CS 4641 Markov Decision Processes Analysis

Prateek Shah (pshah316)
Section A
Professor: Charles Isbell

Due: April 22nd, 2018

Introduction

This paper applies reinforcement learning techniques in order to solve decision problems. Specifically, Markov Decision Processes (MDPs) are considered; MDPs have states, rewards, and actions. In many cases, the information that is presented by the problem can greatly affect what type of algorithm is most optimal to use. In the case of this paper, two MDP problems will be presented. Then, they will be solved via three distinct algorithms to compare performance.

Problem Specifications

Both problems that are considered in this assignment are Grid World problems. In such a problem, an agent can move through a Cartesian grid system, and encounter objects such as walls. There are terminating states that end the problem. Justification for problem relevance is presented at the end of the section.

The first world is a simple 7x7 grid that has 43 states that are reachable. The terminating tile is in the top right corner and the agent starts in the bottom left corner. In this world, all tiles carry a reward of -1, while the terminal tile has a reward of 100. There are black tiles, which are walls. There is also a line of symmetry between the top right and bottom left, which leads to there being a number of theoretically identically optimal paths for the agent. Additionally, the agent has only a 70% of successfully performing an action. The other 30% is distributed among the other 3 directions the agent is able to go (10% each). This world is visualized below:

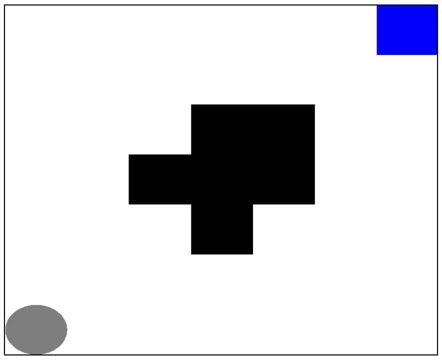

Figure 1: Easy Grid World

The second world is a more complex 21x21 grid with 380 reachable states. As before, the terminating tile is the top right and the agent starts in the bottom left. The probability of a successful action is also the same as the previous world. Again, symmetries are introduced into the world to see how the algorithms behave in such a situation. Visualized:

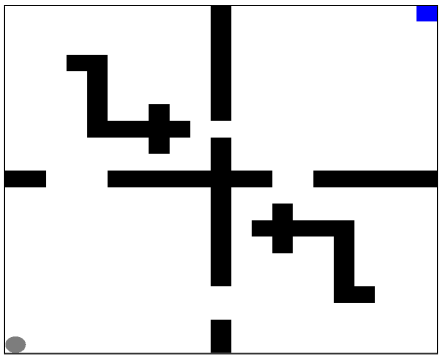

Figure 2: Hard Grid World

These two worlds give a chance to compare various iteration and reinforcement algorithms to see how they perform when the number of possible states increase (by nearly 10x). Further, the choice of Grid World is especially appropriate, as it applies moderately well to the real world. The most clear example is when looking at robots: actuators are not always perfect, and often the entire state space is not known beforehand. This has applicability to classical robotics, but also to more current fields such as autonomous driving. These problems are simplified representations of real problems in Computer Science, and are thus valuable examples to test various algorithms on.

Algorithm Comparisons

Value and Policy Iteration will be considered and discussed in the first section, followed by an analysis using Q-learning. As with the previous assignments, the focus of the analysis will be *performance*. In this case, performance is operationalized by both number of iterations taken to convergence and wall time taken.

Value and Policy Iteration

Value Iteration uses the Bellman equation to iteratively update the utility of all the states, based on the utilities of neighbors (considering those that the transition model says are reachable). Eventually, optimal values for states are reached, and an optimal policy can be derived from that. Meanwhile, Policy Iteration selects a policy and updates all the values based on that policy. Using those updated values, a new policy is chosen. In this way, we know that any new policy is strictly better or equivalent as the previous iteration.

In both situations, a discount factor of 0.99 was used. This value, gamma, represents the weight that is given to a future reward. As such, this high gamma leads to heavy optimizations of future reward. Due to the constraints on how the world was created, a discount factor of 0.99 means that the agent will attempt to find the shortest path through the world, in order to terminate the game most quickly (since there is a reward of -1 for each time step).

The number of actions that an agent should take, according to a policy, is plotted against the number of iterations required. In both cases, empirical testing showed that no more than 100 iterations were needed, and thus, 100 iterations was used as a cap. The comparisons:

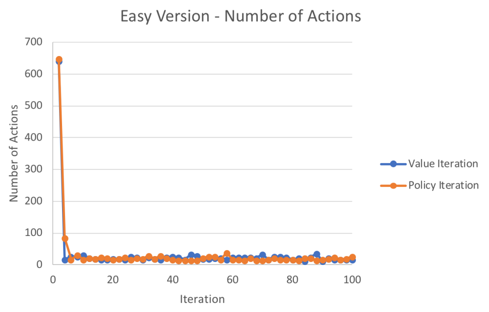

Figure 3: Easy World Actions

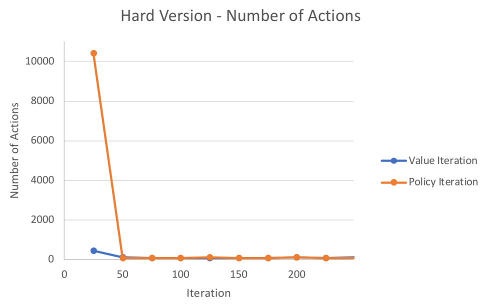

Figure 4: Hard World Actions

In both problems, with both algorithms, the number of actions that are produced quickly converge to the optimum, as a function of number of iterations. The convergence points are roughly the same as what is found in the reward graph below. This makes sense, since the best reward in this type of world will occur when the number of actions are minimized (due to the -1 reward at each time step).

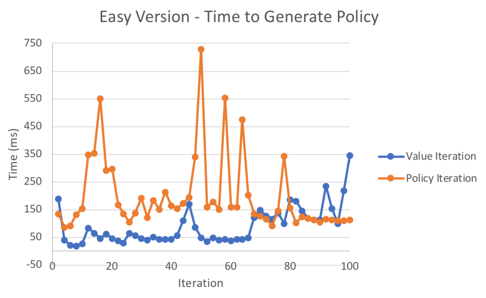

Figure 5: Easy World Time Taken

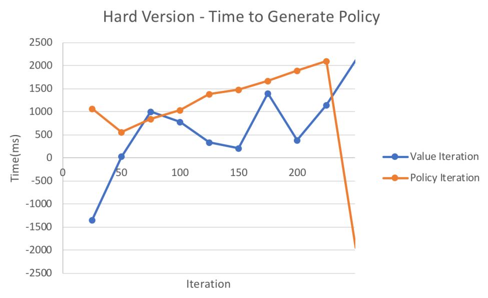

Figure 6: Hard World Time Taken

Discounting random extrema, policy iteration was slower, per iteration, than value iteration. This data confirms what was theorized. Policy iteration involves solving the entire system of equations, versus simple neighborhood calculations for value iteration. This is also seen when comparing the runtimes of the two problems. The difference in time taken in the easy problem is roughly 2x, while it is 15x in the hard problem. The scaling of the problem explains this, as the system of equations scales faster with more states in comparison to the neighborhood.

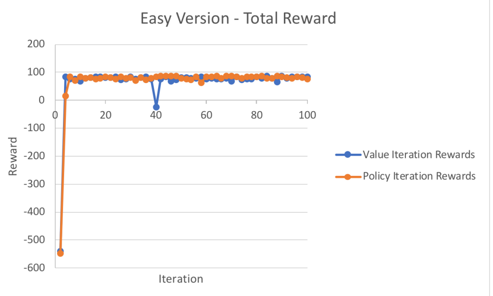

Figure 7: Easy World Reward

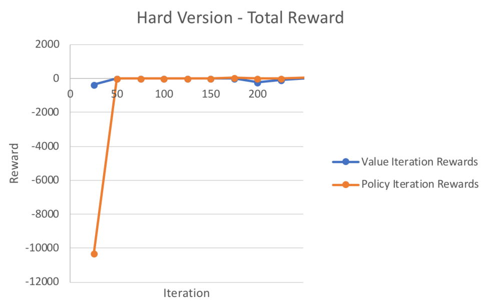

Figure 8: Hard World Reward

The determination of convergence was based on when the rolling average on the above graph became relatively constant (no extra reward was gained from extra iterations). In the easy world, convergence (to a reward of about 85) was reached within 4 iterations with value iteration, and within 12 iterations with policy iteration. In the hard world, convergence (reward around 50) was reached within 20 and 30 iterations, respectively for the two algorithms. In both problems, there were relatively close to the true global optimum (the manually calculated optimal path). The differences in optimal reward are accounted for by the fact that the two grids simply have different sizes.

When we visualize the policies after the iterative cap, we notice the algorithms produce very similar results (arrowing indicating direction, and numbers indicating utility at that state):

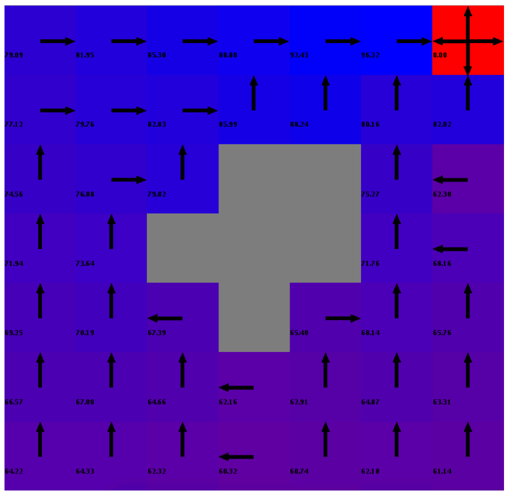
Figure 9: Easy Policy Iteration

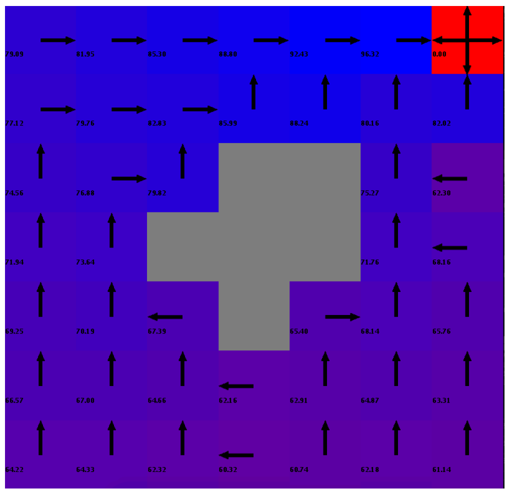
Figure 10: Easy Value Iteration

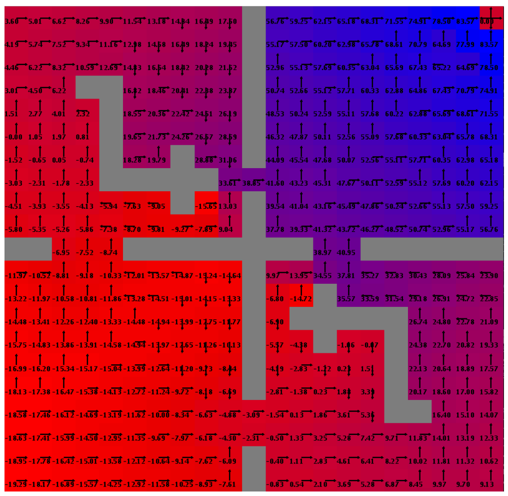
Figure 11: Hard Policy Iteration

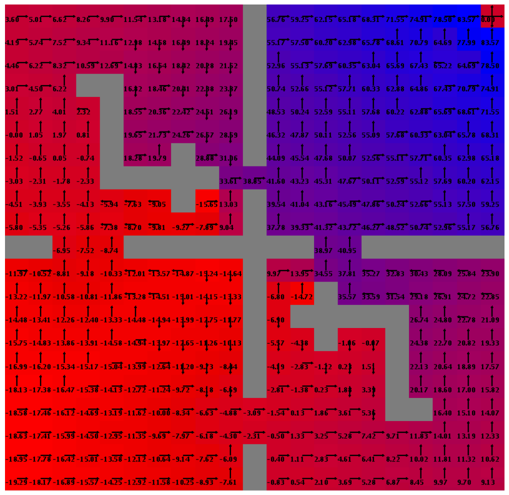
Figure 12: Hard Value Iteration

In general, for these problems, value iteration seems to be the better choice. Since both choices produced correct results, the performance should be considered in terms of time and scalability. Value iterations scaled better to the larger problem. This makes sense since the size of the policy directly effects how long policy iteration takes, but not value iteration.

Additionally, within each iteration, value iteration was much faster. Overall, in the easy problem, it was 8-10x faster, and for the hard problem, it was 20-25x faster.

Reinforcement Learning: Q-learning

When a transition model is not known for an MDP, the previous two algorithms are not very useful. In such a situation, a reasonable reinforcement learning-based strategy would be to use Q-learning. This algorithm does not need a transition model for the world, and instead learns over time. This algorithm has some hyperparameters that can be adjusted. As before, the discount factor (gamma) was kept constant at 0.99, to optimize for future reward. Another parameter was the learning rate, which determines how far back the agent looks in time in consideration of decisions. The standard industry value of 0.10 seemed to work well in these problems. Finally, there is the exploration strategy for the algorithms. This is where three different strategies were tested. The first was a greedy exploration, where the most optimal step in at that time is chosen. This algorithm, of course, is quite simple. The next strategy is epsilon greedy, which is the same thing as the greedy, except for a $1-\epsilon$ chance of selecting a non-optimal move. Finally, the last strategy was Boltzmann exploration, which is effectively a simulated annealing. As such, this method has a temperature variable that determines how likely a non-optimal move can be taken at a given time step (lower is more adventurous). In the last two algorithms, the experiment sets ϵ and temperature values at 0.1, 0.4, and 0.7 to see how this effects the strategy. An iterative cap of 200 rounds was chosen for computational restraints.

First, total reward as a function of iterations is considered in both the easy and hard world:

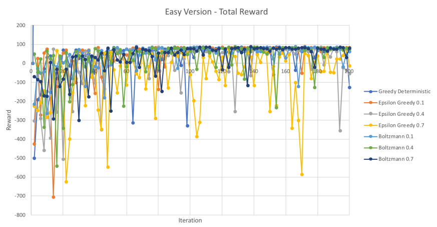

Figure 13: Easy World Reward

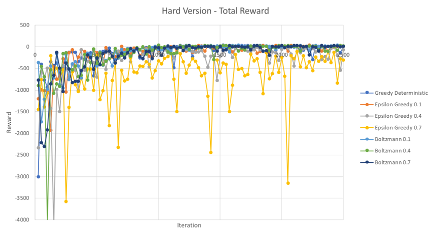

Figure 14: Hard World Reward

Here, we notice that in both problems, the greedy and Boltzmann strategies performed similarly well. While all performed quite well, those two strategies converged more quickly and showed fewer outliers, especially compared to epsilon greedy. However, when looking at the numbers, it is clear that none of them come to an optimal policy like the previous algorithms did. The results were suboptimal by a reward value of 3-8 lower than the previous algorithms. These differences will likely become less important as the overall reward and number of states being considered increases. Further, the more conservative Boltzmann temperature values performed better than the liberal ones, indicating that a *more* greedy approach is quite effective.

Now, time taken is considered:

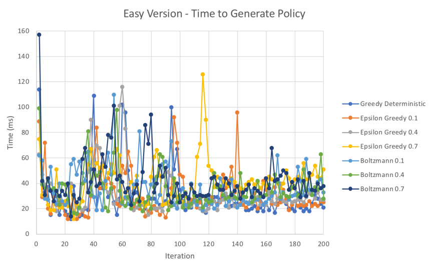

Figure 15: Easy World Time Taken

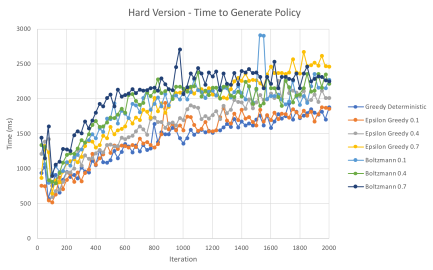

Figure 16: Hard World Time Taken

The most clear difference here is between the time taken in the easy world versus the hard world. It takes approximately 10x as long on the hard world. This differential is smaller than what was seen in the previous algorithms (such a policy iteration), which seems to indicate that Q-learning would scale more readily to larger problems. Unfortunately, the convergence point is many more iterations (2-3x) than in the previous algorithms. The graphs also show that the greedy scheme is the fastest, which is reasonable considering how the algorithm operates. On the other end, the Boltzmann scheme is the slowest, and is reasonable for the same reasons. Since Boltzmann and greedy perform similarly in regard to correctness and speed to convergence, the one with faster iterations is preferred: greedy. This algorithm produces the following policies at the end of the iteration cap:

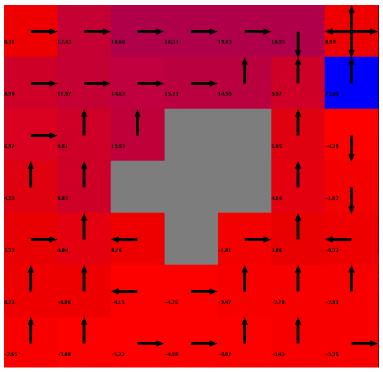

Figure 17: Easy World Greedy Policy

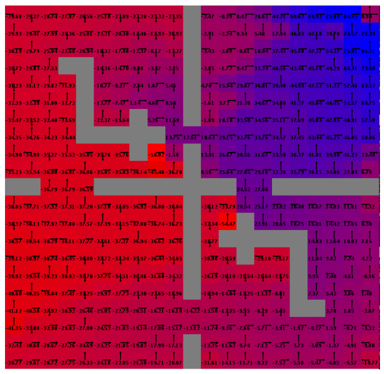

Figure 18: Hard World Greedy Policy

Boltzmann produces:

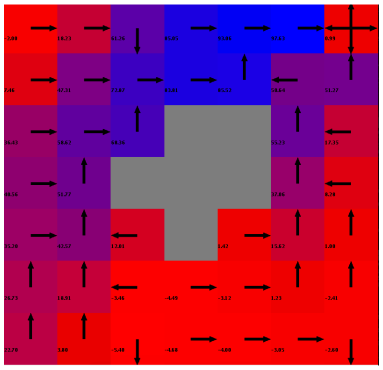

Figure 19: Easy World Boltzmann Policy

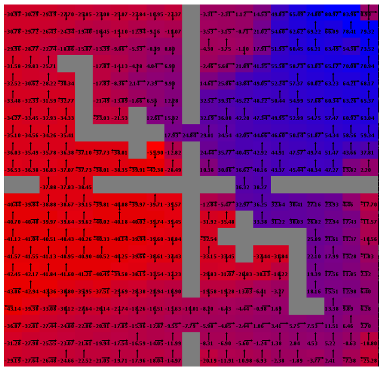

Figure 20: Hard World Boltzmann Policy

For the grid world problems, it appears that the extra complexity from epsilon greedy and Boltzmann were not needed: the simple greedy algorithm performed the most correctly in the shortest amount of time. However, outright, the greedy algorithm still performed more slowly than the algorithms in the previous section.

Conclusions

In general, Q-learning was the slowest algorithm that was tested; it also produced a suboptimal policy. However, this is to be expected. Since there is no transition model for Q-learning, it is rational that it would take longer to produce results: it has to learn a model. The trade off for the time increase is the fact that we can solve MDPs when such model does not exist, which is often the case in the real world. Value and Policy iteration take the advantage of knowing the model in order to converge to a better policy in a shorter amount of time. Knowledge about the MDP that is being solved (domain knowledge) can influence which algorithm is preferred. In the case of Q-learning, it can also inform the exploration strategy that is used.

References

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/