Markov Decision Processes Report

Lijun Huang  
[lhuang349@gatech.edu](mailto:lhuang349@gatech.edu)

# Abstract

This report will conduct experiments on two interesting MDPs, namely forest and frozen lake problem. The report will introduce each of the MDPs and conduct different analysis based on it using value iteration, policy iteration and Q-learning. The mdptoolbox[1] and OpenAI Gym[2] package was used for the implementation of the algorithm and generate MDPs based on it.

# forest mdp – A non-grid world example

## General MDP description

This is an example from mdptoolbox, according to their description, the set-up of the MDP is as follows: “A forest is managed by two actions: ‘Wait’ and ‘Cut’. An action is decided each year with first the objective to maintain an old forest for wildlife and second to make money selling cut wood. Each year there is a probability p that a fire burns the forest.” The parameters are:

* P: probability that a fire burns the forest
* R1: Reward when forest is in its oldest state and “wait”
* R2: Reward when forest is in its oldest state and “cut”
* Discount factor: how we value further vs current
* For all the state except 0 and ending state:
  + Reward is 1 if cut, o if wait

## Determines of convergence

For value iteration, the convergence is determined by error and max iteration. The error threshold is controlled by epsilon, with a decay schedule and a min epsilon.

For Policy iteration, the convergence is determined by is policy stopped to change or max iteration is reached.

## Runtime and number of iteration analysis

This section will try to explore how different things can impact the runtime performance for value iteration and policy iteration. In the Figure 1 above, we can see that when number of states are less than 1000, the runtime increase is trivial with number of states. However, it starts to pick up and grows linearly after that.

Chart, line chart

Description automatically generated

1. Forest MDP runtime with different number of states

.Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

1. Number of iterations and running time chart

Figure 2 above shows the runtime and number of iterations with different discount rate by fixing the number of states = 1600. We can see that as the discount rate increase, the runtime to get convergence growths exponentially. Similar pattern can be seen for the number of iterations took to reach convergence. However, an interesting finding is that though policy iteration seems always took a smaller number of iterations when fixing all other things constant, it actually took longer time to converge.

## Result analysis

### MDP set up: P = 0.02, r1=10000, r2=20, discount factor=0.95, n\_state = 5

In this simple set up of 5 states, with r1 to be so high with discount factor=0.95, and a small probability of the fire, it’s clear that the optimal strategy would be just wait for the whole time. The game is set up in this way in order to easily see if the results we arrive is optimal or not.

In addition, in order to determine which algorithm results better outcome, a simulation is performed with the policy each algorithm outputted. This will use 10000 simulation to derive an expected reward by following a certain policy.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Number of Iter | Time | Expected Reward | Policy | Same Policy? |
| **Policy Iteration** | 4 | 0.009040117 | 7503.23 | (0, 0, 0, 0, 0) | TRUE |
| **Value Iteration** | 6 | 0.000350952 | 7552.10 | (0, 0, 0, 0, 0) |

We can see that they both arrived at the same policy with similar number of iteration and time in this simple set up. In the table below shows the Q-learning result by using different alphas, while holding other set up constant. We can see it able to reach the optimal policy with different alpha, this is expected as the problem is trivial with small number of states.

Q-learning with different alpha

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Alpha | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| Expected Reward | 7516.26 | 7477.98 | 7535.00 | 7525.22 | 7489.38 | 7509.75 | 7513.82 | 7548.84 | 7526.85 |
| Policy | (0, 0, 0, 0, 0) | (0, 0, 0, 0, 0) | (0, 0, 0, 0, 0) | (0, 0, 0, 0, 0) | (0, 0, 0, 0, 0) | (0, 0, 0, 0, 0) | (0, 0, 0, 0, 0) | (0, 0, 0, 0, 0) | (0, 0, 0, 0, 0) |

### MDP set up: P = 0.02, r1=10000, r2=20, discount factor=0.95, n\_state = 50

In this game set up, we increased the number of states from 5 to 50, the expectation is that value iteration and policy iteration should still be able to find the optimal policy because it knows the model and rewards. The reason why the experiment did not choice number of state higher than that is because we only have rewards at the end of state, and there are always chances the forest will burn out, with number of states keeps increasing, even a tiny chance of burn out at each year will lead to “all wait” policy not optimal and thus hard to verify the results. However, 50 states might still be hard for Q-learning, as it would need to compute much more expected rewards for the actions taken in different states.

Let’s observe how value iteration and policy iteration performs in the below table, we can see they both reached the optimal policy, with value iteration runs faster.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Number of Iter | Time | Expected Reward | Policy |
| **Policy Iteration** | 49 | 0.10262394 | 302.35 | all 0s |
| **Value Iteration** | 51 | 0.002382994 | 302.27 | all 0s |

Let’s compare with Q-learning with different parameters, as number of states becomes larger, it’s hard to represent them into the table. But we can use the expected rewards as a proxy to see how the Q-learning performs.

Q-learning results with different Learning Rate table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Learning Rate | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| Expected Reward | 8.89 | 8.89 | 8.89 | 8.89 | 8.89 | 8.89 | 8.89 | 8.89 | 8.89 |

Q-Learning results with different epsilon

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| epsilon | 0.11 | 0.21 | 0.31 | 0.41 | 0.51 | 0.61 | 0.71 | 0.81 | 0.91 |
| Expected Reward | 5.68 | 8.89 | 8.89 | 8.89 | 8.89 | 8.89 | 8.89 | 8.89 | 8.89 |

We can see that no matter how we adjust our Q-learning algorithm, it’s hard to get to the optimal policy. There will always be cut decision in some early stage, which results in low expected reward. One of the reasons should be that the reward for the “wait” is always 0 until the end state, the Q-table value for “wait” is hardly change because of all those 0, so it’s easy for Q-learning to think “cut” is a good strategy to perform at early stage because it will have 1 as reward.

# Frozen lake mdp – A grid world example

## General MDP description

This MDP is generated based on FrozenLakeEnv from OpenAI Gym package with some customization into it. The goal of this game is to navigate a grid world from a starting state to goal state while avoiding the holes along the path, otherwise the agent will be stuck and not be able to reach the goal state.

The original game set up was a reward of 1 at the goal state, while 0 for every other state. However, this would cause some issue for the Q-learning as it would not be able to update meaningfully to the Q-table with almost all the state has 0 rewards. Therefore, in this report, the game set up is as below:

* There are two versions of the game, namely deterministic world and stochastic world.
  + In the **deterministic** world, the agent will be reached to the target state by execute an action with **100% probability**
  + In the **stochastic** world, the agent will reach to the target state by execute an action with **only 33.33% probability**, the rest of the probability the agent will end up with another neighbor state because the path is “slippery”
* If the agent reached the goal state, it would collect a reward of 100
* In every other state, the agent will receive a reward of -1, this is to encourage the agent to find the shortest path possible
* If the agent falls into the hole, it will receive a penalty of -10

## Result analysis

### Deterministic world with 25 states

In the game with deterministic transition probability and 25 states in total, the game set up was pretty easy. The expectation is that three algorithms should all be able to reach the optimal solution pretty easy.

First, let’s compare some results for value iteration and policy iteration:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Number of Iter | Time | Expected Reward | Policy | Same Policy? |
| **Policy Iteration** | 6 | 0.00525594 | 63.10 | (1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0) | True |
| **Value Iteration** | 9 | 0.00054908 | 63.10 | (1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0) |

Both value iteration and policy iteration reached the same optimal policy. From below simple visualization of the optimal policy, with the start at top-left corner, goal at bottom-right corner, blue color represents the hole, and grey color represent the safe place. we can confirm both of them found an optimal policy that will lead to goal state in this deterministic world.

A picture containing square

Description automatically generated A picture containing square

Description automatically generated

1. Policy iteration and Value iteration policy visualization

Let’s observe how Q-learning would perform in this deterministic world. The expectation is that it should perform similarly to above value iteration and policy iteration, because of the deterministic world and easy set up. The Q-learning should be able to visit every possible state often, and thus, be able to successfully update a Q-table that would be helpful to find an optimal policy.

We can see from graph 4 below, even though Q-learning selected a different policy for this game, it’s still a good solution. The orange rectangle highlighted the part where the difference happens, however, after carefully reviews the difference, they are valid optimal solution for this game. This result is expected with our hypothesis in this simple game set up

Shape, rectangle, square

Description automatically generated

1. Q-learning policy visualization

### Stochastic world with 25 states

After above analysis, it’s natural to think what things would change in the stochastic world. It would be expected that Q-learning would perform worse in the stochastics world, as it might no longer true that it can visit every possible state infinitely possible.

First, let’s compare the results for value iteration and policy iteration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Number of Iter | Time | Policy | Same  Policy? |
| **Policy Iteration** | 4 | 0.006428003 | (1, 2, 2, 3, 2, 1, 1, 0, 0, 2, 1, 1, 0, 0, 2, 3, 2, 1, 1, 1, 0, 2, 2, 2, 0) | True |
| **Value Iteration** | 94 | 0.005043983 | (1, 2, 2, 3, 2, 1, 1, 0, 0, 2, 1, 1, 0, 0, 2, 3, 2, 1, 1, 1, 0, 2, 2, 2, 0) |

From the above chart, we can see that policy iteration only used 4 iteration while value iteration used 94 iterations. This result is expected, because one policy can be represented by an infinite number of value functions. It can be the case where two values functions are different but can be mapped to a same policy. Therefore, the number of iterations for doing policy iteration is smaller than value iteration.

Calendar

Description automatically generated with low confidence Calendar

Description automatically generated

1. Policy iteration and Value iteration policy visualization

From the above graph, we can see they also both reached an optimal policy. Compare to the previous game, it will actually try to move away from the hole instead of moving parallel near the hole. This result looks very good for value iteration and policy iteration.

Let’s now see how it will perform in Q-learning in the below chart. We can easily see not only it did not reach the optimal solution, but it also made some clear mistakes for the policy. In the example that highlighted below, there are cases where the policy will guide the agent toward the hole, there also cases it tries to move away from the goal when close. This should be due to the fact that in this stochastics world, only 1/3 of the chance the agent will be able to reach the target state guided by the policy, this caused a lot of confusion and randomness in updating the Q-table, thus it was not able to reach an optimal policy.

Calendar

Description automatically generated with medium confidence

1. Q-learning policy visualization

### Stochastic world with 900 states

Now we have observed that Q-learning will not perform well in the stochastic world in this game, how will value iteration and policy iteration different when the number of states becomes large? In the below chart, it displays the experiment result for 900 state frozen lake problem.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Number  of Iter | Time | Same Policy? |
| **Policy Iteration** | 9 | 0.13112402 | False |
| **Value Iteration** | 144 | 0.15176082 |

We can see that both runtimes went up and they no longer reached to the same policy. In the below two charts, we can see that overall, they reached to similar policy, with difference highlighted in the red rectangle. These results are actually better than expected, as only 1/3 of the chance agent will land on the place that policy guided, it is surprising that both value iteration and policy iteration can reach to similar solution.

. Scatter chart

Description automatically generated

1. Value iteration policy visualization

Chart, scatter chart

Description automatically generated

1. Policy iteration policy visualization

# summary

# References

1. Steven A. W. Cordwell: Markov Decision Process (MDP) Toolbox for Python. <https://github.com/hiive/hiivemdptoolbox>
2. OpenAI Gym. <https://github.com/openai/gym>

# source code

Below is the link to the source code and data used for this report, more details of running the code can be found in file README.txt

<https://github.com/lijunujil/Gatech_ML_Course/tree/main/HW_randomized_Optimization>