COMP4641 Assignment #4
Liu Qinhan (qliu359@gatech.edu)

Reinforcement Learning Report

Introduction

This report is about 2 Markov decision processes (MDPs), one with a relatively small number of states, one larger. It then provides detailed analysis of 2 reinforcement learning algorithms, namely value iteration and policy iteration, trying to solve these MDPs. Their performance will be compared through a bunch of metrics.

Markov Decision Process (MDP)

Before digging into much more details, it is important to get familiar with the MDP. Markov Decision Process or MDP is an abstract mathematical modeling of a real world process and interactions. It is a convenient way of simulating the actual process so that all the interactions within the process can be finished simply by computation. In this way, it is super helpful when the result of decision making are subject to some well-defined randomness at each state.

The following jargons are used to define an MDP:

- S, a finite set of states, from which actions can be performed
- A, a finite set of actions, representing the actions that can be performed over the set of states
- T(s,a,s') = Pr(s' | s, a), a transition model representing the probability of an action a leading to a state s' from a given state s; this model encapsulates the underlying stochastic process (i.e. randomness)
- R(s), an immediate reward function for the immediate value of being in a particular state
- π(s), the policy of the decision maker, specifying which action to take in a particular state

In the context of solving an MDP, the probability of the future states is assumed to be solely dependent on the present state of the system, which is known as the Markov assumption. By solving an MDP, it means obtaining π*, the optimal policy which provides maximum total reward.

Further assumptions include the assumption of stationarity, that is, the only parameter to the policy function is the current state. There are no limitations or other constraints such as time, number of steps, etc, which in real life force changes made to the policy. Also, the utility of sequences is assumed to be logical and consistent in some way; this is done through discounting, allowing to extract finite utilities from potentially infinite sequences of states. Succinctly, they are effectively named as stationary preferences of MDPs.

Value Iterations

The concept of value iteration is rather basic: if a value of utility is assigned to every state that represents the usefulness of that state to the optimal solution in a global scope, simply performing the actions that directs to highest utility at each state yields the optimal policy.

But in a typical MDP, there is nothing about the utility of each state but only the immediate reward of all states, that is, R(s). In this case, the utility of a state can be defined to be the sum of the instant reward of that state and the expected discounted reward of that same state if taking only the optimal actions from that state onward.

These are the steps of a value iteration:

- Assign an arbitrary utility to every state in S
- For each state s in S, calculate a new utility for s based off the utility of its neighbors
- Update the utility U for each state s using the below Bellman (1957) equation:

$$U(s)_{t+1} = R(s) + \gamma \max_{a} \sum_{s'} T(s, a, s')U(s)_t$$

- Repeat the previous two steps until convergence

<u>Policy Iterations</u>

Although value iteration is rather simple and useful but it suffers from 2 major weaknesses. It could require a lot time to converge for some scenarios and it only indirectly finds the optimal policy.

Policy iterations, instead, literately iterates the policy of each state to directly generate optimal policy. These are the steps of a policy iteration:

- Generate an initial random policy, being a permutation of actions for all states in the MDP
- Loop until no action in our policy changes:
    * Compute the utility for each state in S relative to the current policy
    * Update the utilities for each state
    * Select the new optimal actions for each state, changing the current policy in the process

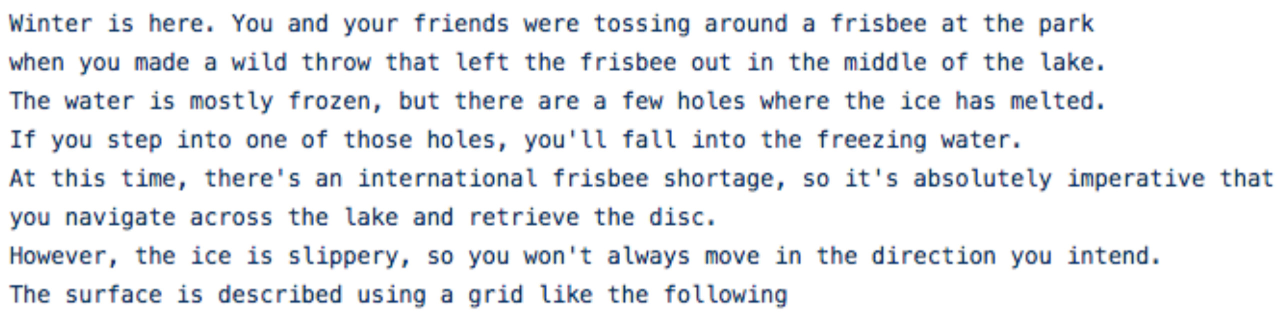
<u>The MDPs Chosen</u>

<u>OpenAI Gym</u>

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It provides simulated environments, that is, MDPs for the reinforcement learning algorithms to interact with. In this report, 2 text games are chosen to be solved by both value iteration and policy iteration.

<u>Frozen Lake 8*8</u>

The description of the game:



```
Winter is here. You and your friends were tossing around a frisbee at the park
when you made a wild throw that left the frisbee out in the middle of the lake.
The water is mostly frozen, but there are a few holes where the ice has melted.
If you step into one of those holes, you'll fall into the freezing water.
At this time, there's an international frisbee shortage, so it's absolutely imperative that
you navigate across the lake and retrieve the disc.
However, the ice is slippery, so you won't always move in the direction you intend.
The surface is described using a grid like the following
```

```
S : starting point, safe
F : frozen surface, safe
H : hole, fall to your doom
G : goal, where the frisbee is located
```

```
"8x8": [
    "SFFFFFFF",
    "FFFFFFFF",
    "FFFHFFFF",
    "FFFFFHFF",
    "FFFHFFFF",
    "FHHFFFHF",
    "FHFFHFHF",
    "FFFHFFFG"
],
```

```
The episode ends when you reach the goal or fall in a hole.
You receive a reward of 1 if you reach the goal, and zero otherwise.
```

This game is chosen because it is a typical grid world that would be exemplified as the the first problem in every reinforcement class. It has only 4 actions possible to be taken at all 64 states, so it is relatively simple to compute yet demonstrates the strength of a reinforcement learning algorithm.

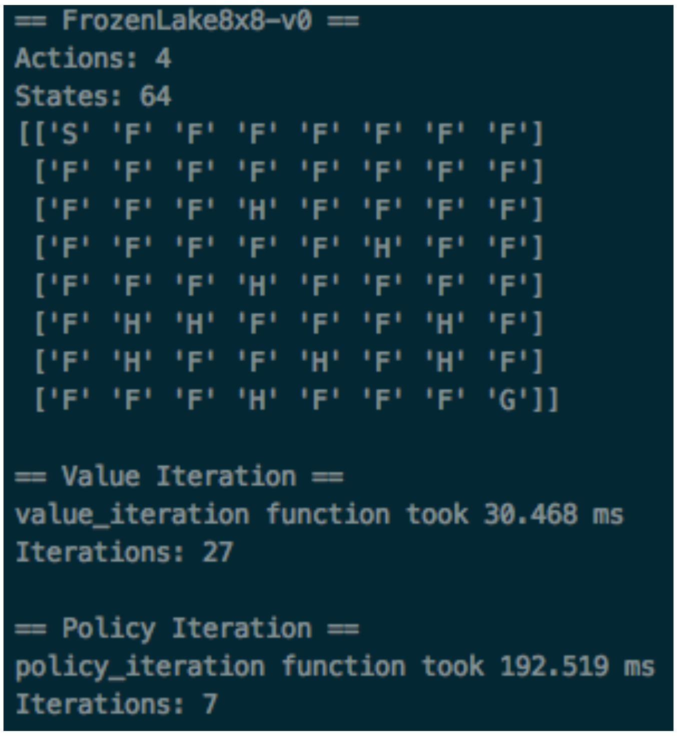
Taxi

There are 4 locations (labeled by different letters) and the task is to pick up the passenger at one location and drop him off in another. You receive +20 points for a successful dropoff, and lose 1 point for every timestep it takes. There is also a 10 point penalty for illegal pick-up and drop-off actions.

This game is more complex comparing to the previous one. It has 6 actions and in total 500 states (because of the fact that the statuses of each passenger and the car are counted into each state of the game).

```
MAP = [
    "+---------+",
    "|R: | : :G|",
    "| : : : : |",
    "| : : : : |",
    "| | : | : |",
    "|Y| : |B: |",
    "+---------+",
]
```

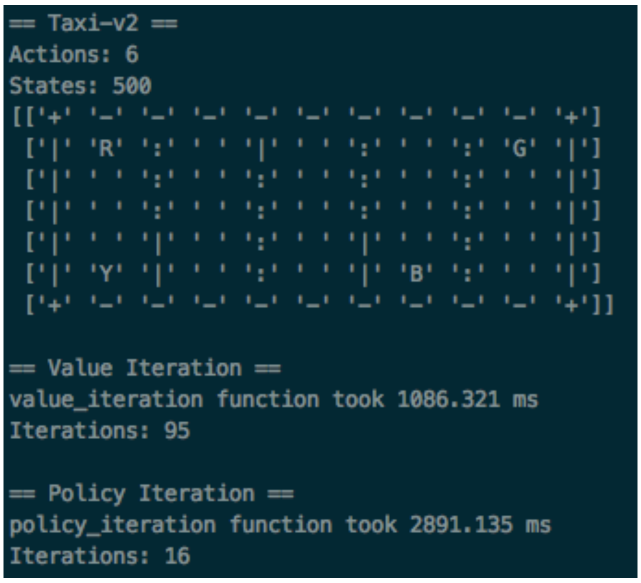
The Results and Analysis



```
== FrozenLake8x8-v0 ==
Actions: 4
States: 64
[['S' 'F' 'F' 'F' 'F' 'F' 'F' 'F']
 ['F' 'F' 'F' 'F' 'F' 'F' 'F' 'F']
 ['F' 'F' 'F' 'H' 'F' 'F' 'F' 'F']
 ['F' 'F' 'F' 'F' 'F' 'H' 'F' 'F']
 ['F' 'F' 'F' 'H' 'F' 'F' 'F' 'F']
 ['F' 'H' 'H' 'F' 'F' 'F' 'H' 'F']
 ['F' 'H' 'F' 'F' 'H' 'F' 'H' 'F']
 ['F' 'F' 'F' 'H' 'F' 'F' 'F' 'G']]

== Value Iteration ==
value_iteration function took 30.468 ms
Iterations: 27

== Policy Iteration ==
policy_iteration function took 192.519 ms
Iterations: 7
```



```
== Taxi-v2 ==
Actions: 6
States: 500
[['+' '-' '-' '-' '-' '-' '-' '-' '-' '-' '+']
 ['|' 'R' ':' ' ' '|' ' ' ':' ' ' ':' 'G' '|']
 ['|' ' ' ':' ' ' ':' ' ' ':' ' ' ':' ' ' '|']
 ['|' ' ' ':' ' ' ':' ' ' ':' ' ' ':' ' ' '|']
 ['|' ' ' '|' ' ' ':' ' ' '|' ' ' ':' ' ' '|']
 ['|' 'Y' '|' ' ' ':' ' ' '|' 'B' ':' ' ' '|']
 ['+' '-' '-' '-' '-' '-' '-' '-' '-' '-' '+']]

== Value Iteration ==
value_iteration function took 1086.321 ms
Iterations: 95

== Policy Iteration ==
policy_iteration function took 2891.135 ms
Iterations: 16
```

To find the same optimal policy, for both games, the number of iterations required for policy iteration is much less than that for value iterations. This is reasonable because although the value of each state in the later iterations of value iteration would still be changing, the optimal policy of each state has already converged. Value iteration would not be sensible to this case and would not be finished until all the values converge.

Though requiring less iterations, policy iterations consume a longer time to converge. Because the work done in each iteration for policy iteration is much more complex than value iteration. In policy iterations, the utilities of all states are computed again and again each iteration and then each state are looped through to find better actions in order to update the set of policy. Whereas in value iteration, only a loop through all the states to update values would be enough.

As the number of states increases, the difference of time needed for the 2 algorithms is decreased (from 30:192 ~= 1:6 to 1086:2891 ~= 1:3). But it does not suggest that the time needed for value iteration would finally exceed that needed for policy iteration when the number of states is large enough.