Markov Decision Process and Reinforcement Learning

Introduction

Markov Decision Process models stochastic process with discrete time assuming Markov property of the process. At each time step, the process can be in a set of possible states and has a set of possible actions and at the next time step it moves into a new state with some probability depending on its current state and action. The Markov property makes sure that the effects of an action taken in a state depend only on that state and not on the prior history, i.e. given current state and action; next state is conditionally independent of all previous states and actions. Rewards are often given after the transition. The core of MDP is to find policy (what actions to take at each of the states) that can maximize our expected total rewards (which is often discounted over infinite horizon).

In this analysis, we will present two interesting MDPs and solving them with value iteration and policy iteration. We will compare the results and also explore reinforcement learning (more specifically, Q-learning) on these two MDPs. We used ABAGAIL (https://github.com/pushkar/ABAGAIL) implementation of Maze MDP and also adopted its MDP implementation to build our gambler's problem. The three algorithms are also available in ABAGAIL and modification has been made to suit our needs.

Overview of Two MDPs for Testing

We choose two MDPs for exploring the performance and characteristics of value iterations, policy iterations and Q-learning. The first one is a gambler's decision with small number of states, and second one is maze with much larger number of states and we make it difficult by setting up trap.

1. Gambler's decision

This classic gambler's problem is quite interesting and even for small number of states is not trial at all. Basically a gambler need make bets on coin flips. If it comes up heads, then he wins as many as he puts as stake. If it is tails he will loses his stake. The game ends when he wins by reaching his goal of certain dollar amount, or loses by running out of money. For each coin flip, he need to decide the size of stake, which is limited to what he has and also he does not want to overshoot to reach his goal. So if he has $10 and want to reach $15 goal, he will not bet more than $5.

The problem can easily be formulated as an MDP. We set the goal to $5 to have small number of states but not too trivial. The state is the gambler's capital {0,1,2,3,4,5}, and

actions a are stakes {1,2…, min(s,5-s)}. The reward is zero on all transitions except when gambler reaches his goal of $5, when it is +1. And we set up in our program to let gambler stay whatever states when he makes ineligible move (e.g., bet $2 when he only has $1). We are going to discount the reward with discount factor 0.95 as default. The final factors that matters is the probability of winning(coin flip as heads) and we assume it is known as a fixed value and we are going to test how it affects the optimal policy.

2. Maze

The maze problem is simple to understand but can be difficult to solve with specific layout and large number of states being set. We are going to first try one simple generic 11x11 maze, then test one with similar size but designed so to make it difficult. We will set up one trap inside that occupies most of the maze with one narrow entrance acting also as exit, so once you get in the trap; it is not easy to get out. We will also try to use same layout but with larger size maze just to see how number of states affect three algorithms. The layouts of the generic and special maze are shown in Figure 1, in which "#" is wall, "o" is the agent and "x" is exit of maze.

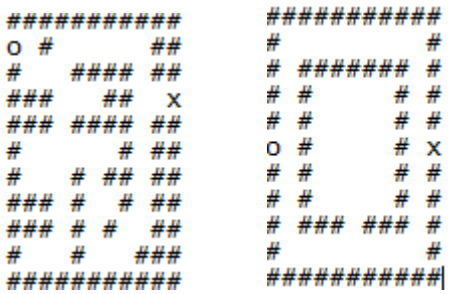

Figure 1 Layouts of maze(left is generic and right is with trap design)

Overview of Three Approaches to Solve MDPs

1. Value Iteration

The first algorithm we will look at is value iteration. The intuition behind value iteration is that we can make our decision simple by just following the actions that maximizing total "value" of each state we expect to visit. And our task is to find the true "value" of each state. It works backward to refine the estimate of value for each state; it terminates when there is hardly any improvement (usually with threshold set). The value iteration can take a long time to converge in some situations, but only requires o*(card(s)^2*card(a))* time at each iteration and it is often the case that action space is much smaller than that of the state space.

2. Policy Iteration

For policy iteration, we actually don't care what the value of each state is; it just a tool to help us find the optimal policy and it should not be necessary to compute value for all states. Since there are only finite numbers of policies for MDP with finite states and actions, search over policies might be more direct and efficient. The idea of policy iteration is to starting with a random policy and iteratively improving it until we reach optimal policy. Policy iteration usually converges fast with small state space, but may be very slow for large state space because each iteration takes o(card(s)^3) time by solving possibly large linear system.

3. Reinforcement Learning

Value Iteration and Policy Iteration works great for finding optimal policy given we know the transition function and the reward for all states in the environment. If we do not have such domain knowledge, what we can do is through a form of reinforcement learning known as Q-learning. Q-learning is a form of model-free learning that agent does not need to have any model of the environment; it only needs to know what states exist and what actions are possible in each state. Q-learning often brings up a tradeoff of exploration and exploitation. Learning through mistake is costly; On the other hand it is the case that some time spent making errors early on can lead to improved overall performance.

Solving the Gambler's Problem

First we set the probability of winning for the gambler to 0.4(less favorable for gambler), value Iteration and policy Iteration both converge very fast. The value iteration finishes in 32 milliseconds and in 132 iterations, while policy iteration finds the optimal in just 4 iterations and in 1 millisecond. We can see the valuation for states (except 0 since it will always be zero) for value iteration shown in Figure 2.

When we change the discount factor from 0.9 to 0.1, it only takes the value iteration 6 iterations to find optimal policy. This suggests value iteration performs much better if it focuses on the immediate rewards for this problem. Policy iteration for problem with small number of states will be a better choice since the computation cost during iterations can almost be ignored. And it is more efficient as it evaluates the policy directly and stops at optimal.

Value iteration and policy iteration always find the exact same optimal policy in our test, which is {0,1,2,2,1,0}. We can ignore the optimal action for state 0 and 5 since they are terminating state. The optimal policy says to bet only $1 when having $1 or $4, but bet aggressively at $2 when having $2 or $3 since the gambler only get rewarded by reaching the goal of $5.

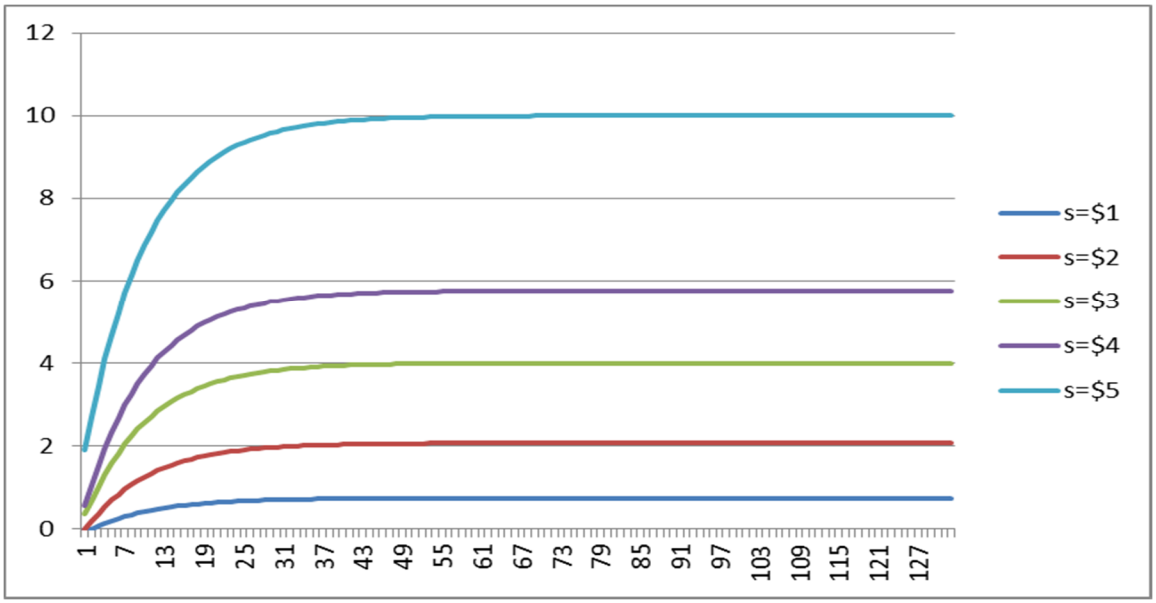

Figure 2 Valuation of states for Gambler's problem

Next, we examine what if the nature favors the gambler. With winning probability 0.6 and discount factor 0.99, the value iteration and policy iteration find the same optimal policy as {0,1,1,1,1,0}. It is a conservative strategy that always bet the amount of 1. However, the optimal strategy will change back to {0,1,2,2,1,0} when we change the discount factor to 0.1. For discount factor 0.9, the optimal policy both find is {0,1,1,2,1,0}. It is mixed strategy that basically let gambler be conservative when he is at risk running out of money(low on the capital) and be aggressive otherwise. This is very intuitive if we view discount factor here as indication of gambler's willingness to stay in the game.

It has be shown in theory that the optimal policy for undiscounted version of this problem is

- Always bet $1 if winning probability is greater than 0.5
- Bet *min*(s,N-s) in state s if winning probability is less than 0.5 (N is the goal)
- Any policy is optimal when winning probability is 0.5

Our findings with discount get similar result as shown above and the findings are also very in line with our common sense as we can see the case for discount factor 0.9 and winning probability 0.6.

Q-learning does not converge to the same optimal policy as mentioned above within 50000 iterations, and the policy Q-learning finds is not consistent across different runs. The policy often contains part of optimal policy found above, such as {0,0,0,2,0,0} and {0,0,0,2,1,0}. Initially we tried greedy strategy for exploration and the only policy Q-learning finds is {0,0,0,0,0,0}. This is understandable as every state except the $5 state get reward of zero. As q-learning updates the new Q-value, it samples the new sates and is most likely return zero. The result is basically to choose the action that does not

change the state, thus we are seeing all 0 for strategy. Epsilon greedy strategy is used to mitigate and the result as shown above is not promising either. Further tuning of the parameters of Q-learning might be necessary to improve the performance of Q-learning on this particular problem.

Solving the Maze Problem

First we tested the three algorithms on the generic 11 X 11 maze and Figure 3 shows the optimal policy found by each algorithms, the direction(actions) is denoted as one of the “<”, “>”,”^” “V”.

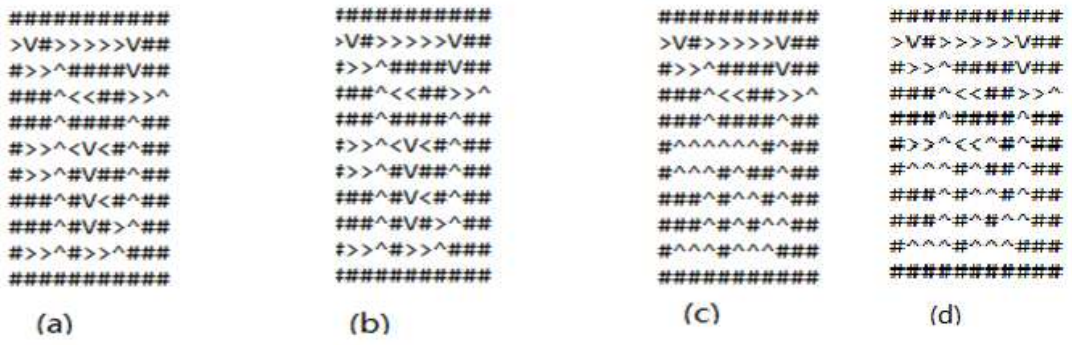

Figure 3 Policy found by three algorithms for maze problem

(a) Value Iteration (b) Policy Iteration

(c) Q-Learning(50000 iteration) (d) Q-Learning(5 million iterations)

As we can see, value iteration and policy iteration does not have any problem finding the optimal policy and they are identical. For Q-learning, the policy is partly different from the optimal policy, but seems generally be a good policy. We increased the iterations from 50,000 to 5,000,000 for Q-learning so it can visit certain state action pair for enough number of times, the result only gets slightly better as shown in (d).

Table 1 Iterations and times to find the optimal policy for 10x10 generic maze

	Value Iteration	Policy Iteration	Q-learning
# iterations	157	15	50,000
Time(millisecond)	187	610	142

As shown in Table 1, value iteration need more iterations to find the optimal policy than policy iteration as it needs to updates value for each state, while it still takes less time than policy iteration since policy iteration incurs high computation cost inside each iteration (as we know it is in o(|s|^3)). Value iteration also converges fast as shown in figure 4. If we repeat what we did for discount factor in our previous analysis for gambler's problem, we can find similar result for value iteration algorithm. The number of iterations needed to find optimal policy is reduced dramatically from 157 to 9, and it is less than what requires of policy iteration (from 15 to 13). Similar change for Q-learning only makes it find a policy that is always to go "up".

For the specially designed maze problem, we want to see how the three algorithms handle the trap. The trap has one entry/exit point, which makes evaluate the value of state near the entrance/exit difficult since the optimal policy will try to avoid getting into the trap and there is only one point to get out once getting in the trap.

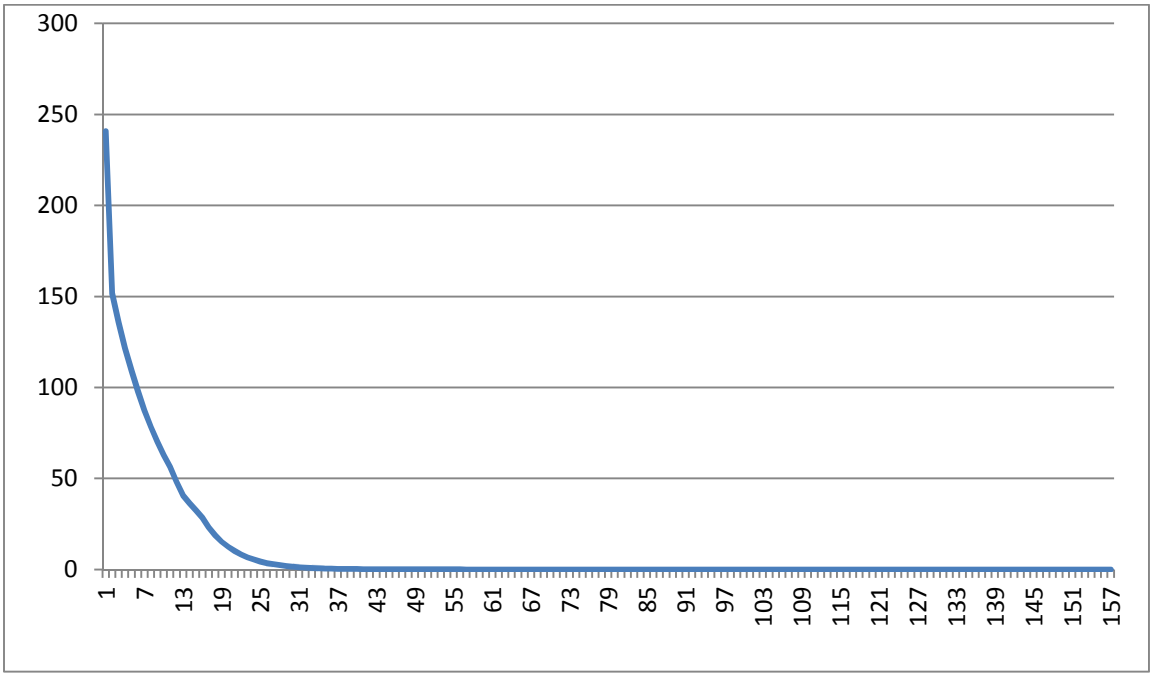

Figure 4 Valuation Difference by iterations for maze with value iteration

Figure 5 shows the optimal policy found by the three algorithms. It is not a surprise that both value iteration and policy iteration find the same optimal policy as the number of sates for this problem is not very large. Q-learning however does not even give a good policy as it only looks to go one direction for all states.

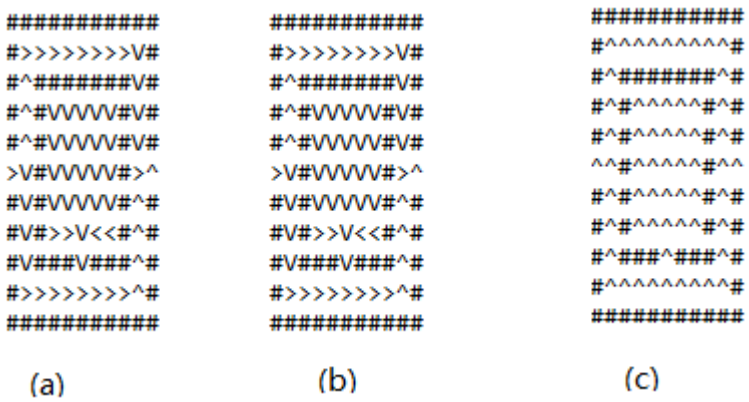

Figure 5 Policy found by three algorithms for maze problem(trap layout)

(a) Value Iteration (b) Policy Iteration (c) Q-learning (50000 iterations)

As we increase the number of states and set similar layout for 20x20 maze, we expect the policy iteration take much longer as its computation cost is up quickly in each of the iterations. Table 2 is consistent with our expectation. Policy did it with much longer time. And Q-learning could not find even a good policy while value iteration and policy iteration find same optimal policy which is similar to the one shown on Figure 3 and 5. We further increased the maze size to 100x100 and we only get result from value iteration within reasonable time 995078 milliseconds with 166 iterations, Policy iteration just takes too long(over 7 hours) and we have to kill the process. The policy found by value iteration is pretty good and very similar to what we saw figure 5, except in upper left corner of the trap box, the policy only loos to go up, which confirms that the trap is indeed posting some challenge to value iteration. There are two obstacles, one is high computation cost associated with the larger number of states and the second is trap that value iteration (or any learning algorithm) also faces.

Table 2 Iterations and times of to find optimal policy for 20x20 maze

	Value Iteration	Policy Iteration	Q-learning
# iterations	157	32	50,000
Time(millisecond)	1547	12999	438

Conclusion

From the discussion above, we can see that value iteration and policy iteration are guaranteed to find optimal policy, but they are different in performance depending on the nature of the problem, especially regarding to computation cost. For value iteration, it is not obvious when to stop and introducing epsilon adds more requirements of domain knowledge to set epsilon value. Though in our example and in practice policy is often

optimal long before the value function has converged. Policy iteration is more desirable approach when the number of states is not large or the policy space is fairly searchable since it searches the optimal policy directly and it is guaranteed to improve the policy at each iteration. For problem with large number of states like the 100x100 maze, the computation cost is too high and value iteration quickly become the favorable approach. Q-learning on the other hand does not perform well in both our MDPs most likely due to the nature of the problem, the reward is only possible when the agent finally reach the goal and agent get nothing along the path. This further delayed nature of the problems requires Q-learning to visit state with reward enough number of times to discover the optimal policy, unfortunately, the zero reward states will become dominant and it becomes a challenge for Q-learning to even start with good value estimate., although it could finally converges given enough iterations and we do see some improvement if we increase number of iterations.