Assignment #4: Markov Decision Processes

Connor Bohan – connor.bohan@gmail.com

MPD #1: Doors and Keys

Description

The first MDP is based on the grid world example. It is a 15x18 tile world with four rooms. The first two rooms are connected while the third and fourth rooms are blocked off by doors. The second room has two keys in it which the agent can pick up and use to open the doors. Both keys can unlock both doors, so the agent can get the keys in either order. The agent is trying to get to a chalice in the corner of the fourth room. The agent has four possible actions in each state: north, east, south, and west. Moving onto a space with a key picks up that key. Moving onto a space with a door while the agent has a key opens the door and consumes the key. According to Burlap's reachability analysis, there are 1035 reachable states.

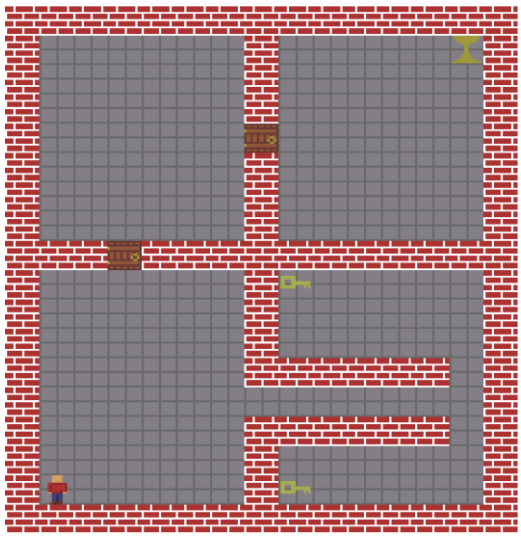

Figure 1: The visualized representation of the Doors and Keys MDP problem. The agent is in the bottom left and the chalice is in the top right.

Part of what makes the Doors and Keys MDP interesting is that it adds a layer of complexity on top of grid world. Both keys can be in one of three state: on the ground, on the agent, or consumed. The doors can be either open or closed. So, while the number of open grid spaces is fairly small, there are actually 36 times that many states (3 states for the first key * 3 states for the second key * 2 states for the first door * 2 states for the second door). The second interesting thing about this MDP is that it has many unreachable states. For example: its impossible to be anywhere past the first door while both keys are still on the ground. I'm not sure what impact, if any, this will have on policy iteration or value iteration. Another interesting characteristic is that there are several ways to get to the reward. The optimal way is to collect both keys first and then unlock both the doors. However, the keys can be collected in either order.

Value Iteration

I decided to test the Doors and Keys MDP over multiple levels of stochasticity (i.e. the probability that performing an action will result in the intended outcome). When there is no stochasticity, value iteration requires 31 iterations to converge. Convergence took .42 seconds on my machine. When I decreased the probability that the agent would move in the correct direction to 50% (i.e. there is a 50% chance the agent will move in the correct direction and a 16.6% chance the agent will move in any of the other three directions) it took 136 iterations or .98 seconds to converge.

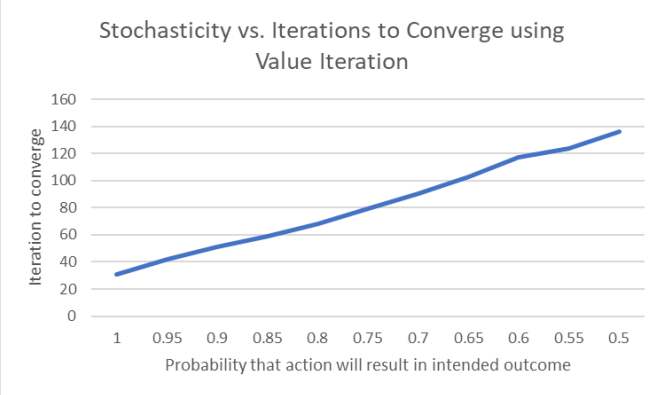

Figure 2: The number of iterations that value iteration needed to converge compared to the probability that the agent moves in its intended direction. As the probability decreases, the number of iterations required seems to increase linearly.

Due to the high dimensionality of this problem it is a little difficult to show the policy that value iteration decided upon. I've decided to show four different 'slices' of the policy. One slice shows what the agent will do when it has zero keys, there is two slices for when the agent has just one of the keys, and finally there is one slice for when the agent has both keys. An important thing to note is that the way the policies are rendered is flipped vertically from how the visualizer renders the room. In the visualizer the starting room is on the bottom left, while on the policies visualizations is rendered on the top left.

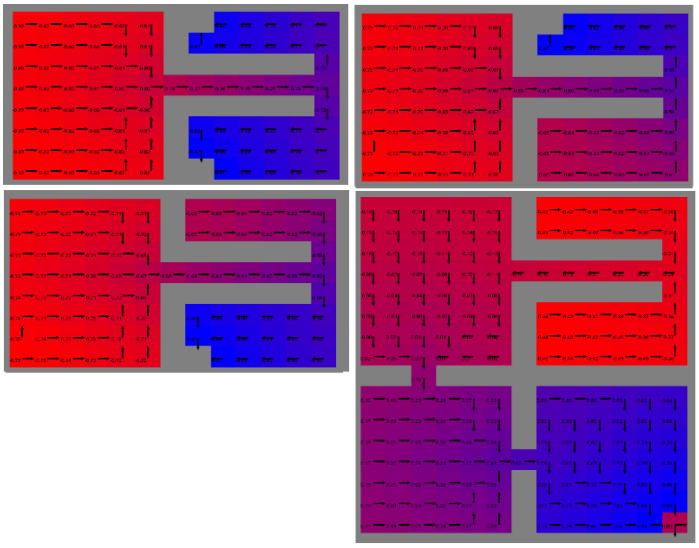

Figure 3: The top left image shows the policy when the agent has zero keys. The top right and bottom left images show the policy when the agent has one key, and the bottom right image shows the policy when the agent has both keys.

From these policy visualizations, we can see that the policy that value iteration settled on was to go get both of the keys and then go to the reward.

Policy Iteration

Policy Iteration worked… weirdly on this MDP. I made sure to keep the discount factor and max delta the same between the value iteration and policy iteration tests. With zero stochasticity, policy iteration took 33 iterations to converge compared to 31 of value iteration. Based on my understanding, policy iteration should always converge in fewer iterations than value iteration. I'm going to chalk this one up to the black box nature of Burlap's value and policy iteration methods (possible because VI is updating values in place in a random order?). Another odd characteristic of policy iteration is that it converges in fewer iterations the more stochastic the world is.

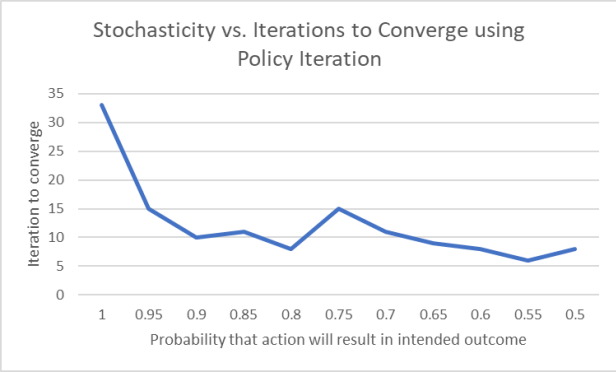

Figure 4: The number of iterations that policy iteration needed to converge compared to the probability that the agent moves in its intended direction.

One of the reasons that policy iteration took fewer iterations to converge when agent movement was more stochastic was that the inner value iteration loop took longer. The inner loop on took around 15 iterations on average when there was no randomness to the agent's movement. When there was high randomness in the agent's movement it took around 100 iterations to evaluate the policy. Policy iteration was much slower in clock time that value iteration. It took as little as 4 seconds and as many as 7 seconds to converge on my computer.

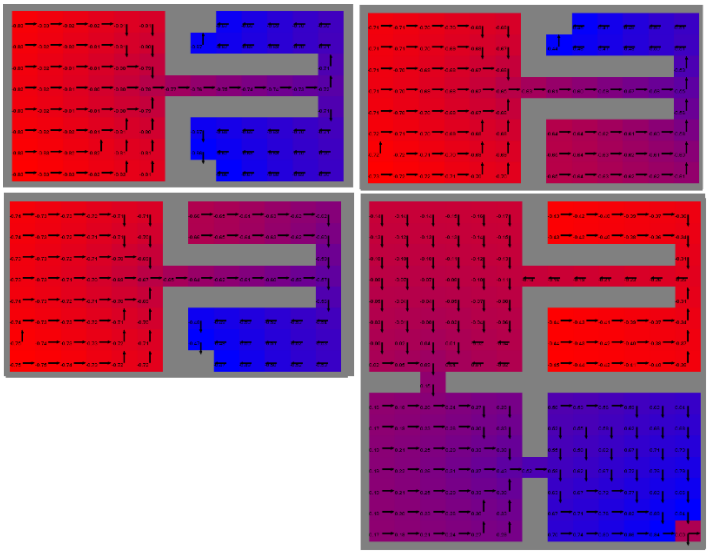

Figure 5: This figure shows the policies that policy iteration chose for when the agent has zero keys, one key, and two keys.

Policy iteration chose a similar policy to value iteration. It decided to get the other key first and there are a few areas where it chose a slightly different, equally valid path through some of the rooms. Overall, I don't see any advantage to using policy iteration for the Doors and Keys MDP.

MPD #2: Frozen Lake

Description

The second MDP is the less complex of the two, with only 100 reachable states. It's a variation on the Frozen Lake problem. I designed my version to have three obvious paths the agent could take. The first path gets the agent to the goal the quickest but is the most dangerous, the second is less dangerous but slower, and the third is the least dangerous but slowest. The reason I did this was to be able to test and compare how value iteration and policy iteration perform with variable stochasticity and decay values. When the agent is better able to control its own movement, I expect it to pick the fastest path. And when it is less able to control the direction of its movement, I expect it to pick the safest path. Also, as the gamma value decreases, I expect it to pick faster paths. I'm interested to see how value iteration and policy iteration respond (in terms of computation time) to changing agent stochasticity and different gamma values.

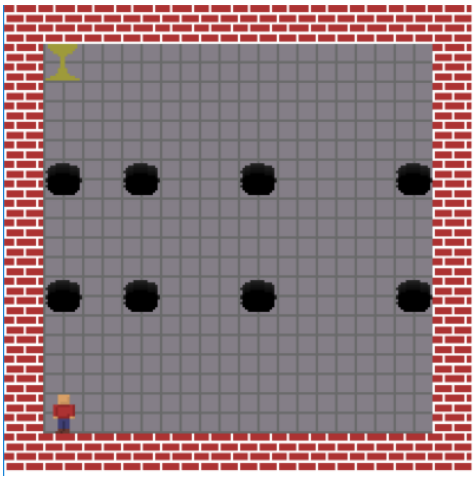

Figure 6: The visualized representation of the Frozen Lake problem (I know it doesn't look much like a Frozen Lake). The holes are worth -1 points and the chalice is worth 1 point. There are three paths the agent can take. In the first path, the agent will be surrounded by holes on both sides. In the second path the agent will only be next to one hole at a time. In the final path, the agent will have room for error on both sides.

Value Iteration

I started with a gamma value of .99 and a 100% chance that the agent would move in its intended direction. The results were what I expected. The agent chose the shortest path. From there I begin reducing the chance that the agent would move in its intended direction. At a 93% chance that the agent would move in its intended direction, value iteration decided that the second path was the most optimal. Finally, at 86%, it chose the third path.

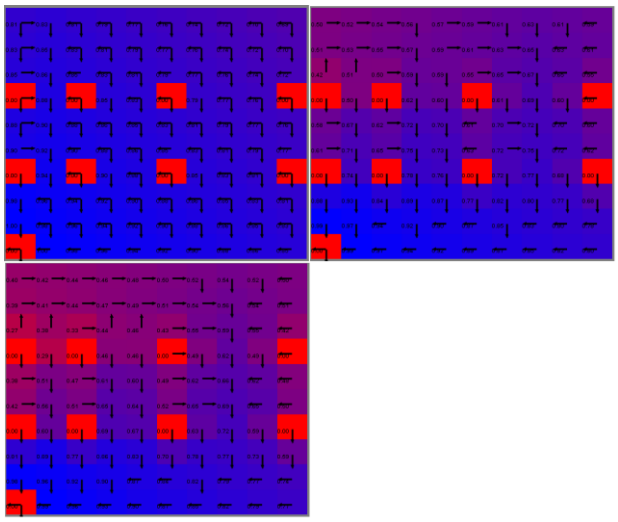

Figure 7: Top-left: 100% agent movement accuracy, top-right: 87% agent movement accuracy, bottom-left: 80% agent movement accuracy. You can see that as agent's stochasticity increases, the agent chooses safer paths.

Once again, the number of passes before convergence is roughly linearly proportional to the stochasticity of the agent. Convergence took, on average, around .245 seconds.

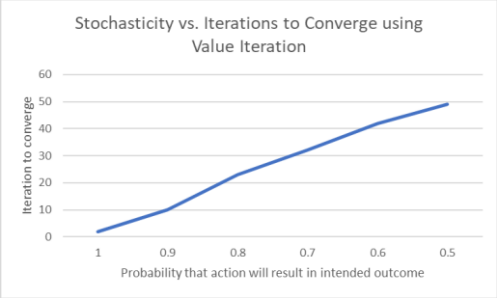

Figure 8: Iterations to converge compared to the chance the agent will move in the direction it intends. With 100% movement accuracy, value iteration converges in just two passes. I think this is possible because Burlap's value iteration algorithm starts from the goal and flood-fills values outward instead of each pass only being able to see values from the previous pass.

Next, I kept the movement accuracy value of 80% and altered the gamma value. Small changes towards the upper end of the gamma range (.95 - .99) tended to have large impact on the resulting policy. With a gamma of .99 the agent chooses the safest route. With a gamma of .97 the agent chooses the middle route. However, I couldn't find a gamma value that caused the agent to choose the fastest route. Interestingly, once the gamma gets low enough (around .7), the cost of getting near the holes overpowers the reward of getting to the finish. The agent will simply try to stay as far away from the holes as possible even though it is a sub-optimal policy. This illustrates the importance of picking a suitably high gamma value.

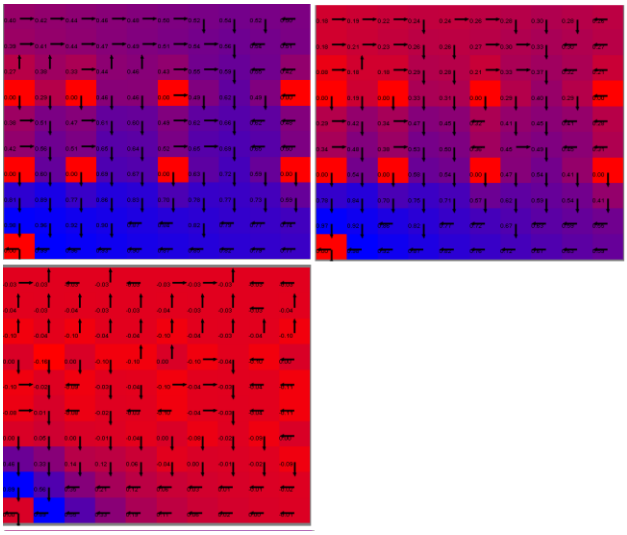

Figure 9: Top-left gamma = .99, top-right: gamma = .97, bottom-left: gamma = .7

Policy Iteration

Policy iteration produced policies that were identical to the ones produced by value iteration. Unlike with the doors and keys problem, policy iteration was not substantially slower than value iteration. It tended to finish in roughly .35 seconds (part of the difference could be because Burlap's policy iteration prints out more information to the console than value iteration).

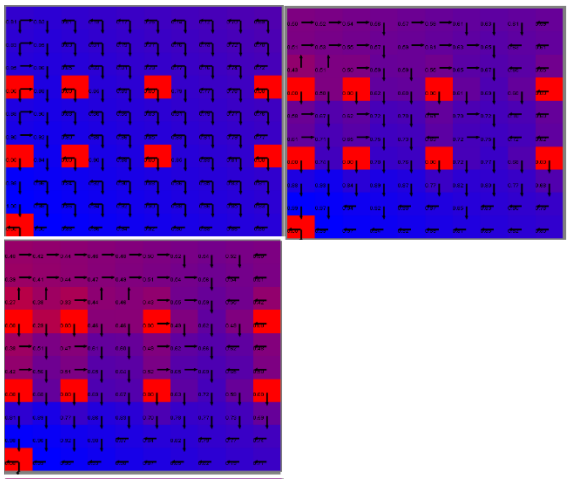

Figure 10: Top-left: 100% agent movement accuracy, top-right: 87% agent movement accuracy, bottom-left: 80% agent movement accuracy. The policies selected by policy iteration appear to be identical to the ones selected by value iteration.

Again, policy iteration does not suffer as much as value iteration when agent stochasticity increase. Regardless of how random the agent's movement is, policy iteration tends to converge in very few iterations. However, this does not mean it takes less clock time complete. From what I can tell, it seems like value iteration does much better (in terms of clock time) than policy iteration for large MDPs (like the doors and keys problem) and in smaller MDPs they do about the same.

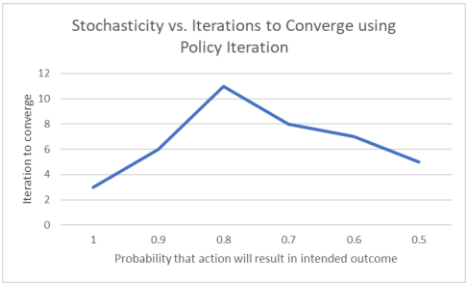

Figure 11: Iterations to converge compared to the chance the agent will move in the direction it intends. There is no obvious trend with policy iteration like there is with value iteration.

Q-Learning

Frozen Lake

The reinforcement learning algorithm I chose was Q-Learning, mostly because it was the one that I understand the best. When setting up a Q-Learning agent on the frozen lake problem, I decided to set up conditions where the best policy would be to take the middle path. To do this, I set the probability

that the agent would move in its intended direction to 87%. I wanted to see if the Q-Learning agent would be able to figure out that going through the middle path provides a slightly higher expected reward than taking the long path. I figured that the agent would not learn to take the fast path because, given its random movement, it is very unlikely to make it through that way. To start with, I set the initial Q-value for every <state, action> pair to 0.5. This was to discourage risk taking. If the agent falls in a hole using a given <state, action> pair, it won't want to do that again because everything else has a slightly positive value. I also set the learning rate to 0.5. I didn't have a good rational for this. The value must be in the range 0-1, so I chose the middle. I hoped this would be a good balance between learning quickly and not getting stuck performing actions that lead to success early on. The agent did not learn to take the middle path. Instead, it decided on the safer but longer path. Looking at what the agent did early on, it seems that it was discouraged by some early failures while going down the first two paths. Neither changing the learning rate nor the initial Q values affected that path that the agent ended up using. However, when using a high learning rate, the agent tried using the middle path

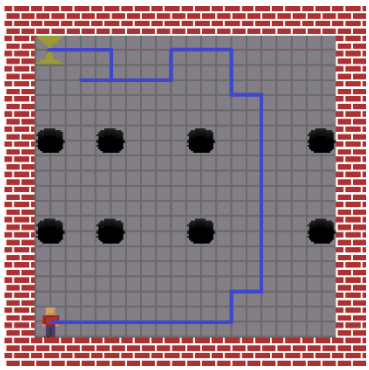

Figure 12: Agent's path after 1000 episodes of learning. There were still some inefficiencies by this point but it had been taking the third path for almost 800 episodes.

Doors and Keys

For the doors and keys MDP, I set the probability that the agent would move in its intended direction to 75%. I thought that would help the agent explore new states while still giving it enough agency to learn quickly. I set the initial Q values to 0 since there are many steps required to solve this MDP. I didn't want a non-zero Q value to be higher than the Q values that the agent learns along the way. I set the learning rate to .5 because that worked fairly well for the frozen lake MDP. Due to the relatively high number of states, it took a very long time for the Q learner to figure out how to solve the doors and keys problem. After about 300 iterations (which took the better part of an hour) it finally found a mostly-optimal solution. The learner decide that the best course of action was to get both the keys (bottom first then top) and then go through both the doors. This is the same policy that value iteration and policy iteration decided on. There isn't much interesting to say about reinforcement learning on this MDP. Since getting both keys and then opening both doors is, by far, the fastest way to solve the MDP, it was inevitable that the Q learner would eventually find that policy. And since there are no ways for the agent to fail (receive a negative reward) the Q learner could not be more risk averse than policy/value iteration like it was with the frozen lake MDP.