Christopher Sullivan

Dr. Brian O'Neill

CS-390-02

Due 5/5/2023

Markov Decision Processes: An Algorithmic Analysis

Introduction

Within this machine learning study, multiple tests were conducted using the Python *mdptoolbox* library across two state-transition problems, with two related reinforcement learning algorithms being applied to each problem individually. In order to correctly apply these algorithms, the problems in question were represented as Markov Decision Processes. Briefly, a Markov Decision Process is a logical problem structure in which the problem consists of states, actions, reward values, and a transition-probability model. Within a Markov Decision Process or MDP as they shall be referred to henceforth, the goal is to use probability-control techniques in order to find an optimal policy, or a list of actions per state which will lead the learning agent in question to amass the highest total reward value possible.

The goal of this study was to analyze and compare the findings reported by each algorithm in order to gain understanding of how they operate across differing types of MDP problems. In Python, test functions for both algorithms were created with the aim of finding the optimal policy of each MDP across varying iteration limits, reward discount factors (γ), and numbers of total states. As output, each testing function reports the final policy produced by its' respective algorithm, as well as the number of iterations taken to reach a point of convergence. Using these outputs as metrics, we can analyze the closeness of the reported policies to the optimal policy, as well as the relative efficiency of each algorithm measured in iteration count.

Overall, the structure of the study should prove sufficient in highlighting the individual strengths, weaknesses, and tendencies of each reinforcement learning algorithm.

Algorithms

The two algorithms which were employed in this study are of similar nature, as they are both reinforcement learning algorithms which, given the proper configuration, guarantee convergence on the optimal policy. While both algorithms employ the same underlying formula in order to calculate reward-utility of states, the way in which they produce their policies differs significantly between them.

The first algorithm was the procedure referred to as *Value Iteration*. Value Iteration begins with a set of arbitrary values, commonly referred to as utilities, each representing the overall value of being in a certain state at a given time. Using the Bellman Equation to calculate the overall time-based utility of each state, these utility values are updated upon each iteration until the algorithm converges. In standard Value Iteration, convergence occurs once two serial iterations produce utility sets of which the difference is less than a given threshold of similarity, epsilon. In the case of this study and in typical use, this stopping criterion epsilon is set to a value of 0.01 for all Value Iteration instances. Finally, once the algorithm converges, a policy is constructed using the produced value function and the utilities of each state, wherein the action which leads to the highest future reward is assigned to each individual state.

Furthermore, the second algorithm used within this study was *Policy Iteration*. Like Value Iteration, the algorithm commences by assigning arbitrary values, here representing the action which is to be taken while in each state. Using a slight variation of the previous Bellman Equation, Policy Iteration calculates the updated utility of each state based on the actions that can be taken from said state, and where this action leads the agent. This differs from Value Iteration in that rather than calculating the utility of each permutation of state-action transitions, only immediately available actions are considered, which leads to quicker iteration time, in addition to the possibility of Policy Iteration converging on the global optima in fewer iterations than the optimal value function being found. Thus, while both algorithms are used to generate the optimal policy, the differing methods by which they produce it can lead to various behaviors upon observation in test.

Markov Decision Processes

As previously mentioned, this study was conducted upon two separate MDP problems; the first titled 'Grid World', and the second referred to as 'Forest Management' or simply the 'Forest Problem'. In the first problem Grid World, the goal is for the learning agent to make lateral movements towards a positive reward within a finite 'world' in which there are 11 possible states, each representing one square of the grid. In this MDP, the actions available to the agent are 'Up', 'Left', 'Down', and 'Right'. Pictured below is a visual representation of the state and reward layout used within the Grid World MDP.

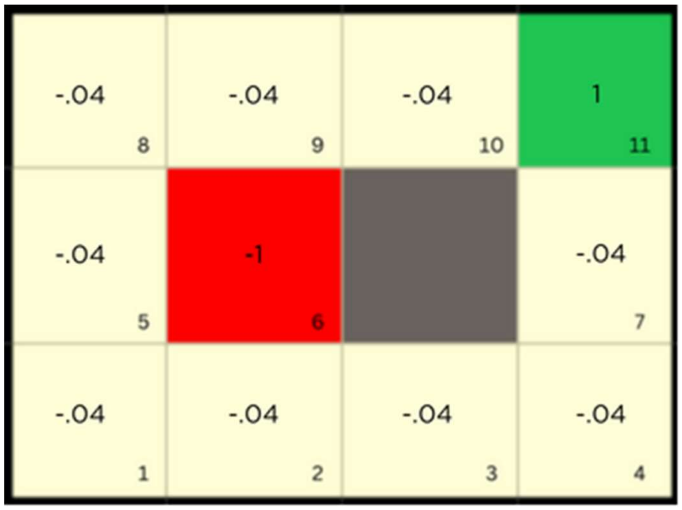
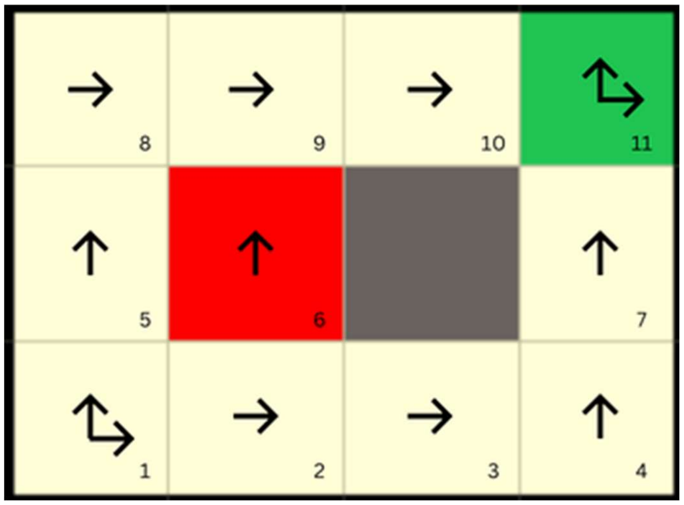
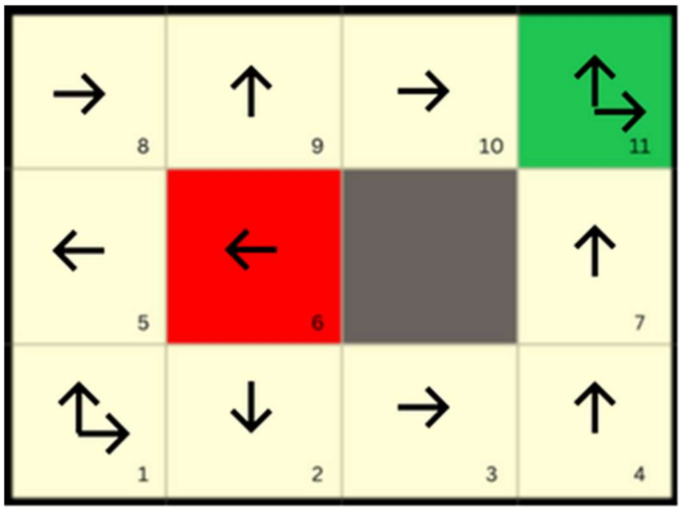


Figure 1.1: *Grid World State, Reward Visual*

Using this layout as the MDP, two transition-probability models were created and tested, each representing a different variation of the agent's movement probabilities. In the first model, the agent's actions are deterministic, and the direction in which they choose to move will always correspond with a transition to exactly one state. Conversely, in the second model, the agent's actions are represented stochastically, with each movement action possessing an 80% chance to lead the agent in the desired direction, and a 20% chance of the movement being rotated 90° in either direction. Thus, using this model, if an agent wishes to make an upwards move, they have an 80% chance of moving up, with a 10% probability of being sent right and a 10% probability of going left. Due to this chance aspect of the second model, the optimal policy must differ from the original in order to account for the probability of 'accidentally' entering the negative state 6. Finally, within this problem model, it is to be noted that moving from a state into an 'outer wall' or into the gray square results in the agent remaining in the same state following the action. This has been accounted for in the transition probabilities of each model by adjusting the percentages accordingly. Pictured below are the optimal policies for each transition-probability model.





Figures 1.2, 1.3: *Optimal Policies of Deterministic, Stochastic Models from Left to Right.*

In addition, the second MDP Forest Management is a problem with *S* number of states representing consecutive years of forest growth. Although this problem was initially difficult to interpret referencing the documentation from *mdptoolbox,* the transition-probability and reward matrices' structures were used to determine the details of the problem domain. Regarding *S*, if the agent is currently considering state 50, this would represent a forest which is 50 years of age. Furthermore, the MDP model of this problem consists of two actions 'Cut' and 'Wait'. Within the transition probability model, each year where wait is chosen, there is a probability *p* that a fire will occur and the state will be reset to state 0. Conversely, if the agent chooses to cut the forest, the state is guaranteed to reset to 0. As deemed by *mdptoolbox*, the default value of *p* is 0.1.

Derived from a goal of balancing forest preservation and profiting off of wood, the base reward model consists of a reward of 4 for choosing 'wait' in the oldest state, a reward of 2 for cutting in the oldest state, and a reward of 1 for cutting in any state besides 0. While this model is valid in forming an MDP, it was deemed to be insufficient during testing, as it frequently led to 'cut' being chosen in every state except for the first and last. During testing, this was only prevented by heavy manipulation of the reinforcement algorithms through raising the discount factor γ to values such as .6 or higher. Nonetheless, the change in results was too minimal to justify the problem being robust enough for algorithm analysis. Therefore, it was found to be necessary to produce a new reward model, in which the agent is more easily influenced to find an equilbrium between waiting while risking fire, and prematurely cutting to minimize loss. For this purpose, the *R* matrix in the *forest()* function of the *example2.py* file was updated and utilized in *main.py*; further numerical details on the changes made to this matrix can be found in the *README*.txt file. In sum, the new matrix favors cutting for the first half of the forest's life cycle, and prefers waiting once the forest has reached half its' peak age, and is closer to being a full-grown, preservable forest.

Grid World

Upon creation of the matrices representing Grid World in Python, the model was ready for evaluation through *mdptoolbox*. In order to simplify the process of matrix creation, each of the two transition models and the reward model were all individually hard-coded and stored in the *grid_world.py* file. Once this was finished, each model was passed to the unmodified ValueIteration() and PolicyIteration() functions of *mdptoolbox*, and the results observed.

Value Iteration:

To begin, Value Iteration was performed on the deterministic Grid World model. As 0.1 is the base discount or time-step factor in *mdptoolbox,* it was used as the initial testing value in each scenario. Additionally, it should be noted that for all relevant tests involving Value Iteration, the stopping criterion epsilon is set to the default value of 0.01, as going any lower than this negatively affects convergence, and does not aid in yielding an optimal policy in any way. Although it may be argued that incrementally raising this value may aid in quicker convergence, this was deemed unnecessary given the small scope of the study in addition to the lack of results upon attempting.

Upon testing the deterministic model with the initial discount value γ, the algorithm returned a policy of: ['Up', 'Left', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Right', 'Up'] ordered by state index. This represents the correct action being taken in 7 of the 11 states, with states 2, 3, 8, and 9 being mislabelled by the policy. Although these mislabellings represent a policy which is short of optimal, the agent never takes an action which would direct it into the negative state 6 under this policy, which is a good sign of reinforcement learning taking place. However, state 3 being labelled with the 'Up' action causes the agent to become trapped in an loop here while accumulating small negative rewards, which is certainly suboptimal. Going further, the value of γ was incrementally raised by 0.1, with the action accuracy increasing by one each time until the optimal policy was produced at a γ-value of 0.4. Observing this, it can be assumed that this boost in discount value allowed for the agent to consider further rewards with more weight, and thus look ahead to the goal. However, as the discount rate was increased from 0.1 to 0.4, iteration count also increased from 2 to 6. While this is minimal, it should be taken into account for problems of a higher scale, as a higher value of γ appears to raise the amount of value resolution required.

Once the optimal policy was found for the deterministic model, the algorithm was then fed the stochastic model and tested. Upon convergence with a γ-factor of 0.1, the algorithm reported a policy of ['Right', 'Down', 'Up', 'Up', 'Left', 'Left', 'Up', 'Right', 'Up', 'Right', 'Up'], representing a near-optimal policy. Here, each state is labelled with the correct action save for state 3, which is labelled as 'Up'. However, raising the discount factor did not simply fix the issue; by raising it to .2 and above, state 9 would begin to falsely report as 'Right', while state 3 corrected itself. Thus, the value was subsequently raised, then further lowered by small fractions. Additionally, due to suspicion of the red state 6's negative reward perhaps not being weighted enough to be avoided by the agent, the green and red square reward values were raised to 2 and -2 respectively. Despite this, the optimal policy was never reported by the algorithm, with 10 states or less being correctly reported each time. Thus, the most efficient use of *mdptoolbox's* Value Iteration function for the stochastic Grid World was deemed to be at γ == 1, where iteration count was at a low of 2. These results beg the question of why the optimal policy was not ultimately found; perhaps the implementation of the algorithm does not handle stochastic models with complete accuracy. Regardless, the function converged quickly with a near-optimal policy, and was thus capable of solving most of the stochastic Grid World with ease.

Policy Iteration:

Following Value Iteration, *mdptoolbox's* Policy Iteration implementation was passed the Grid World models. Foremost, the deterministic model underwent the algorithm. Within one test, the algorithm returned the optimal policy immediately with a γ-value of 0.1. Notably, this policy was achieved in one less iteration than Value Iteration's optimal policy, taking 5 iterations to converge. Given the deterministic nature of the transition model, it is logical that a high γ-value is not needed to find the optimal policy, as the agent can use Policy Iteration to consider its' immediate available moves without much calculation as to where it may end up by chance. This speaks to the ability of Policy Iteration to converge early on the optimal policy and return it. Since the optimal policy was reported immediately, not much further testing was needed. While the γ-value was lowered further in an attempt to produce a sub-optimal policy or otherwise new result, the algorithm continually reported the correct policy. Thus, it can be said that Policy Iteration handled this determinstic MDP extremely well, as was expected prior to testing.

Finally, Policy Iteration was used in order to calculate the optimal policy of the stochastic Grid World model. As with the previous model, and dissimilar to Value Iteration's handling of the stochastic world, the algorithm converged on the optimal policy within one test, using the default reward discount rate. While it took one more iteration than Value Iteration to converge at 3, this is a highly insignificant sacrifice given that each function returns instantly with this problem, and considering the boost in policy accuracy. Overall, Policy Iteration easily solves this MDP problem, with far less trouble than Value Iteration. However, given the robust nature of these algorithms in solving MDPs, the high performance of each checks out. One reason why Policy Iteration may have handled the stochastic model better is the inclusion of the epsilon variable in Value Iteration, which may have conversely affected the handling of the policy production once it converged.

Forest Management

Secondly, the next problem to be solved by each algorithm was the Forest Management problem. Unlike Grid World, this problem was supplied in the *example* module of *mdptoolbox*, and is scalable in its' number of states using the parameter *S*. Thus, while testing both algorithms, tests were conducted over *S* values of 50, 75, and 150 in order to provide a state space which is larger than Grid World for the sake of performance comparison. As previously discussed, the base model was deemed insufficient and thus while testing was done on it, the analysis was performed solely using the revised model.

Value Iteration:

To begin, Forest Management was conducted using Value Iteration. Initially, using a fire probability *p* of 0.1 and the same value for γ, it was observed that as the number of states increased, so too did the proportion of time which would be spent waiting for forest growth. Suppose that S' represents the state in which cutting is no longer optimal and waiting begins. When *S* = 50, S' = 48. With *S* = 75, S' = 56. Finally, when *S* = 150, S' = 76. However, the state in which preservation began did not always increase concurrently with *S*. For example, for both *S* = 75 and *S* = 100, S' remained as state 56. This represents the aforementioned equilibrium being reached by the MDP, which was the goal of creating the revised reward model.

Of note is the effect had by both percentage parameters on this function. Primarily, the parameter *p* is inversely proportionate to S', which checks out mathematically due to the nature of *p*'s effect on the transition-probability matrix. As the chance of wildfire rises, the willingness to preserve a forest goes down, and thus rewards are maximized by simply cutting for wood even while a forest is deep into its' lifecycle. Contrarily, as γ increases, so does S', as the weight of the agent's consideration for future rewards is raised. When both are at high values greater .5, *p* appears to take slight precedence, as it resets the state to 0.

As expected for Value Iteration, iteration count did not increase as *S* did, as convergence does not rely on the number of states but rather the similarity of produced utilities. Across all calls, the algorithm took 4 iterations to converge on a policy. Due to the large state space and probabilistic nature of the algorithm as well as its' varying policies as a function of *S*, calculating the optimal policy across all tests and parameter permutations was beyond the scope of this project. Thus, results for this problem are analyzed comparatively between the two algorithms.

Policy Iteration:

Secondly, the Forest Management problem underwent Policy Iteration. While testing this algorithm on Forest Management, the results produced were practically identical. Across all permutations of *S, p,* and γ, both algorithms reported policies which were exactly alike, or rather produced equal values of S'. This leads to the belief that the algorithms are both reporting optimal policies, as it appears that nothing has interfered with the convergence of either one individually. Differing from Value Iteration, however, Policy Iteration converged in two iterations when faced with this problem, concluding that it solves Forest Management quicker than its' counterpart. Ultimately, it is not surprising that these two algorithms performed so alike, as they possess the same goal, contain the same underlying formula, and guarantee the same result when used properly. Thus, the results of the study through analyzing Forest Management are consistent with prior expectations.

Conclusions

Overall, it was obtained that Policy and Value Iteration, while different in their methodology, are highly similar in their output and usages. Furthermore, while it appears by observance of the study that Policy Iteration may handle stochastic models better, this is something which would need to be further analyzed in order to be proven. As future diligence, it could be suggested that the reward values of Grid World be modified for optimal convergence, or the Forest Problem be represented with a more circular, continous Markovian problem in order to account for the effects of actions in the oldest state. However, these are topics for another study, which may be revisited in the future.