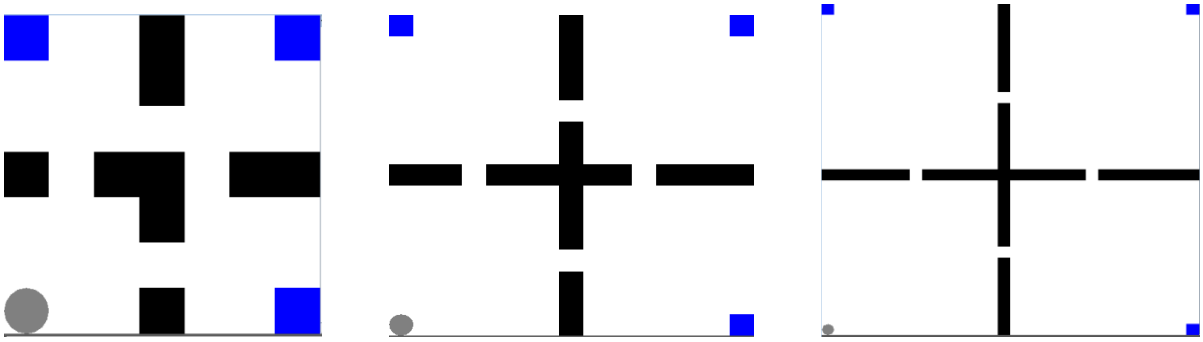
Project 4: Markov Decision Processes

CS-7641 Machine Learning
Spring 2015
Name: Fujia Wu
GT Account: fwu35

1. The MDP Problem

The MDP I choose to analyze is the "Four Room" problem. It is a gird world. The schematic is shown in Figure 1. The world has one agent, and three termination states. The walls are layered out such that the world is divided into four rooms. There is a door (or window) connecting a room with its neighbors. The starting state is set to be the low left corner, and the three termination states are set to be the other three corners.

For this problem, a uniform negative reward (-1) is assigned to all states except the termination states. The reward for the two termination states at the upper left and lower right is set to be +5, while the reward for the termination state at the upper right is set to be +10. Therefore, depending on distance of the current state to the termination states, the agent might prefer to move to one termination state over another. The transition model has 0.8 probability moving in the direction of the targeted action, and the other 0.2 spitted by the rest 3 directions.



N = 7 **N = 15** **N = 30**

Figure 1 Schematic of the N×N grid world with one agent, one termination state and four rooms, each of which has one door, for N = 7, 15, and 30

This problem is interesting to me because it has a good balance between simplicity and complexity. The resulting policy from the planners or learners will also be easily understood. Furthermore, the size N of this problem can be varied without worrying about the change of the fundamental nature of the problem (such as solvable or unsolvable). Therefore, the pure effect of the number of states can be studied.

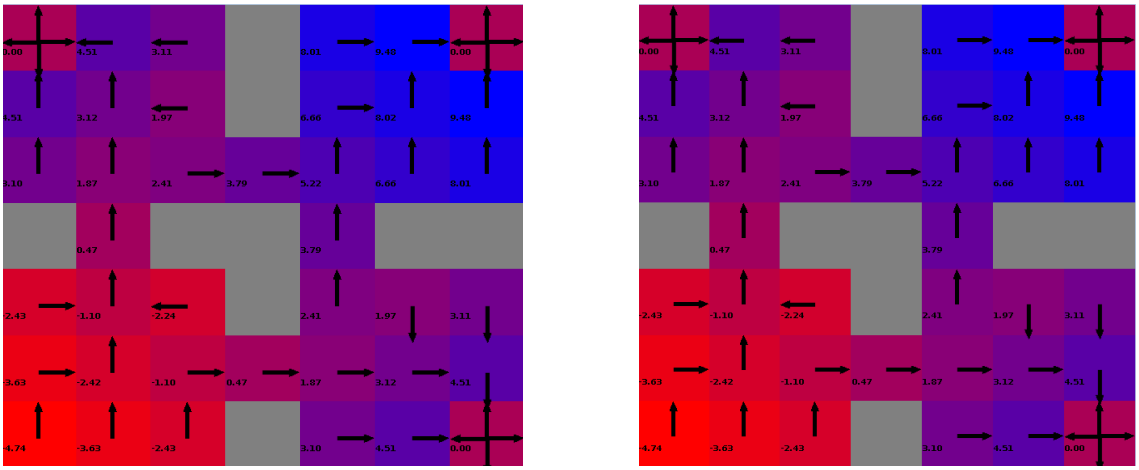
I implemented this problem using the BURLAP java code library. My code is in the file FourRoom.java. The following commands are used compile and run them:

javac –cp [path to burlap jar] FourRoom.java
java –cp [path-to-burlap-jar] FourRoom [algorithm] [N]

where [path-to-burlap-jar] is the path to BURLAP jar library, [algorithm] can be value, policy, qlearn, corresponding to value iteration, policy iteration and Q-learning. [N] specifies the size of the grid N ×N.

2 Value Iteration and Policy Iteration

BURLAP comes with the value iteration planner ValueIteration and the policy iteration planner PolicyIteration, as well as the visualization tool. I first run the value and policy iteration with N = 7, i.e, a 7 ×7 grid world, with the discount rate $\gamma = 0.99$ (i.e., put large value of future reward). The resulting utility at each state and policy are plotted in Figure 2.



Value Iteration **Policy Iteration**

Figure 2 Resulting utilities and policy iterations on a 7 ×7 four-room grid world with $\gamma = 0.99$. The utilities for all states are colored from RED (lowest value) to BLUE (highest value). The arrows indicate the policy.

From Figure 2, first it is seen that the resulting utilities (the color and the number of each grid) from value iteration and policy iteration are indeed the same. As a result, the policies based on these utilities are also the same. I have also verified that same utilities and policies are also achieved for value iteration and policy iteration for N = 15 and N = 30.

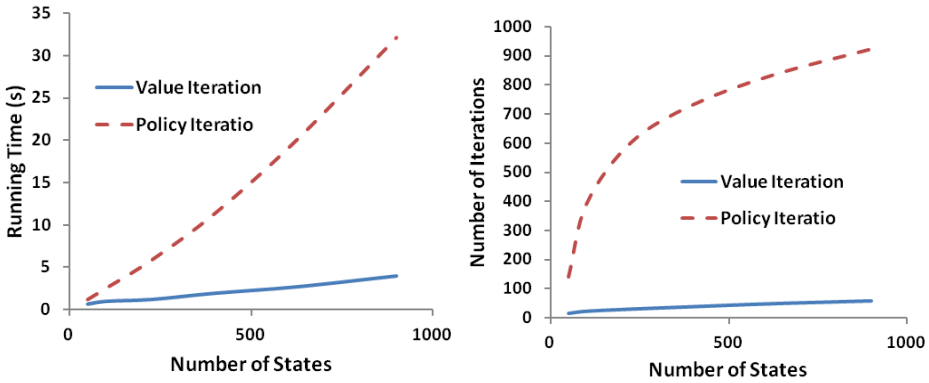


Figure 3 Running time and number of total iterations as function of number of states for value iteration and policy iteration for γ=0.99 (see the attached file Statistics.xlsx for the raw data).

Next, let us look at the running time and total number of iterations for value and policy iteration. Figure 3 plots the results as a function of the number of states. From Figure 3, it is seen that both the number of iterations and running time for policy iterations are significantly larger than value iteration. This is contrary to what was indicated in the video lecture notes. This means that policy iteration is not necessary faster that value iteration.

It is seen that in both the value and policy iteration method, the discount value (γ) is important. It indicates how the planner values the current reward or the future expected reward. I have now changed γ from 0.99 to 0.5. Figure 4 plots the total running time and total number of iterations for γ=0.5.

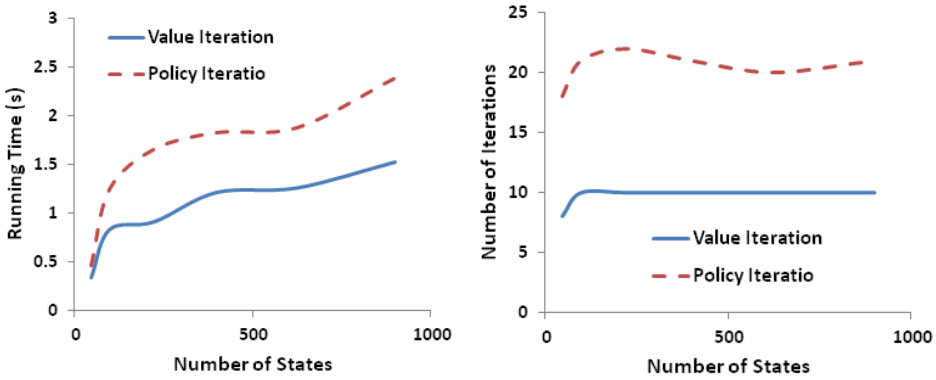
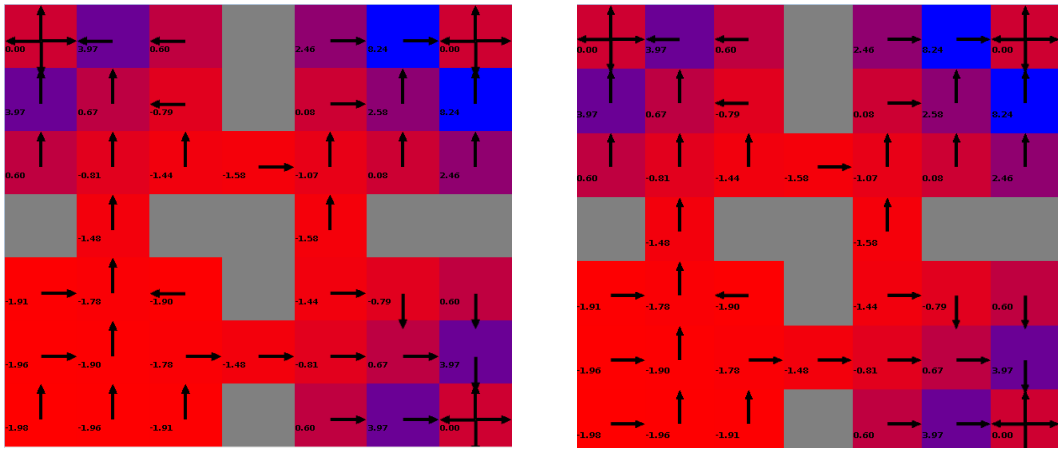


Figure 4 Running time and number of total iterations as function of number of states for value iteration and policy iteration for γ=0.5 (see the attached file Statistics.xlsx for the raw data).

With γ=0.5, it is seen from Figure 4 that although the running time and total number of iterations for policy iteration are still larger than those for value iteration, they are almost on the same order of magnitude. This means when γ is reduced, the running time and number of iterations for policy iterations significantly decreased.

Now, let us look at how the resulting utilities and policy change at γ=0.5. Figure 5 plots the utilities and policy for value iteration and polity iteration with N=7 and γ=0.5. Comparing Figure 5 with Figure 2, it is seen that both the resulting utilities and policies are indeed different for γ=0.5 and γ=0.99. This means the planning results are somehow sensitive to the value of γ. In addition, for γ=0.5, value iteration and policy iteration still converge to the same results, i.e., in Figure 5 left and right figures have the value and arrows.



Value Iteration **Policy Iteration**

Figure 5 Resulting utilities and policy iterations on a 7×7 four-room grid world with $\gamma = 0.5$. The utilities for all states are colored from RED (lowest value) to BLUE (highest value). The arrows indicate the policy.

3. Reinforcement Learning

Value iteration and policy iteration for solving MDP both assume that the agent knows, in prior, the reward function and transition function at all states. This is not true in the real world. In the real world, the agent does not necessarily know the world, except its own state and the world near it (i.e., neighbors). This brings to the need of reinforcement learning, which does not assume that the learner knows the reward and transition function of the world.

I used the classic Q Learning algorithm to learn the four-room problem mentioned above. To be consistent with the value iteration and policy iteration, I used the same discount factor $\gamma = 0.99$ for Q Learning. A learning rate 0.9 is used, and all the Q values are initialized to be 0.

Q Learning needs to follow a policy to decide the next state to learn from the current state. This is where the trade-off between exploration and exploitation needs to be considered. To implement this, I used the EpsilonGreedy policy provided by BURLAP, which takes a parameter ε. EpsilonGreedy let the Q Learning to have probability ε to take a random action, and the probability (1-ε) to take an action toward the next state with the maximum Q value. The following three lines of code connect Q Learner with the EpsilonGreedy policy.

```
Policy learningPolicy = new EpsilonGreedy(epsilon);
```

```
LearningAgent agent = new QLearning(domain, rf, tf, discount, hashingFactory,
                                    initialQ, learn_rate, learningPolicy, 1000);
((EpsilonGreedy)learningPolicy).setPlanner((OOMDPPlanner)agent);
```

I tried two values of ε to investigate the effect of exploration-versus-exploitation: ε = 0.1 and ε = 0.9. For ε = 0.1, the Q Learner will visit and learn the states with high Q value more often, therefore it is lean to exploitation. On the other hand for ε = 0.9, the Q Learner will visit more random states, therefore it is lean to exploration.

For each ε, I let the Q Learner run until it reaches the termination state. This is called a episode. I then let Q Learner run for multiple times of episodes. Let first look at the running time. Figure 6 plots the running comparison for ε = 0.1 and ε = 0.9 for different sizes of learning episodes. It is seen that the case with ε = 0.9 takes significantly longer time than the case with ε = 0.1.

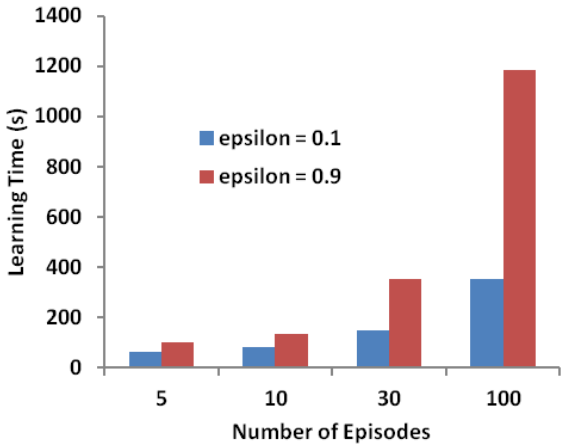


Figure 6 Compare of running time for Q Learner a 7×7 four-room grid world on with ε = 0.1 and ε = 0.9 (see the attached file Statistics.xlsx for the raw data).

The reason for the longer learning time for ε = 0.9 is that in this case, the Q Learner will visit/learn more random states, rather than high scoring states. Since all states have negative reward except the termination states, the learner with ε = 0.1 will tend to visit/learn the states closer to the termination states, and as a result will terminate sooner in average. Whereas, the learner with ε = 0.9 will visit random states more often, and thus has a less chance to terminate in each learning episode. Figure 7 plots the number of state transition steps as a function of the learning episode. It is seen that, the learner with ε = 0.9 indeed do much more transition steps in each episode than the learner with ε = 0.1.

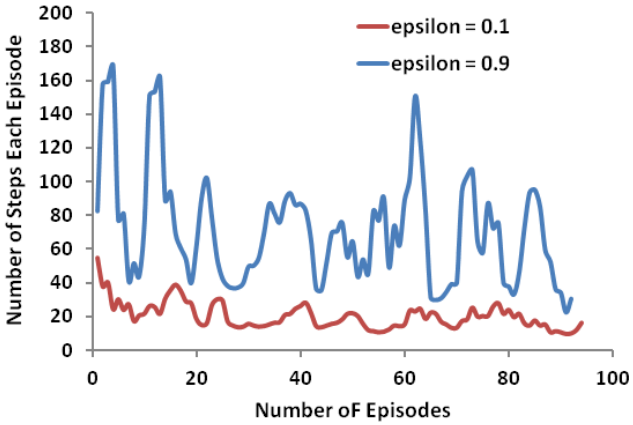
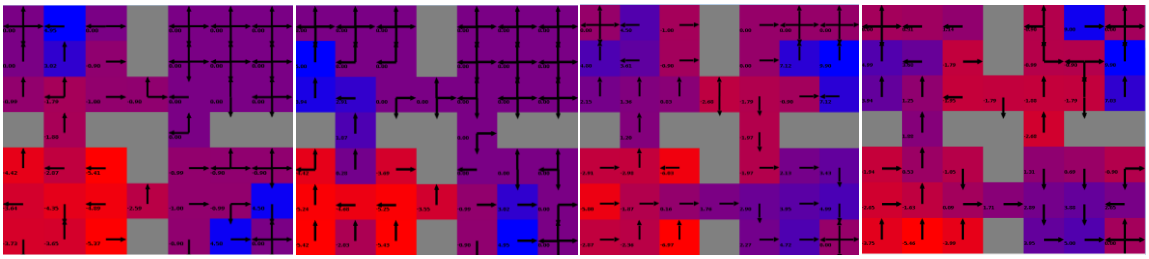
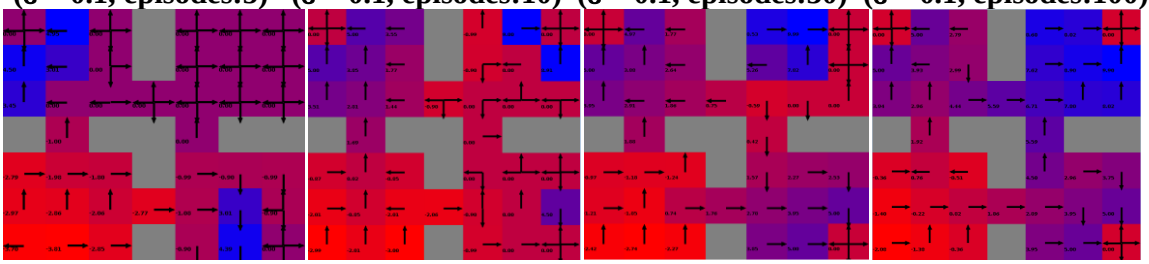


Figure 7 Compare of running time for Q Learner a 7×7 four-room grid world on with ε = 0.1 and ε = 0.9 (see the attached file Statistics.xlsx for the raw data).

Finally, let's look at the result of Q Learning. Figure 8 plots the resulting utilities and policy iterations on a 7×7 four-room for Q Learner when the learning episode is at 5, 10, 30 and 100, respectively. First, it is noted that when a state's neighbors have default Q value or equal Q value, the policy is that to choose one randomly. The BURLAP visualizer draws arrows in all directions. For states with default Q value (unlearned by the Q learner), there will be arrows in all four directions, forming a "cross".



(ε = 0.1, episodes:5) **(ε = 0.1, episodes:10)** **(ε = 0.1, episodes:30)** **(ε = 0.1, episodes:100)**



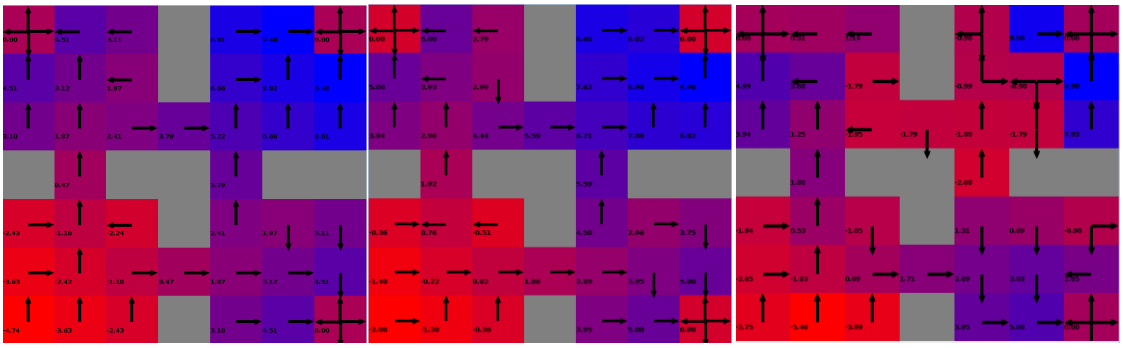
(ε = 0.9, episodes:5) **(ε = 0.9, episodes:10)** **(ε = 0.9, episodes:30)** **(ε = 0.9, episodes:100)**

Figure 8 Resulting utilities and policy iterations on a 7×7 four-room for Q Learner with ε = 0.1 (top row) and ε = 0.9 (bottom row) when the learning episode is at 5, 10, 30 and 100.

From Figure 8 it is seen that at episode 5, almost only the states at the lower left corner have been meaningfully learned since the starting state is at the lower left corner. As the

number of episodes increase, reasonable Q values and policy are assigned to more and more states. This indicates that Q value and corresponding policy more and more correct as Q Learning proceeds, which is not surprising. In addition, it is seen that the case with ε = 0.9 seems to have better results, with low Q values (RED color) located in the lower left room and high Q values (BLUE color) located in other three rooms. This is also reasonable because with ε = 0.9 each episode of the learning takes longer time to terminate and as a result performs more learning cycles.

Figure 9 compares the resulting Q-values and policies of Q Learner with the utilities and policies from Value Iteration, in which all the transition model and state rewards are known. Indeed, it is seen that the results with ε = 0.9 is quite similar to that of value iteration, while the results with ε = 0.1 is not so.



(Value Iteration) **(ε = 0.9, episodes:100)** **(ε = 0.1, episodes:100)**

Figure 9 Comparison of learning results for Value Iteration (left), Q Learner with ε = 0.9 (middle) and Q Learner with ε = 0.9 (right)

4. Summary

In this assignment, I explored MDP and Reinforcement Learning. I created a “Four Room” grid world problem. Then I solved the same problem with different numbers of states by varying the dimension of the grid world. I performed planning with Value Iteration and Policy Iteration, as well as the reinforcement learning with Q Learning algorithm. The running time, convergence, performance results of these planning/learning methods are then evaluated and discussed.