1. MDPs Description

1.1 Forest Management

This MDP problem is introduced in this link.

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.

Here is how the problem is modelled. Let {0, 1 . . . S-1 } be the states of the forest, with S-1 being the oldest. Let **'Wait' be action 0 and 'Cut' be action 1**. After a fire, the forest is in the youngest state, that is state 0.

The transition matrix P of the problem can then be defined as follows:

```
             | p 1-p 0.......0  |
             | .  0 1-p 0....0  |
P[0,:,:] =   | .  .  0  .       |
             | .  .         .   |
             | .  .          1-p |
             | p  0  0....0 1-p |

             | 1 0.........0 |
             | . .           . |
P[1,:,:] =   | . .           . |
             | . .           . |
             | . .           . |
             | 1 0.........0 |
```

The reward matrix R is defined as follows:

```
           |  0  |
           |  .  |
R[:,0] =   |  .  |
           |  .  |
           |  0  |
           |  r1 |

           |  0  |
           |  1  |
R[:,1] =   |  .  |
           |  .  |
           |  1  |
           |  r2 |
```

In my experiments, I defined p=0.9, r1=number of states + 1 and r2=2.

1.2 BlackJack

This MDP problem is introduced in this link and link.

It is a simplified version of the popular BlackJack card game, see details in this link.

For simplicity, the game is modified in following ways:

- There is no double betting.
- The Ace only counts as 11 points.
- The stack of cards is drawn with replacement, so that every card is always equally likely to be drawn, regardless of previous drawned cards.

A state is defined by 3 factors:

- Skipped, is 1 if the player skipped in the past (and hence must skip until the end of the game) and 0 otherwise
- The card count of the player
- The card count of the dealer

With default setting in my experiment, we have 2 players and 1 dealer. The number of feasible states with the default BlackJack rules is 723.

A blackjack for the player results in a reward of 1.5, otherwise a win results in reward 1 and a loss in reward 0. Any non-final game state has reward 0.

Also, the game resets after the game is over.

2. Experiments

Here are some methodologies and rules that I followed while running experiments.

- All algorithms are tested on various number of states.
- For each number of state, I also tested algorithms several times and calculated the average of metrics.
- The **metrics** I used for algorithm comparison include convergence speed, running time and the expected total reward starting from initial state.
- **Stopping criterion** is defined as that the maximum change in the value function at each iteration is compared against epsilon. Once the change falls below this value, then the value function is considered to have converged to the optimal value function.

Here are the hyper-parameters I need to tune:

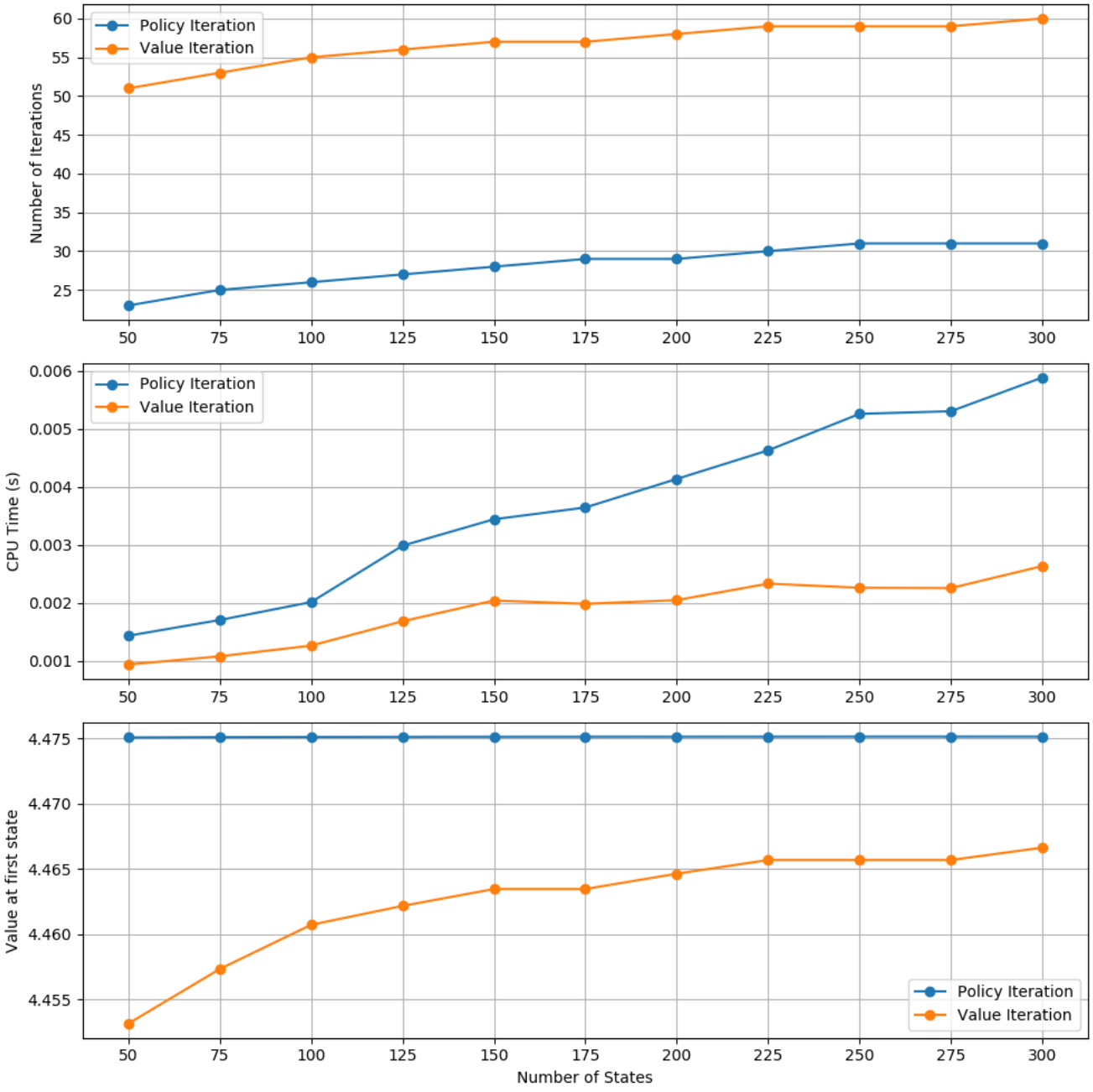
- Stopping criterion epsilon.
- Maximum number of iterations. The algorithm will be terminated once this many iterations have elapsed.
- Discount factor. The per time-step discount factor on future rewards. Valid values are greater than 0 upto and including 1. If the discount factor is 1, then convergence is cannot be assumed and a warning will be displayed.

2.1 Forest Management

2.1.1 Policy Iteration and Value Iteration

Here I chose epsilon = 0.01, maximum number of iterations=100000, Discount factor=0.9.

Below is the plots for experiments on Policy Iteration and Value Iteration.



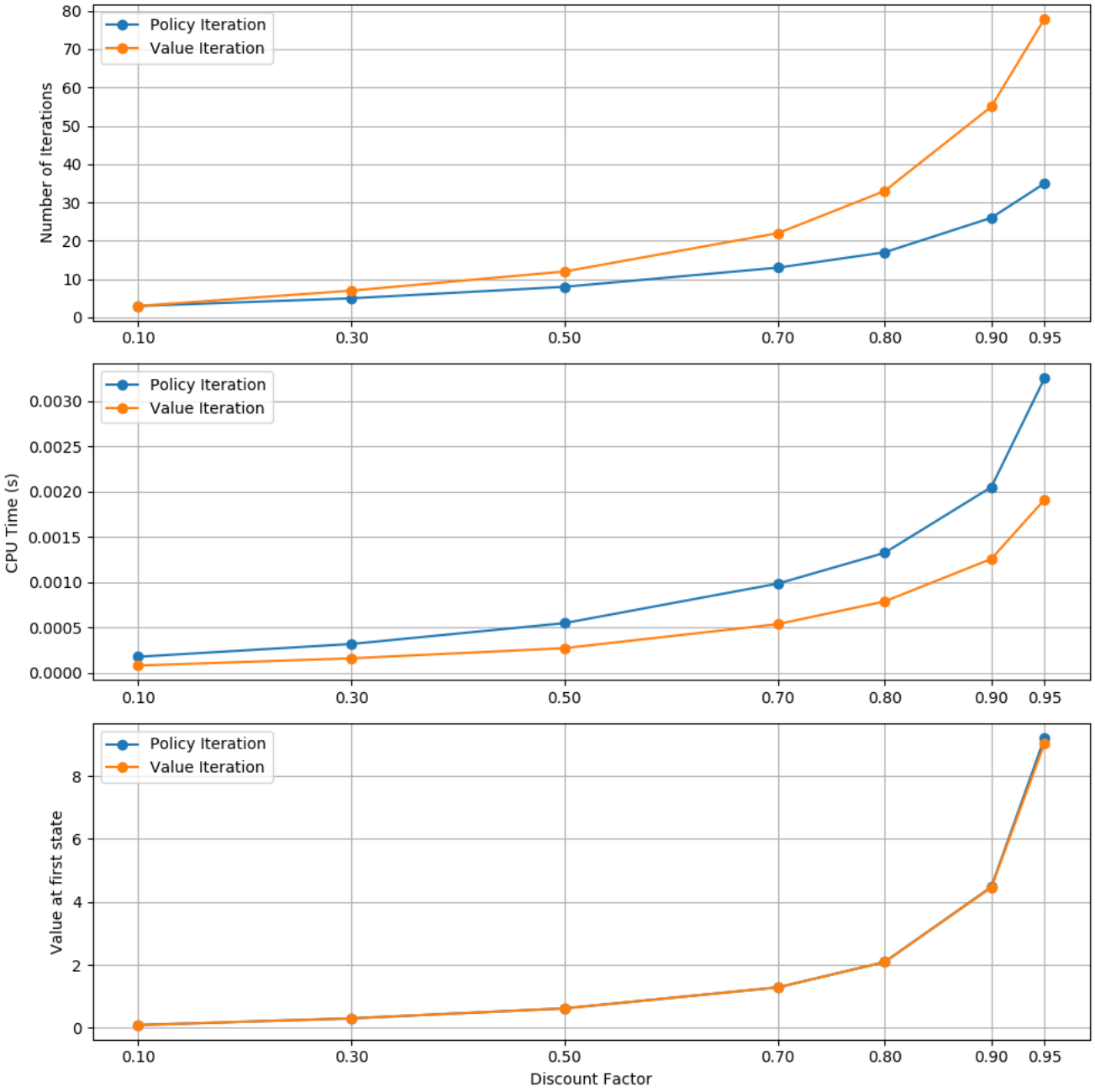
First, Policy Iteration takes fewer iterations to converge than Value Iteration, Policy Iteration converges faster than Value Iteration. Policy iteration is faster than value iteration, as a policy converges more quickly than a value function.

Second, both Policy Iteration and Value Iteration will always converge to the same policy, no matter the number of states tested here.

Third, as number of states in this forest management problem increases, the number of iteration and running time needed to fine optimal solution also increases.

Forth, as number of states in this forest management problem increases, both algorithms are able to find the optimal policy.

Below is the plots against different discount factors.



As we can see, as the discount factor increases, as we weight more on future reward, both algorithms need more iterations to converge, but both tends to find better policy overall.

This is because as the discount factor increases, we rely more on the future reward rather than current reward. A small discount factor/learning rate tend to decay too fast and give the agent less chance to learn. As we all know, a slow learner are more likely to find the better solution.

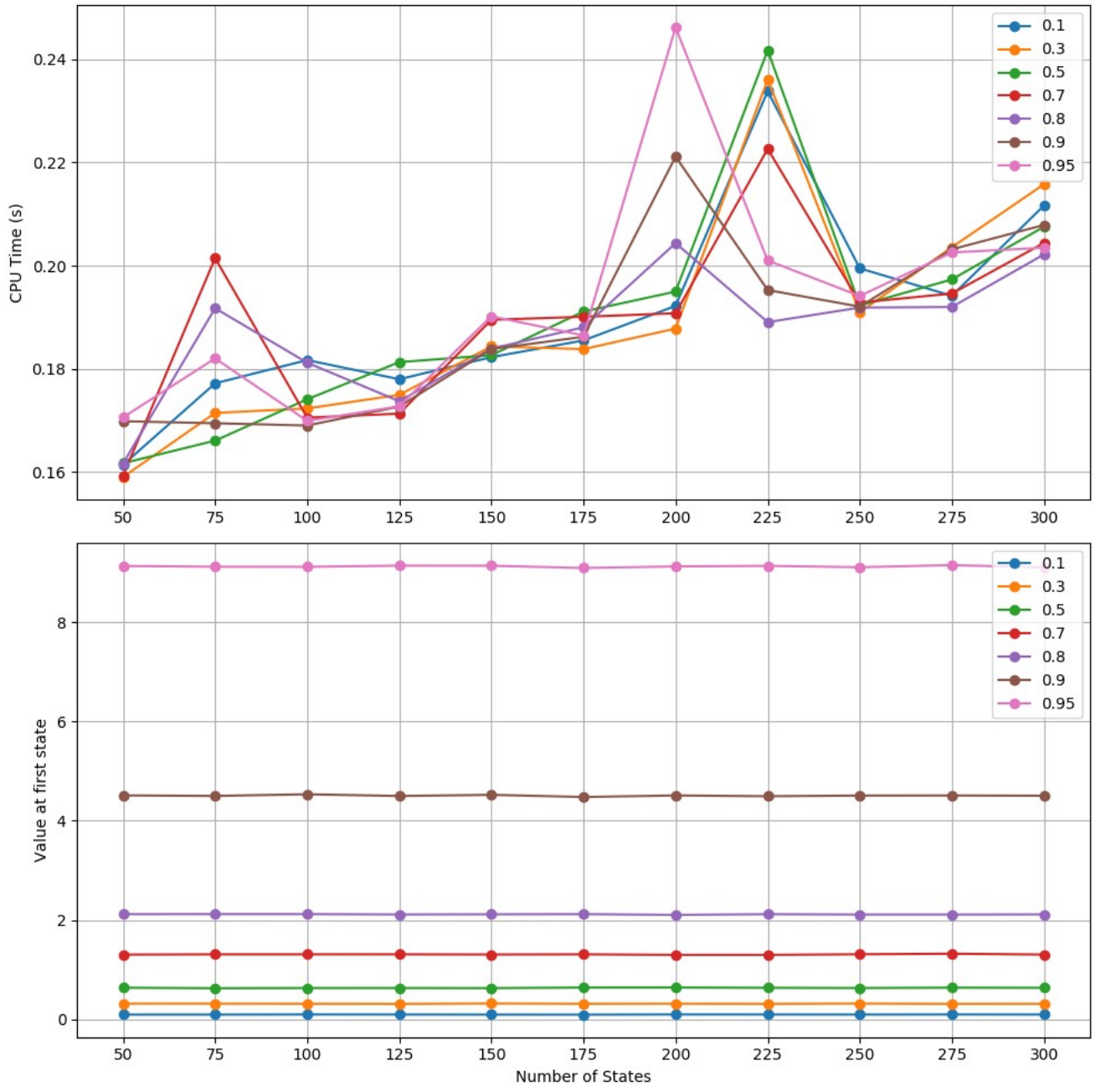
But when discount factor =1, with no discount on future reward, convergence can not be guaranteed. I didn't show the performance on discount factor =1 because I did try the experiment, but it never converge.

2.1.2 Q-Learning

Here I chose epsilon = 0.01, maximum number of iterations=100000.

In order to balance between exploration and exploitation, and to tryout different exploration strategies, I testes Q-learning on various discount factors.

Below is the plots for Q-learning performance against different discount factors.



First, as discount factor increases, as we weight more on future reward, the running time also increase because it takes more time and learning experience to converge.

Second, as discount factor increases, the expected total reward starting from initial state also increases. Hence with larger discount factor, Q-learning tends to perform better and is able to find better policy. A small discount factor/learning rate tend to decay too fast and give the agent less chance to learn.

When discount factor =1, with no discount on future reward, convergence can not be guaranteed. I didn’t show the performance on discount factor =1 because I did try the experiment, but it never converge.

Third, as number of states in this forest management problem increases, the number of iteration and running time needed to fine optimal solution also increases.

Compare with Q learning (e.g., model-free reinforcement learning), Policy Iteration and Value Iteration have specific state transition probability (e.g., model-based reinforcement learning).

Compared with Policy Iteration and Value Iteration, Q-learning tends to run slower and needs more iterations to converge. This is as expected because in Policy Iteration and Value Iteration we assume we already knew the model and rewards while in Q-learning we have to explore the world to learn the model and rewards.

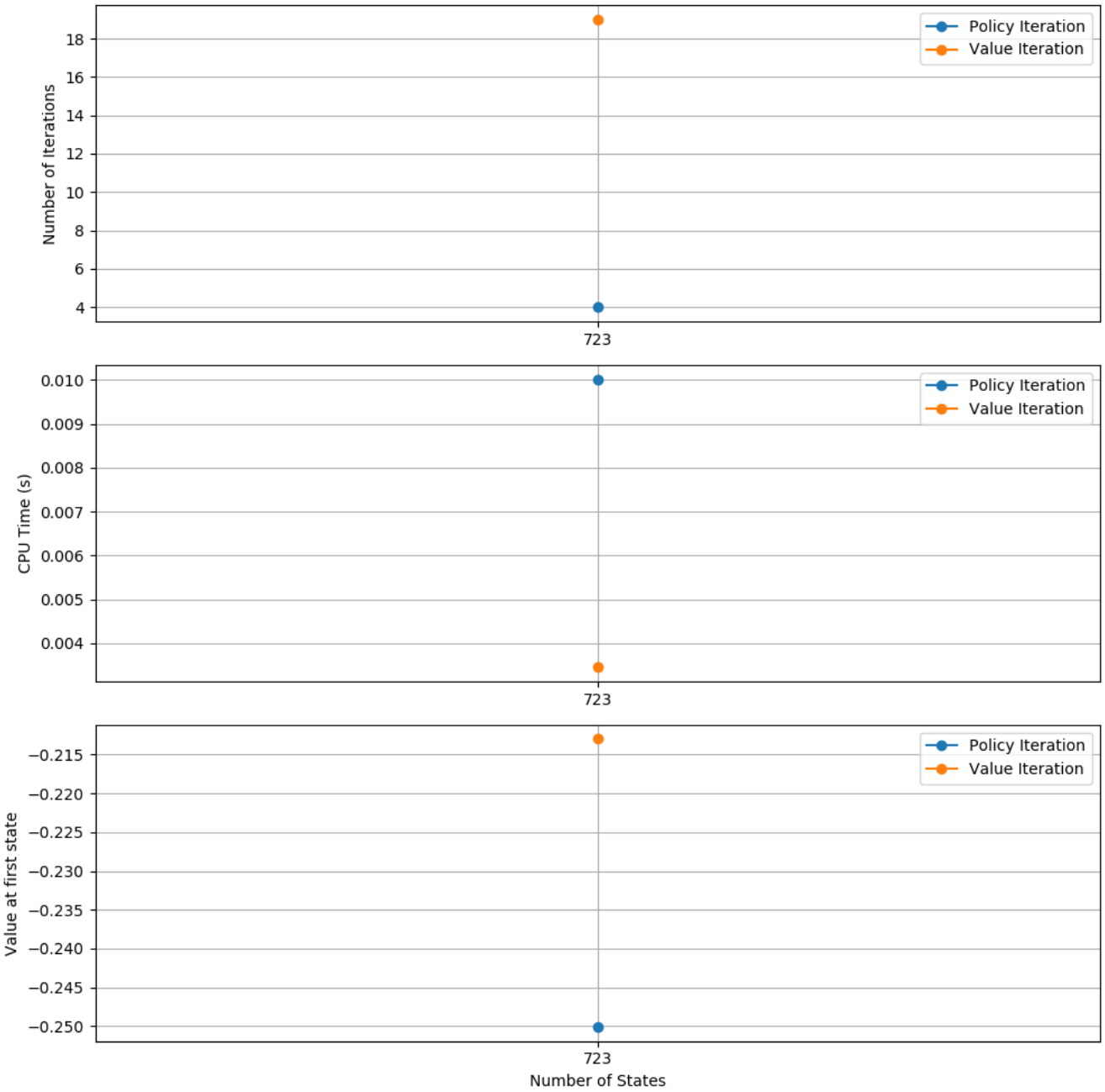
With discount factor 0.95, compared with Policy Iteration and Value Iteration, Q-learning can also find the equally best optimal policy.

2.2 BlackJack

2.2.1 Policy Iteration and Value Iteration

Here I chose epsilon = 0.01, maximum number of iterations=100000, Discount factor=0.9.

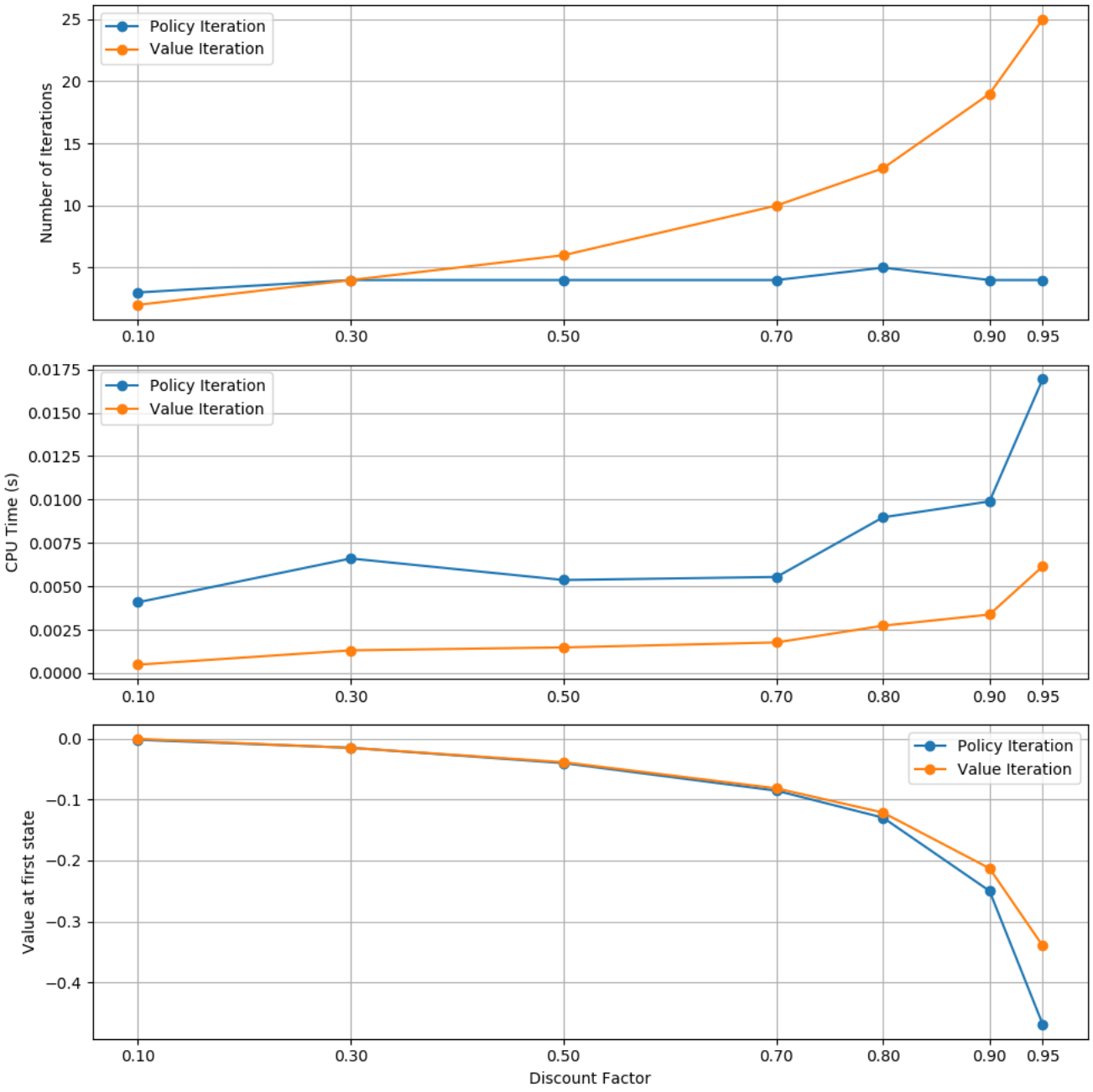
Below is the plots for experiments on Policy Iteration and Value Iteration.



First, Policy Iteration takes fewer iterations to converge than Value Iteration, Policy Iteration converges faster than Value Iteration. Policy iteration is faster than value iteration, as a policy converges more quickly than a value function.

Second, both Policy Iteration and Value Iteration will always converge to the same policy, no matter the number of states tested here.

Below is the plots against different discount factors.



As we can see, as the discount factor increases, both algorithms need more iterations to converge.

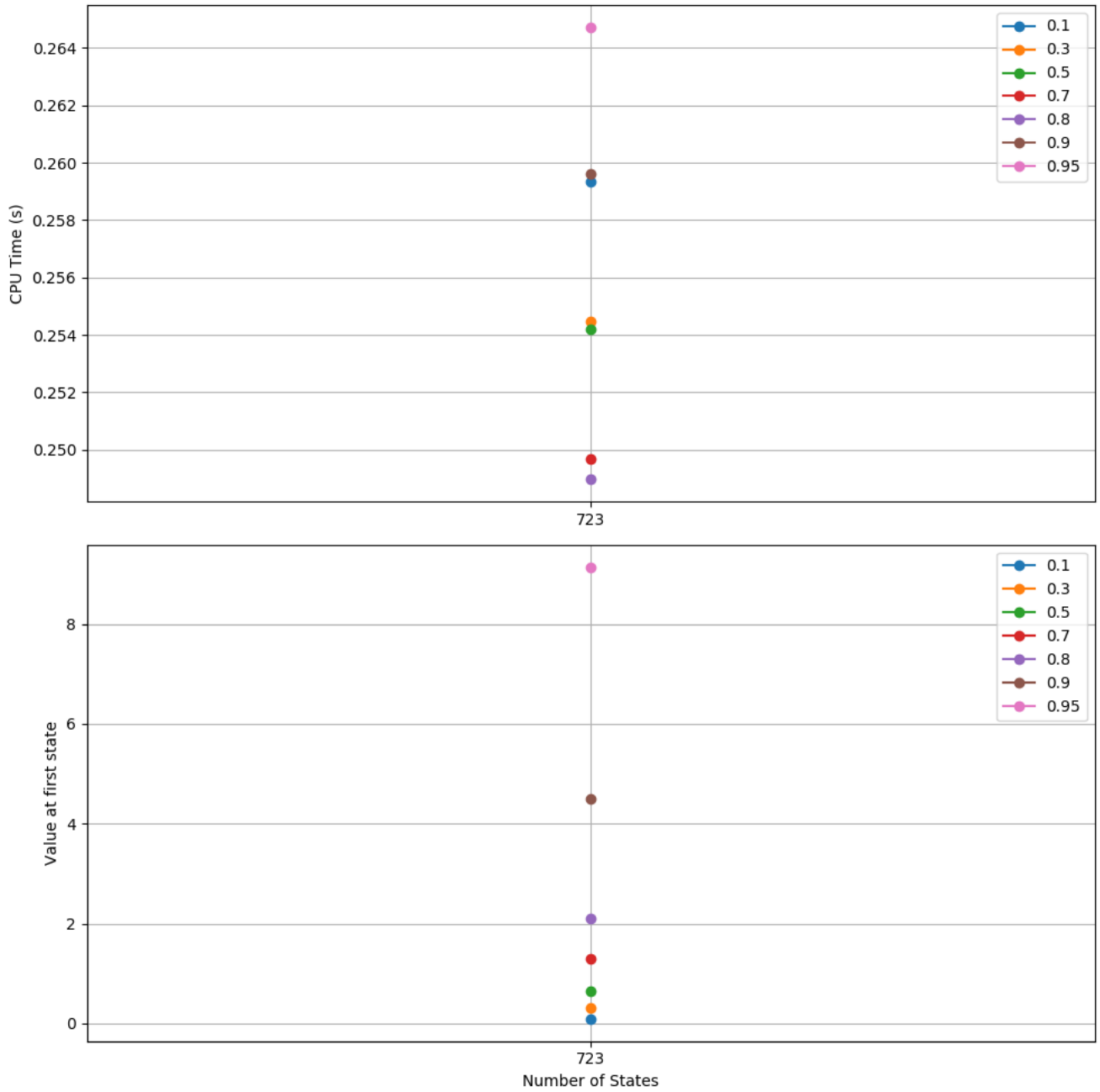
Interestingly, here as the discount factor increases, both Policy Iteration and Value Iteration tend to get worse. This is because of the nature of the problem, Policy Iteration and Value Iteration are not able to find any good solution at all.

When discount factor =1, with no discount on future reward, it never converge.

2.2.2 Q-Learning

Here I chose epsilon = 0.01, maximum number of iterations=100000.

In order to balance between exploration and exploitation, and to tryout different exploration strategies, I testes Q-learning on various discount factor.



First, as discount factor increases, the running time and number of iterations needed also increase.

Second, as discount factor increases, the expected total reward starting from initial state also increases. Hence with larger discount factor, Q-learning tends to perform better and is able to find better policy.

Same as before, a small discount factor/learning rate tend to decay too fast and give the agent less chance to learn. As we all know, a slow learner are more likely to find the better solution.

When discount factor =1, with no discount on future reward, it never converge.

Compared with Policy Iteration and Value Iteration, Q-learning tends to run slower and needs more iterations to converge. As mentioned before, this is as expected because in Q-learning we have to explore the world to learn the model and rewards.

Compared with Policy Iteration and Value Iteration, Q-learning tends to have way better expected total reward starting from initial state. This is because Policy Iteration and Value Iteration tend to stuck at local optima and unable to solve problems with large number of states and large search space.