Reinforcement Learning Exercise

Steven Chen, Heejin Jeong
{chenste, heejinj}@seas.upenn.edu

November 26, 2017

1 Reinforcement Learning

In Reinforcement Learning (RL), a learning agent seeks an optimal behavior by interacting with an environment. In this exercise, we will formulate a Markov Decision Process (MDP) for the `Maze` environment described below and compute or learn its values by applying Policy Iteration (PI), off-policy Monte Carlo control, and Q-learning. Specifically we will examine the performances of the two off-policy reinforcement learning algorithms by comparing their learned Q-values with the optimal Q-values, Q^* that you compute through PI, and by semi-greedily evaluating learned values. We provide a code for the simulation of the environment, *maze.m*, and code templates to complete this exercise in *RL_source* folder.

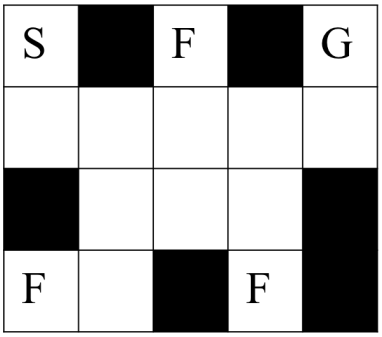

Figure 1: `Maze` domain

The diagram of the `Maze` domain is shown in Fig.1. "S" is the start location and "G" is the goal. Three flags are located in "F" and the agent's goal is to collect the flags and escape the maze through the goal. It receives a reward equivalent to the number of flags it has collected by the time it reaches the goal. A learning agent can perform one of the four cardinal actions and a state should have information about flags that have been collected as well as the current location of the agent in the maze. Therefore, it has a total of 112 states and 4 actions. The black blocks represent wall and the agent stays at the current state if it performs an action toward a wall or off the map. The agent slips with a probability 0.1 and performs the next clockwise action (for example Up becomes Right).

1. **MDP.** Using the information provided above, formulate a Markov Decision Process (MDP) of the `Maze` domain with the MDP tuple, $\mathcal{M} =< \mathcal{S}, \mathcal{A}, P, R, \gamma >$. In this assignment, the discount factor will be fixed as $\gamma = 0.95$. Additionally, complete the `get_state()` and `set_state()` functions in *maze.m*. `get_state()` maps the information of the current location of the agent and of collected flags to a state ID. It takes `obj` as an input and returns the current state ID defined in your MDP. `set_state()` maps a given state ID to the location of the agent and collected flags. It takes `obj` and `state` as an input and sets the `obj.collectedFlags` *and* `obj.agentPos` properties. You can add additional functions

and variables if necessary. *Note:* Some of the state IDs will never be reached (for example when the agent is on a flag and the flag is not collected). To simplify the solution, you can ignore these edge cases and treat it as a valid state.

2. **Policy Iteration.** Implement the Policy Iteration dynamic programming algorithm using the provided template code. Use a policy evaluation threshold of 0.0001. Record the optimal values and actions for the following state IDs: 1,30,65,95,112. Also, state the number of iterations needed for the policy to converge.

3. **Monte Carlo Control.** Implement the Monte Carlo control method with ϵ-greedy action policy to `Maze` using the provided template code, *MonteCarlo.m* and *egreedy_action.m*. Use a time-varying parameter, $\epsilon_t = n_0/(n_0 + n(s_t))$ where $n(s_t)$ is the number of visits to the current state s_t, and n_0 is a constant and set as 100 by default. For the step size of the Monte Carlo update, use $\alpha_t = (n(s_t, a_t))^{-1}$ where $n(s_t, a_t)$ is the number of visits to the current state and action pair. Initialize the Q-values with 0.0.
   Run the algorithm for 1,000 episodes. At every 10 episode during learning, evaluate the current action values by computing the Root Mean-Squared Error (RMSE) $\sqrt{(|\mathcal{S}||\mathcal{A}|)^{-1} \sum_{s \in \mathcal{S}, a \in \mathcal{A}} (Q(s,a) - Q^*(s,a))^2}$ with the optimal action values, Q^*, found in the previous section (Use *rmse.m*). Additionally, semi-greedily evaluate the current action values using the provided function *greedy_eval.m*. This function performs ϵ-greedy policy with $\epsilon = 0.1$ and returns the averaged total number of steps to escape the maze, bounded by 100 steps, and the averaged reward at the end of each termination over `numItr` iterations. Repeat the learning for 10 trials and plot the evaluation results averaged over the trials against episode number. Experiment with different values for n_0. What is the best value for n_0 according to your evaluation results?

4. **Q-learning.** Apply Q-learning with ϵ-greedy to `Maze` using the provided template code, *Qlearning.m* and *egreedy_action.m*. Likewise, use $\epsilon_t = n_0/(n_0 + n(s_t))$ for the action policy and initialize the Q-values with 0.0. Run the algorithm for 1,000 episodes and evaluate the current action values at every 10 episodes during learning using *rmse.m* and *greedy_eval.m*. Plot the evaluation results averaged over 10 trials against episode number. Experiment the algorithm with $\alpha \in \{0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$ and with different values for n_0. What are the best values for the parameters according to your evaluation results? Compare the performance of Q-learning to the one of Monte Carlo control.