithms. This is likely due to the fact that this algorithm did not have any transition model to work on to help generate the Q-values to derive a policy. Q-learning is expected to perform worse in this regard because of the lack of a model. In addition, when comparing the performance of the different epsilon values, it can be seen that lower values seemed to perform better, meaning that the algorithm prefers the greedy/optimal approach to the sub-optimal steps. This makes sense based on the nature of the problem, since the reward system is setup so the optimal policy should generate the shortest path to get to the terminating state.

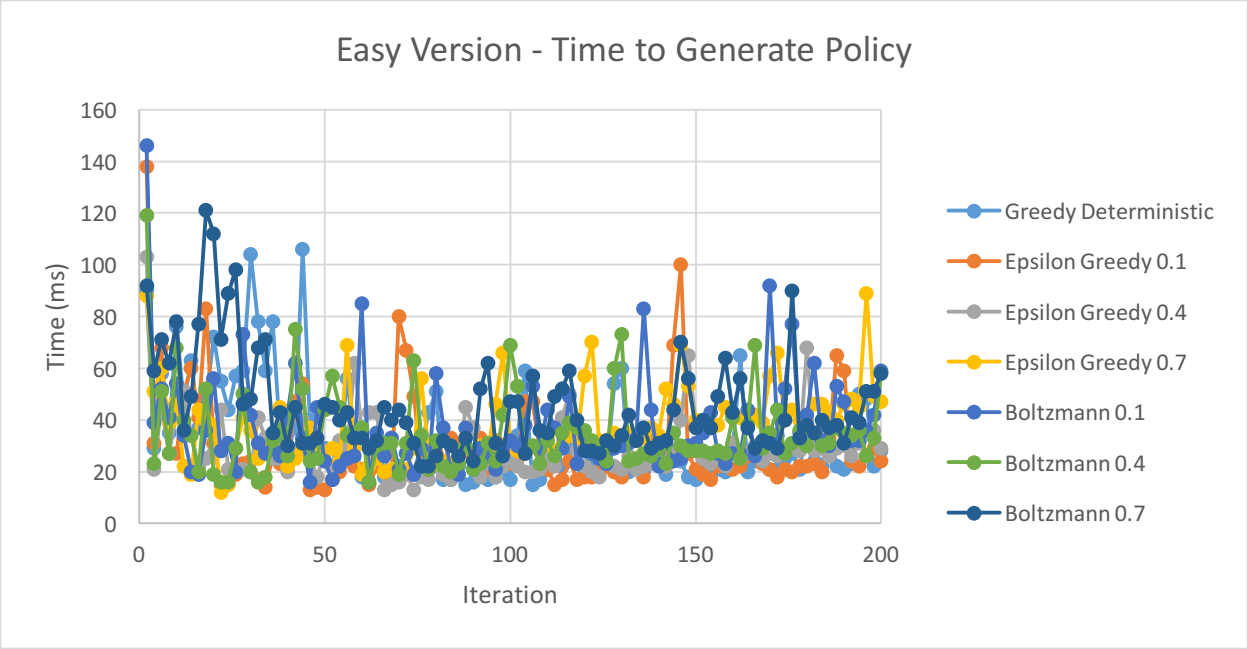

Figure 15: Q-Learning Easy Time to Policy

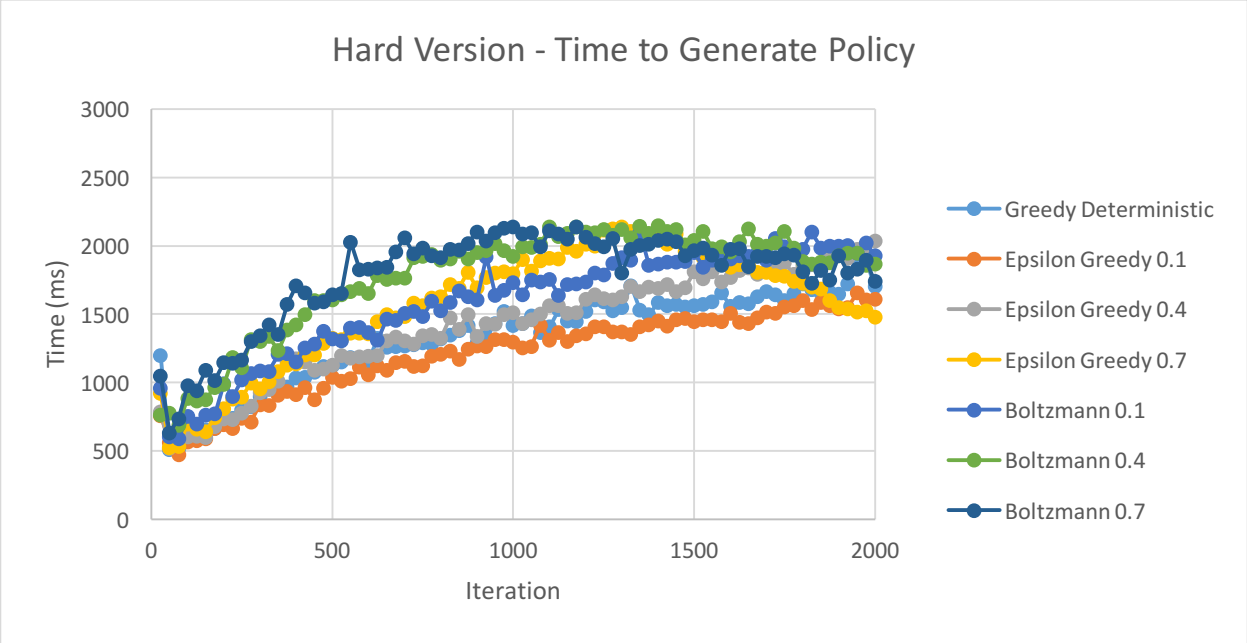

Figure 16: Q-Learning Hard Time to Policy

When looking at the above graphs, they depict the amount of time taken for each of the algorithms to run at the different iteration numbers. When comparing the hard and easy version of the problem, it is fairly straightforward to see that the easy version takes less than 10x as long to complete the algorithm as the hard version of the problem. This makes sense because the size of the hard problem is much larger than the easy problem, so there are more states to traverse for q-value updating. When comparing the algorithms amongst themselves within each problem, it can be seen that the greedy algorithms performed much quicker, where a smaller epsilon value had a better performance time. This makes sense because, like before, the nature of the problem

is to support the greedy approach for exploration, so taking optimal steps is preferable to sub-optimal steps, which means that the greedy approaches should take less time to complete, which is evident in the graphs. The Boltzmann algorithms perform the slowest, which makes sense because it is essentially performing simulated annealing, which is a much slower process than the typical greedy approach.

As mentioned before, these algorithms were measured based on speed of convergence, total time taken to generate the policy, and how close the policy is to the optimal policy. When looking at these measurements, it can be seen that the greedy deterministic and the Boltzmann strategy both resulted in similar policies at the end that were equally close to the optimal policy based on the total reward generated. With this in mind, when looking at their time for convergence to the policy, it can be seen that the greedy deterministic algorithm takes less wall-time to converge than the Boltzmann strategy, making this the best strategy for the q-learning algorithm. Because this is the "best" exploration strategy, the policy generated from the greedy and Boltzmann algorithms are shown below:

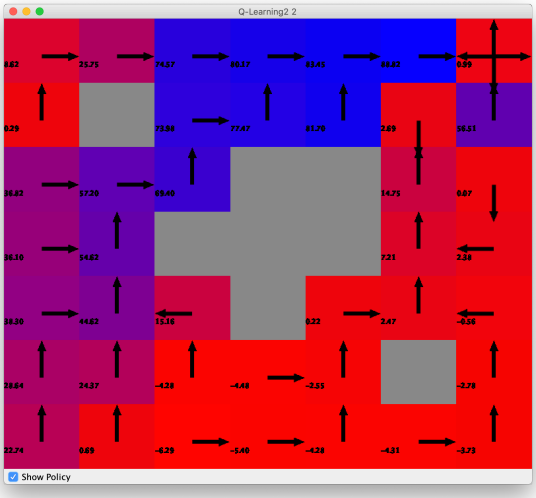

Figure 17: Easy Deterministic Greedy

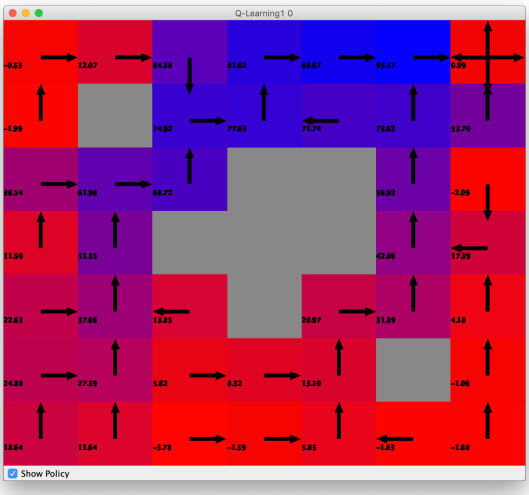

Figure 18: Easy Boltzmann

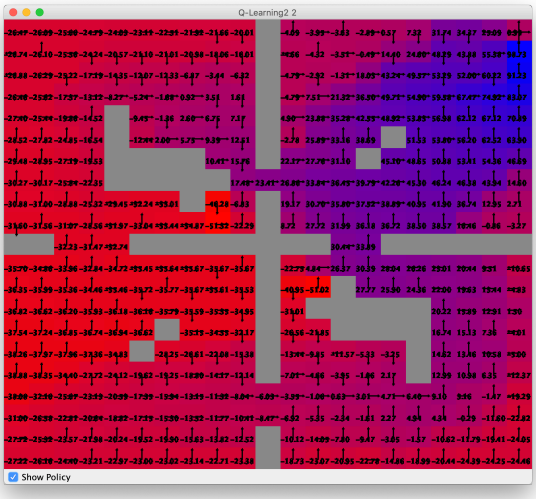

Figure 19: Hard Deterministic Greedy Figure 20: Hard Boltzmann

The above graphs are visual representations of the Q-learning algorithm policies with the Boltzmann and deterministic greedy exploration strategies. When comparing the 2 visualizations above, the easy problem resulted in similar policies that were generated; however, in the hard problem, it can be seen that the deterministic greedy exploration strategy produced a more accurate policy for the problem. In addition to this, the greedy deterministic strategy performed much quicker and converged in less iterations.

Conclusion

Generally, when comparing the value and policy iteration algorithm to the q-learning algorithm, it can be seen that the q-learning algorithm took the most amount of time to converge to a policy. This makes sense based on the nature of the algorithm, since the q-learning algorithm does not have a transition model or any model to base the algorithm off of, which makes it take more time to converge. In addition to this, when comparing the correctness of the policy generated, it can be seen that the q-learning algorithm generates a policy that does not produce as much reward as the value or policy iteration algorithms. Similar to convergence time, this is the case because the q-learning algorithm must learn the model, whereas the other algorithms are provided with the transition models. If the transition model is available, it is evident that the value iteration algorithm is the ideal MDP algorithm to be used based on its speed of convergence and generation of an optimal policy in comparison to the policy iteration algorithm and the q-learning algorithm. However, in the event that there is no transition model, it can be seen that based on the design of the problem, the deterministic greedy exploration strategy is the best approach for using the q-learning algorithm.

The above experiments embody the importance of the domain knowledge of the problem at hand. They demonstrate how the algorithms perform differently because of the problem. The knowledge of the transition model plays a big role in demonstrating how an algorithm can converge quicker with more accuracy as seen throughout this paper. Under the right circumstances with the correct domain knowledge (such as the transition model), the value iteration algorithm is the best for these problems because of its accuracy and speed. In the event that Q-learning must be used because of a lack of a transition model, based on the nature of this problem, the best exploration strategy to be used is greedy deterministic strategy. Overall this paper demonstrates the importance of domain knowledge when it comes to determining the optimal algorithm for solving MDPs.

References

GitHub Code: https://github.com/tobincolby/CS4641-MDP
Algorithms Source: *BURLAP* (http://burlap.cs.brown.edu/) + associated documentation
Algorithm Testing Template: https://github.com/juanjose49/omscs-cs7641-machine-learning-assignment-4/blob/master/