Policy Iteration Simulation for Markov Decision Processes

Malav Dave
malavdave@lewisu.edu
CPSC-57200-001, Fall 2020
Artificial Intelligence 2
Lewis University

I. Introduction

The purpose of this paper is to analyze the results obtained from the environment simulation conducted for this assignment. A robot world environment simulator was designed from existing code sourced from GitHub and tested using custom code [1]. The goal was to compare the values obtained from manual calculation of expected utility for each state (using modified policy iteration) versus the expected utility calculated by exact policy evaluation. The environment was assumed to be fully observable, and therefore, said to have been following a Markov Decision Process (MDP).

The rest of the paper is organized as follows: Section II discusses the methodology used to solve this problem. Section III presents and discusses the results and Section IV provides a conclusion and scope of further testing.

II. Methodology

A. Tools Used

The tools used to do the policy iteration simulation was done entirely using *Python 3.8*. The coding environment used was *Jupyter Notebook*, an interactive web application software that functions as a python IDE. The bulk of the code had already been provided in the GitHub repository [1]. There were many dependencies for this program, including: *random, numpy, defaultdict,* and *utils.py*.

B. Summary: Policy Iteration for Markov Decision Process (MDP) Approach

This section will describe the various parts of the code in sequential order.

There were many classes in the *mdp.py* file (sourced from GitHub) that attempted to model the MDP pattern, as well as other things like turning a world into a grid, modeling Partially Observable Markov Decision Processes (POMDPs), etc. There was an inheritance structure that was followed throughout the code; however, it is not conducive to the analysis that this paper will divulge in, therefore it will not be covered any further.

The code that will be discussed here was quite simple and easy to understand. There were two environments that were tested for comparing the expected utilities with that given by the policy iteration algorithm. First, it was tested on a 4 x 3 world and then on a 5 x 3 world. The algorithm that calculated the expected utilities of each state was condensed into one function that took in four parameters: the MDP environment represented as a grid, the optimal policy yielded by the policy iteration algorithm, the starting state, and the terminal states.

The algorithm worked in 5 simple steps. Firstly, the orientation (north, south, east, west) of the robot was determined for the current state. The robot was then moved by simulating the movement using the *choices* function in the *random* package. This package allowed to simulate the three moves the robot could do (go forward, turn right, or turn left) as a weighted probability distribution, with the probability of each move as its associated weight. The robot was then moved in the simulated direction after orienting it according to the optimal policy's indication. The utility for that state was calculated and added to the cumulative utility.

This process was repeated until the robot had reached a terminal state. The simulation was run 100,000 times and then the average of all the expected utility values from these trials was returned. The resulting expected utility was then compared with the utilities for that state, as calculated by the policy iteration algorithm.

This algorithm is based on the simplified version of Bellman's Equation (presented in equation 1). The approximation of Bellman's equation is presented in equation 2.

$$U_i(s) = R(s) + \gamma \sum_{s'} P\big(s' \big| s, \pi_i(s)\big) U_i(s') \qquad (1)$$

$$CU_s = \sum_0^i CU + \gamma^k \cdot R(s) \qquad (2)$$

Bellman's equation tells us that the utility of a state is measure by the reward of being at the current state plus the discounted Utility of the previous states (represented as a weighted average). In equation 2, the only thing that is different is that we are not taking into account the probability of the state given a particular policy and the action executed in that particular state, because this approximation works only for the optimal policy. It already is evaluation the optimal policy and therefore knows which action it should take. The noise (in probability) associated with the robot's movement is accounted by simulating the movements as a weighted probability distribution, with the help of the *choices* function in the *random* modoule.

III. Results And Discussion

It can be said that the results came out to be very good. The expected utility values that were calculated were very close to the true utility values. The results from the different experiments will be described in more detail in this section. The results will be discussed in the following manner: firstly, the true utility value table (obtained from policy iteration) will be displayed, followed by the expected utility value table that was calculated. A general discussion of how close the values of the two tables were, will then be given and finally, the run times of each experiment will be presented.

Before a thorough discussion of the results is commenced, it should be noted that the rewards used for each state were constant throughout the two different sized state spaces. All states except the terminal states had a reward value of -0.04. The terminal states had a value of either +1.0 or -1.0. The gamma value was 0.9 for both state spaces.

A. *Results: The 4x3 (column x row) Robot World*

For this robot world, there was one obstacle at (1, 1). The coordinates are listed in (column, row) format and it should be noted that they are indexed from 0 onwards. There were two terminal states at (3, 2) and (3, 1).

The true utilities for the optimal policy are shown in table 1 below, followed by the calculated expected utility values shown in table 2.

TABLE I. True Utility Table For the Optimal Policy (4x3 World)

0.5094	0.6496	0.7954	1.0
0.3985	None	0.4864	-1.0
0.2965	0.2540	0.3448	0.1299

TABLE II. Average Expected Utility Table (4x3 World)

0.5093	0.6497	0.7945	1.0
0.3979	None	0.4840	-1.0
0.2958	0.2546	0.3456	0.1268

As it can be seen from tables 1 – 2, the average expected utility values are very close to the true utility values. Since the policy that these values are calculated for is the optimal policy, if the simulation was run for even a higher number of trials, say 1 million, then the expected utility values would be even closer to the true utility values than what can be seen in tables 1 - 2. In other words, it can be said that the expected utilities approach the true utilities yielded by the execution of the optimal policy.

B. *Results: The 5x3 (column x row) Robot World*

For this experiment, the robot world had two obstacles at (1, 1) and (3, 0). There were two terminal states at (1, 0) and (4, 2).

The true utilities for the optimal policy are shown in table 3 below, followed by the calculated expected utility values shown in table 4.

TABLE III. True Utility Table For the Optimal Policy (5x3 World)

0.5229	0.4103	0.4999	0.4019	--1.0
0.6651	None	0.6483	0.5088	0.2518
0.8130	1.0	0.8114	None	0.1723

TABLE IV. Average Expected Utility Table (5x3 World)

0.5230	0.4101	0.4997	0.4014	-1.0
0.6652	None	0.6478	0.5085	0.2523
0.8132	1.0	0.8118	None	0.1732

Again, the average expected utility values are very close to the true utility values. However, it can be noticed that it is closer for some utility value than for others. It can be speculated that the states that are farther away from the terminal states have a bigger difference between the expected utility and true utility values than those states that are closer to the terminal states. This seem to be only the case for state whose reward value is +1.0, not -1.0.

C. *Time Taken for Policy Iteration of Different Environments*

As the name suggests, policy iteration, is an iterative algorithm that enumerates every possible policy that allows a robot to get to a terminal state. The goal of policy iteration is to find the optimal policy that maximizes the expected utility of the robot from any given state. It is intuitive to think that as the size of the robot world increases, the time it takes to complete the process of policy iteration also increases.

This was the case when three different sized robot worlds were experimented with to note the time it took for policy iteration. The results are shown in table 5.

TABLE V. POLICY ITERATION TIME COMPARISONS

Robot World Dimensions	Average Time Taken to Finish Policy Iteration
4 x 3	2 milliseconds
5 x 3	4.99 milliseconds
5 x 5	9 milliseconds
5 x 8	16.5 milliseconds

It can be observed from table 5 that as the dimensions of the robot world increase, the time it takes to finish policy iteration also goes up. From the 4 experiments conducted, it seems the relationship could be exponential, and intuitively, this makes sense because as the dimensions increase, the number of policies to iterate increases exponentially. There is a caveat to consider when measuring running time for policy iteration: the time it takes to iterate the policies for the different environments is entirely dependent on how powerful the processor is, how many threads and cores are occupied (In other words how much CPU time is allotted to this specific task), the memory, and other specifications of the machine. Therefore, the times in table 5 are averaged over 10 trials to account for the variance.

IV. CONCLUSION AND FURTHER TESTING

In conclusion, it can be said that modified policy iteration (approximation of the exact policy evaluation) works quite well. The estimates of the expected utilities were very close to the true utilities as shown by the exact policy iteration algorithm. It was discovered, however, that the modified policy iteration needs to run averaged over a large number of trials, otherwise, it does not approach the true utility values for a certain state. This indeed makes it a very time-consuming method, if it is considered as an alternative for larger state spaces.

Literature on MDPs indicates that exact policy evaluation can be solved in exactly $O(n^3)$ time [2]. It would be interesting to see what the time complexity of the modified policy iteration algorithm is. After considering the empirical evidence presented in this paper, it begs the question that whether it is solvable in polynomial time.

For further testing, the average time taken to finish policy iteration could be tested further with more examples of different sized state spaces. From those results, it can be analyzed whether there the relationship between the state space and the average time taken to finish policy iteration can be captured by a specific distribution. Another scope of further testing is to try the modified policy iteration with very large state spaces and see if the same amount of accuracy remains as the results presented in table 1-4.

REFERENCES

[1] Norvig. (2020). aimacode/aima-python. GitHub. https://github.com/aimacode/aima-python

[2] Russell, S., & Norvig, P. (2009). Making Complex Decisions. In Artificial Intelligence: A Modern Approach (3rd ed., pp. 645–658). Pearson. http://aima.cs.berkeley.edu/