**Victor Ma**

**CS-7641**

**HW #4 – Markov Decision Processes and Reinforcement Learning**

**INTRODUCTION**

In this assignment, we design and implement two interesting Markov decision processes (MDP), one with a “small” state space and the other with a “large” one. We solve both MDPs using value iteration, policy iteration, and Q-learning. Then we examine and compare the results.

**TWO MDPs**

**1. Stock Trading problem.** Based on a simplified notion of an investment portfolio for the stock market, the Stock Trading Problem describes the following scenario. You have a pile of cash that you want to increase through investment. Your plan is to invest a fixed amount of money into a single stock at a time. When should you buy, sell, and hold?

As we define the MDP, there are N + 1 states, where N is an odd positive integer. The first state (0) corresponds to holding no stock. The remaining N states (1 .. N) correspond to N different and ordered values of a stock including the "base" value at which you purchase it, which is state (N+1)/2. In other words, state 2 is more valuable than state 1 but less valuable than state 3. Due to a hard constraint on N, this stock has a minimum loss and maximum gain.[[1]](#footnote-1)

There are three actions: buy, sell, and hold. Buying transitions from the stock-less state to the "base" value stock state with 100% probability. Selling transitions from any stock state to the stock-less state with 100% probability. Upon holding, a stock value will change to one of its adjacent values, either higher or lower, at each iteration with equal probability, i.e. 50% each. It cannot remain the same value.

The reward for buying a stock is -1. Next, the reward for a selling a stock is equal to α raised to the power of Δ where Δ is the current state minus the base state. For example, if base state is 4, and the stock is currently in state 6, i.e. a higher value, then the value of selling this stock is α^(6-4) = α^2. Finally, the reward for holding a stock is β, which is negative. This corresponds to the so-called opportunity cost of investing in stock rather than something safe but low gain, such as bonds or a bank account.

**Implementation**. In our implementation, N=7, α = 2, and β = -0.05. This is our "small" state space problem.

**Why is it interesting?** This is an example of a small state space problem whose solutions will be easy to verify, and possibly even figure out easily manually. It is a related to a real-world problem in an industry where having a better solution is extremely lucrative. The small penalty has a meaningful interpretation, that of an opportunity cost, which is something investors have to consider when deciding how to use a finite source of money. Finally, this problem can be easily extended to multiple stocks with arbitrary values and transitions.

**2. Theseus and the Minotaur**. In Robert Abbott's "Theseus and the Minotaur" puzzle[[2]](#footnote-2), the hero and king of Athens, Theseus, must escape the labyrinth while avoiding the Minotaur. The Minotaur moves two steps for every one which Theseus; however, it has a set of rules it must follow, e.g. it tries to move closer to Theseus, it will move horizontally before moving vertically, etc. Knowing these rules and the layout of the labyrinth allows Theseus to evade his pursuer and escape.

The Maze MDP that we have implemented is inspired by this puzzle. We have designed a simple maze and chosen initial positions of Theseus and the Minotaur. We consider our state space to be the set of all possible pairs of positions of Theseus and the Minotaur. What's different is that the Minotaur now moves only one step at a time and randomly in each direction (north, south, east, west) with equal probability. Two things are accomplished by this change:

1) Chance is incorporated into the MDP as a result of Theseus' movement, whereas in Abbott's puzzle, all state transitions are deterministic given Theseus' action.

2) We now have an example of a MDP with a "large" state space. If there are N possible positions, then there are N^2 states.

In our formulation, there are four actions for Theseus: north, south, east, and west. There is no option to stand still, per se. As mentioned earlier, Theseus moves deterministically while the Minotaur moves randomly. If either moves into a wall, they will bounce back and remain in the same spot. As a special transition, when Theseus meets the Minotaur (which can be viewed as a moving end position) or when he reaches the exit, both with automatically return to their initial position. Consequently, the game resets.

There are three rewards. If Theseus meets the Minotaur (shares the same position), he receives a penalty (or negative reward). If he successfully escapes by reaching the goal position, he receives a positive reward. Finally, for any step he takes which does not result in either of these two situations, he receives a small step penalty, to encourage him to escape sooner and/or not die of starvation.

**Implementation.** In our implementation, we designed an 8 x 7 grid world with some obstacles. There are therefore at most (8 x 7)^2 = 3136 states, though not all of these states are reachable since they correspond to situations where at least one of the two agents is occupying the same position as and obstacle. If we only include states that are “legal”, then there are 1225 states in total.

The reward is +1. The penalty is -1. The small step penalty is 0.01.

**Why is it interesting?** This problem demonstrates a “large” state space problem as well as a grid world problem. In this MDP, the state space grows not only with the size of the grid world itself, i.e. the number of cells, but also with the number of (artificial) agents in it. In this case, the addition of the Minotaur increases the number of states from N to N^2. The addition of other agents would increase it even further. This problem could be extended to imitate the Pac-man game, where the goal is to avoid numerous ghosts.

Unlike the Stocks problem, this MDP simulates the presence of an adversarial agent, though this is still short of a true game theory problem since the Minotaur is acting randomly. In other words, it is not a rational agent. This problem has both positive and negative reward terminal/absorbing positions like the grid world problem; however, a key difference is that the negative position here (the Minotaur) is actually moving! Another key difference is that the agent’s (Theseus’s) actions themselves are deterministic while the environment is not. Although the scenario is inspired by a Greek myth and a puzzle, the problem is related to that of navigation and avoidance. It has therefore applications to, for example, self-driving cars which need to reach a certain destination while avoiding other moving objects, e.g. other cars.

**IMPLEMENTATION**

The two MDPs, including both transition and reward matrices, were implemented and solved using Python, the NumPy module for matrix manipulation, and the PyMDPToolbox module for its value iteration, policy iteration, and Q-learning implementations. Both MDPs are treated as an infinite process, rather than episodic. Thus, for the maze problem, the cumulative reward is counted over multiple runs. Nonetheless, performance of a policy is calculated as the average reward per action, so this difference will not be material.

**VALUE AND POLICY ITERATION**

Using a discount rate of 0.9, we ran value and policy iteration on both MDPs. The performance of the resulting policies was the average reward over 100K transitions in a simulation while following those policies.

Below, we show the performance results on the Stocks MDP.

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **# of iterations** | **Execution time (sec)** | **Average reward** |
| Value iteration | 10 | 4.880e-04 | 1.7520e-01 |
| Policy iteration | 5 | 3.039e-03 | 1.7091e-01 |

The optimal policies from both algorithms were exactly the same: (0, 2, 2, 1, 1, 1, 1, 2), where 0 = “buy”, 1 = “hold”, and 2 = “sell”. Because they were the same, any differences in the average reward metrics are due to chance in the simulation. Interpreted, this policy says to buy a stock when you don’t have one. Then, hold until one of two things happen: 1) Your stock loses value twice from the base value, in our case a 75% drop, in which case you should cut your losses and sell. 2) You reach the maximum value, in which case you should also sell. This strategy seems reasonable since there should be a point at which you’re more likely to gain more by starting with a new stock at the base value than by trying to recover your losses with your current stock and then increasing from there.

What’s interesting is that this is similar to a simple mid-term stock policy from the William J. O’Neil’s book “How to Make Money in Stocks”[[3]](#footnote-3) in which you sell a stock if it drops below 7% of its buy-in value and you should sell if it increases to more than 30%. You choose stocks based on the “CAN SLIM” strategy that may bias the investor towards stocks that behave more like the MDP we have defined. Of course, there is the constraint that we always buy a stock for a cost of 1, and the assumption that we have an infinite reserve of cash. Still, it’s nice to see that a reasonable albeit simplistic trading strategy arises.

Below we show the results on the Theseus and the Minotaur (a.k.a. the Maze) MDP.

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **# of iterations** | **Execution time (sec)** | **Average reward** |
| Value iteration | 51 | 8.711e-01 | 3.8571e-02 |
| Policy iteration | 15 | 1.300e+01 | 3.8821e-02 |

The solution policies for value and iteration and policy iteration differ in only 26 states out of 3136. Out of these 26 states, 13 correspond to “illegal” states, i.e. those where at least one of the Theseus or Minotaur are occupying an obstacle. Because both algorithms converged and their average rewards are so close, it’s likely that there are multiple optima, either global or local. For example, you’re in a corner and you want to stay in the same place. You could walk into one of the two walls, both of which result in the same state. Thus, the policies are not qualitatively different.

For both MDPs, we found that policy iteration converged in many fewer iterations although overall execution time was longer. Given that value iteration converges faster yet results in the same policy, it seems we should prefer value iteration to policy iteration. However, we only tested on these two MDPs, so there may be cases where policy iteration will converge faster.

**Q-LEARNING**

For Q-learning, we used a discount rate of 0.9. The exploration strategy was as follows: at iteration n (from 0 to MAX\_ITER), choose a random action with probability 1/log(n+2), otherwise, take the optimal action according to the current estimate of Q. This strategy balances exploration vs. exploitation by ensuring that there is always a chance to explore, but over time, the learner will increasingly prefer to exploit instead and maximize cumulative reward. Finally, decaying update coefficient is 1/sqrt(n+2). Thus, subsequent updates have less and less impact on the estimate of Q.

Below are the performance results of Q-learning on both MDPs.

|  |  |  |  |
| --- | --- | --- | --- |
| **MDP** | **# of iterations** | **Execution time (sec)** | **Average reward** |
| Stocks | 2.0e+5 | 3.206e+00 | -5.0000e-02 |
| Maze | 5.0e+4 | 5.926e+01 | -1.5378e-02 |

The solution policy for the Stocks MDP from Q-learning was: (1, 2, 1, 2, 2, 2, 1, 2). This is very different from the solutions found by value and policy iteration. Interpreted, this policy suggests “holding” when you have no stocks, i.e., not buying any stocks at all! And thus, the average reward resulting from following this policy is simply the opportunity cost or penalty from holding stocks (0.05) from the definition of the Stocks MDP.

Compared to policies resulting from policy and value iteration, the Q-learning derived policies perform much worse. For example, in the Stocks problem, Q-learning failed to find a strategy that provided a net gain, even though the state space was rather small. We did notice that re-running the algorithm sometimes yielded a policy with a net positive average reward, though it was still not as good as with value or policy iteration. This indicates that the Q-learning algorithm might have fallen into local optima or failed to converge. Given the high number of iterations (and we tried even much more), the latter case seems unlikely. Changing the exploration strategy and update coefficient may lead to better policies. We only tried one other exploration strategy, which was to always have a 10% chance to pick a random action instead of the best action according to the current estimate of Q. This did not seem to change the results much. It could be that simply not having a model of the world as captured by transition and reward matrices makes it that much harder to learn the optimal policy.

On the plus side, Q-learning, on a per iteration basis, was much faster than both value and policy iteration by at least an order of magnitude on both problems. And although its policy was inferior, it was able to learn without knowing the transition and reward matrices.

A compromise approach may improve Q-learning. For example, learn over two phases. In the first phase, we can focus mostly on exploration and modeling the world, perhaps even try to estimate the transition and reward matrices. In the second phase, use these estimates to perform value iteration, which gives us a rough estimate of the values of each state and thus an initial estimate of the Q function. Then, we can proceed with Q-learning as normal.

**CONCLUSION**

We designed and implemented two MDPs. One imitates a stock trading problem, while the other captures a navigation-and-avoidance problem set in a Theseus and the Minotaur scenario. We found that policy and value iteration, through their use of the transition and reward matrices which define the MDP to be solved, greatly outperformed-Q-learning in terms of reward maximization, despite being slower. Unfortunately, in many real life problems it is either not possible or not practical to obtain these exact matrices in the first place. So Q-learning may be the only practical option of all three. That said, it might be possible to make a very coarse estimate of the matrices that can be used to initialize Q for Q-learning.

1. Of course, this is not realistic, but it simplifies the problem and keeps the state space small. [↑](#footnote-ref-1)
2. Abbott, R. (1990). Mad Mazes: Intriguing Mind Twisters for Puzzle Buffs, Game Nuts and Other Smart People. Adams Media, 1990 (<http://www.logicmazes.com>) [↑](#footnote-ref-2)
3. ONeil, W. J. (1988). How To Make Money In Stock: A Winning System In Good Times Or Bad. [↑](#footnote-ref-3)