CS7641 Assignment#4

Markov Decision Processes

Yancheng Liu (yliu723@gatech.edu)

I. Two MDPs Used in This Study:

Markov Decision Processes, or MDP, provide models for decision making in situations where an action taken in a state depends only on that state and not on the prior history.
A MDP model contains:

1. A set of states S in the domain
2. Possible action a given in certain state s
3. A real valued reward r associated with each state s
4. A transition function T describing of each action's effects in each state
5. (optional) A discount factor r applied to rewards in the future states, thus a reward n steps away is discounted by r^n, while 0<r<1. Small r represents more emphasis on current rewards.

Four Room

The first MDP I chose to work on is a simple four-room navigation problem. The layout is shown in Figure 1. The gridworld has dimension of 11x11. Each room is connected via a doorway to the adjacent room. Each possible location of the agent corresponds to a state in this environment. The reward for each state is -1. The agent always starts in the bottom left corner, shown as a solid black circle. It can move one step at a time. The termination condition is set at the upper right corner, as shown by a blue square. The goal is to terminate the process via the least-cost route.

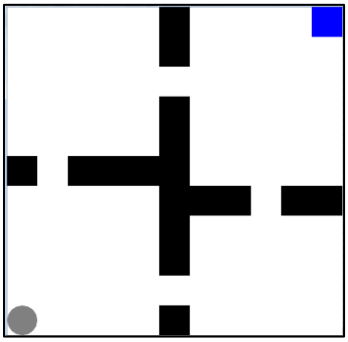

Figure 1. Layout of the Four-Room MDP.

I chose to work on four-room MDP because it is a classical MDP problem. It has simple domain that easy to set up and understand, yet it is complex enough to test and compare all the MDP algorithms and parameters I wanted. The four-room MDP is also a good model for many situations in the realistic world. For example, patients with emergency need to be moved to ICU across many patient rooms, or some very heavy or temperature sensitive cargos need to be exported through warehouses in a 2-D layout. The four-room problem will provide useful guide to plan ahead.

Random Maze

I made up the second MDP called random maze. The dimension of the gridworld is 25x25. Then I randomly put 250 blocks in this domain leaving 425 states reachable. One of the possible layouts is shown in Figure 2. Again the agent starts from the bottom left corner and terminates at the upper right corner. The reward of each state is still -1, thus the goal is to get out of world ASAP. Compared to the previous four-room MDP, I increase the dimension of the domain, so that I can compare how the size of states can affect the performance of the planning, and the learning algorithms. Also since I can create similar but independent domain layouts by change the random seeds, it is convenient to test the stability of certain algorithm. Additionally, random maze can be good model for realistic problems like traffic planning and river rerouting.

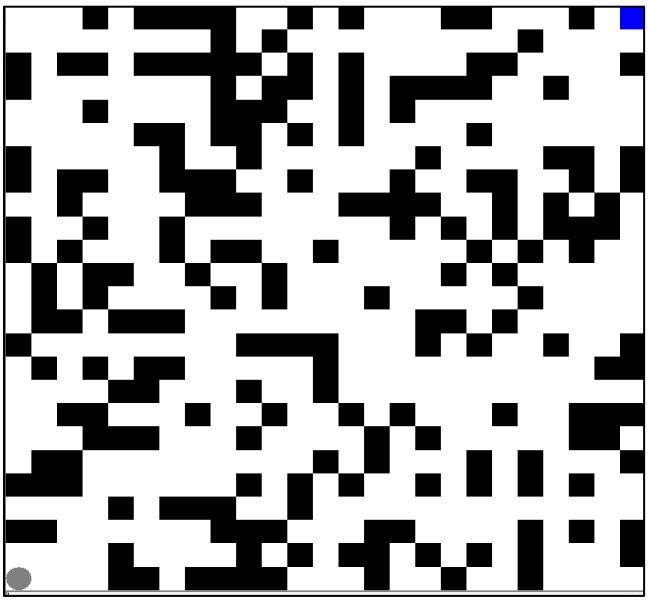

Figure 2. One possible layout of the Random Maze MDP.

Non-deterministic action

In both FourRoom and Maze MDPs, the agent is allowed to go one step at a time in one of the four directions: north, east, south, and west. I set the action of the agent to be nondeterministic: with 0.8 probability the agent will go as intended, with 0.2/3 probabilities it will go the other three directions, mimicking that in the real world, the action of an agent can fail. However, for the simplicity of the problem I assume that the location sensor of the agent is always accurate and the transition probabilities will not change when an agent is solving the MDPs.

II. Solve MDPs using Value Iteration and Policy Iteration

Solving an MDP means finding the optimal policy that maximizes the total reward. Total reward is defined as the sum of the discounted expected utility of every state visited by the agent. In this study, since the reward for each state is set uniformed as -1, the optimal policy will be the one minimizes the steps the agent takes to reach the terminal state. Here I use two algorithms, value iteration and policy iteration for solving both MDPs.

For these MDPs, only reward of each individual state is known. However, if the state's true value or utility (taking future rewards into consideration) can be calculated, then the decision making will be quite simple. The agent should just follow actions that give the maximal expected utility. So **the essential idea for value iteration** is to estimate the utility of each state through iterations. Basically, each state will be assigned an initial utility value, then this value will be updated based on current rewards, as well as the utility of its neighbors, until convergence. Value iteration is a good choice that guarantees convergence. However, it can costs unnecessary steps to update utility values, even if the underlying policy is not changing. So why not changing the policy directly? The improvement leads to another algorithm called policy iteration. **Policy iteration** starts with a randomly selected policy, and it calculates the utility based on current policy and then choose a better policy based on updated utility. When the policy is not changing any more, the optimal policy is found.

Based on how they are designed, it is expected that compared with value iteration, **policy iteration will need fewer numbers of iterations** to find the optimal policy. And it is confirmed

by the results running value and policy iteration on both four-room and maze MDPs, shown in Table 1. With the default maxDelta 0.001, the four-room problem takes value iteration 38 iterations to converge and only 9 for policy iteration. Similarly, for the maze MDP, values iteration needs to iterate for 74 times while only 10 for policy iteration. When I **tweaked the value of maxDelta**, I noticed that smaller maxDelta values corresponds to higher number of iterations, longer running time before converge and closer to optimal policy (fewer steps from start to stop). Moreover, when maxDelta = 0, neither of the algorithms converged (annotated by * in Table 2 and 3), the program is terminated by other criteria (reaching maxIterationTimes).

Table 1. Comparison of using VI and PI to solve both MDPs.

Table1		VI			PI	
	Time	Iteration #	Step #	Time	Iteration #	Step #
FourRoom	995	34	22	1916	9	28
Maze	3337	70	75	11262	10	66

Table 2. Changing MaxDelta when using VI and PI to solve the FourRoom MDP.

Table 2 FourRoom		VI			PI	
Threshold	Time	Iteration #	OP Step #	Time	Iteration #	OP Step #
1	infinite	11	infinite	1446	9	32
0.5	1008	25	26	1601	8	31
0.1	1277	28	33	1729	8	28
0.01	935	31	25	1884	9	29
0.001	995	34	22	1916	9	28
0.0001	1249	38	22	2143	9	29
0	1359	100*	21	3971	9*	25

Table 3. Changing MaxDelta when using VI and PI to solve the Maze MDP.

Table 3 Maze		VI			PI	
Threshold	Time	Iteration #	OP Step #	Time	Iteration #	OP Step #
1	infinite	12	infinite	5544	16	70
0.5	3254	63	68	6102	13	70
0.1	3201	64	74	9024	10	60
0.01	3287	67	69	11561	10	72
0.001	3337	70	75	11262	10	66
0.0001	3316	74	65	11743	10	63
0	3509	100*	61	14415	10*	59

In terms of time requirement, **value iteration is significant faster than policy iteration** in my examples. I reason it is because the evaluation step of policy iteration is usually expensive involving solving a large system of linear equations. Both MDPs shown here have relative small state numbers. For problems with really large state numbers, policy iteration will be potentially faster as evaluation step can be approximated with computing utilities for a fixed policy rather than evaluating each state as in value iteration. Moreover, **value iteration and policy iteration did converge to the same answer** for both MDPs, since both iterations provide the same utility, thus the same optimal policy (Figure 3 and 4). The **number of states** does affect the performance of value and policy iteration. The MDPs I used have the dimension of 11x11 and 25x25. With 5-fold of difference in size of states, I noticed that higher number of states correlated with longer running time and more iterations till converge, in both algorithms. This result is reasonable due to the fact there are more possibilities to explore given more states. I tried to set up domains with even higher dimensions, but the following Q-learning algorithm ended up running for too long.

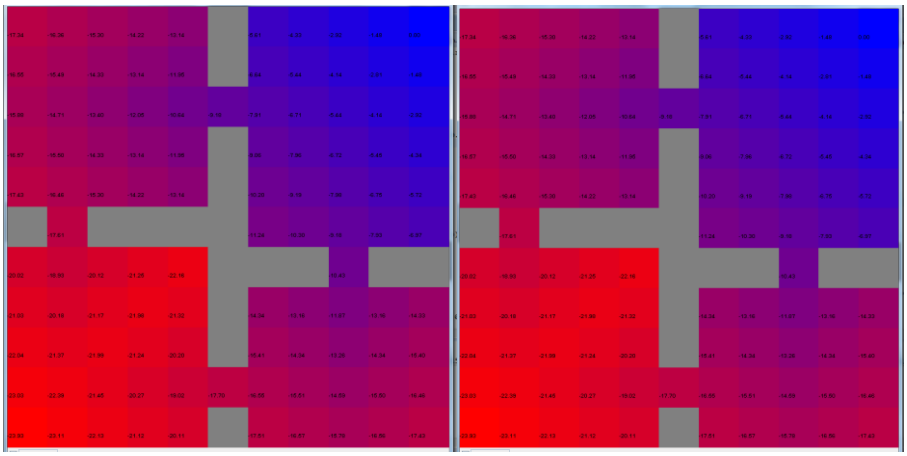

Figure 3. Utility of each state in Four-Room MDP computed by VI (left) and PI(right).

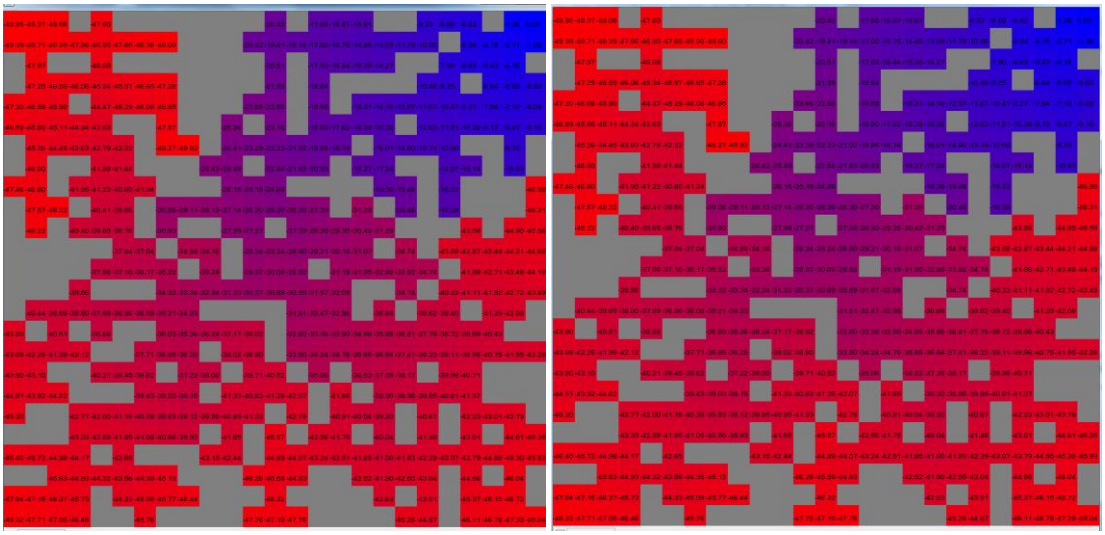

Figure 4. Utility of each state in Random Maze MDP computed by VI (left) and PI(right).

III. Solve MDPs using Q-Learning

As shown above, both value iteration and policy iteration works well for solving the MDPs. However, these two algorithms require a model or a priori knowledge of the domain. If the information for the transition function and the reward for each state is lacking, I will not be able to apply these algorithms. In these scenarios, which happens often in practice, reinforcement learning algorithms should be used. In this section, I use Q-Learning to solve the FourRoom and Maze MDPs.

Q-learning is a model-free learning algorithm. So information about certain state is acquired while the agent visits that state. Initially, each state will be assigned an estimated Q-value, somewhat similar with the definition of utility in value iteration. When the agent visits certain state, the Q-value will be updated with reward received from that state, plus the Q-value of the best possible successor state. As the agent visits more and more states, the Q-values of visited states will keep being updated. However, which action should the agent choose to test? Should the agent stick onto the route which seems a bit more promising or should it explore more possibilities around? There is always a dilemma between exploration and exploitation. How much time should be spent on learning to improve overall performance without sacrificing execution? For instance, in case of Boltzmann Q-Learning, a parameter named k (also referred to

as temperature) is introduced to control the probability to select non-optimal actions, reminiscent of simulated annealing algorithm in random optimization. The value of k is large at the beginning, allowing uniformly exploration and decreases over time.

Compared with value iteration or policy iteration**, Q-learning takes much (~100 fold) longer and final policy provided is not as good** indicated by large # of step size from start to end (Table 4). These disadvantages became even more significant when the size of states gets bigger. However, the policy found via Q-learning is completely independent of any prior domain knowledge. Thus, in cases that the agent has no access to rewards or transition information about the states, which is quite often, Q-learning remains to be a powerful algorithm to find a relatively optimal policy.

Table 4. Comparison of planning and Q-learning algorithms in solving both MDPs

Table4					
FourRoom	Time	Step #	Maze	Time	Step #
VI	995	22	VI	3337	75
PI	1916	28	PI	11262	66
Q-learning*	129165	42	Q-learning*	1547504	nd
				* greedy, 30 episodes	

I chose to examine three strategies: **greedy, epsilon-greedy, Boltzmann**. These are three different ways to trade off exploration and exploitation. The greedy strategy is to always select the action with maximized Q-value, while epsilon-greedy means with a probability of epsilon ($0 \leq$ eplison ≤ 1) a random action will be selected. In case of tie, action will be selected randomly. Boltzmann algorithm introduces a parameter temperature k, defining a decreasing rate of randomized action over time.

From Table 5, I noticed that **the running time of Boltzmann Q-learning is much longer** than the other two greedy strategies for Four-Room MDP**.** It is because the rooms are connected with each other through a very narrow opening. **If the agent is not able to pass through the door with high confidence, it will be penalized by bouncing in the room for much longer.** Boltzmann Q-learning has a higher probability to choose a random action,

especially at high temperature, so end up spending much more time in one room. Moreover, epsilon-greedy run a little bit slower than greedy algorithm due the randomness introduced by epsilon. For the four-room MDP, the optimal policy (shown by step size or negative of total reward) obtained by Boltzmann Q-learning are comparable those from greedy Q-learning. So in this case greedy is the best strategy and Boltzmann is the worst. Next, as shown in Table 6, the running times of these three strategies are similar for the random maze MDP, because the blocks are distributed quite randomly. Unfortunately, due to technical reason I failed to record the optimal policy for Random-Maze MDP via Q-learning. Moreover, by **tweaking some parameters** for Q-learning, I noticed when epsilon value from epsilon-greedy strategy is reduced from the default 0.5 to 0.1, the running time decreased. When temperature parameter from Boltzmann Q-learning are decrease from the default 100 to 10, the performance was also improved (Table 7).

Table 5. Comparison of Qlearning using three different exploration strategies to solve FourRoom MDP.

FourRoom	Greedy		epsilon-Greedy		Boltzmann	
# of episode	time	step	time	step	time	step
1	10206	43	11313	42	31817	41
10	84532	43	79341	38	310827	43
20	113041	43	123111	42	nd	nd
30	129165	42	153796	37	669450	42
50	162573	43	214227	42	nd	nd
100	223931	39	300748	43	2521885	43

Table 6. Comparison of Qlearning using three different exploration strategies to solve Maze MDP.

Maze	Greedy	epsilon-Greedy	Boltzmann
# of episo	time	time	time
1	160064	87224	100574
30	1547504	1522210	1552126

Table 7. Changing parameters of the Epsilon-greedy and Boltzmann exploration strategies.

FourRoom	Time	Step #
Epsilon-greedy		
Epsilon 0.1	114229	40
Epsilon 0.5	153796	37
Boltzmann		
k (temp) 100	669450	42
k (temp) 10	428867	42
		30 episode