CS440: Artificial Intelligence **Fall 2014**

Problem Set 2

Mariya Vasileva *Handed In: September 29, 2014*

1. Problem 1: WorldWithoutThief

(a) A purely-greedy Q-Learning agent with $\epsilon = 0.0$ will always select the optimal action a^* at every state s, which will result in the highest utility at the subsequent state, $Q^*(s', a')$. In other words, the agent selects the "safest" possible action, and finds and exploits a local optimum, rather than continue to explore the world for a globally-optimal solution. Thus, the agent is guaranteed to get "stuck" in an infinite loop, since there is no randomness in the choice of action in each state, and the purely-greedy agent's behavior is deterministic.

The rewards associated with different positions int he 5×5 grid of the world vary, since some positions are associated with negative rewards, i.e. penalties, that depend on the value of the slipperiness. Every # position has a slipperiness ranging from 0.0 to 1.0, and when the agent steps into a # position, it has a probability equal to the slipperiness that it may drop and break the packages, thus incurring a negative reward. What we observe when running the simulator with $\epsilon = 0.0$, learning rate $\alpha = 0.1$, and reward discount factor $\gamma = 0.9$, is that the agent initially incurs negative rewards, since while exploring the environment, it enters the "slippery" positions and "learns" that they lead to negative rewards.

Since utility is a function of the reward in the current state and the discounted rewards for all possible future states, then entering a "slippery" position means the greedy agent learns that position has a lower utility value, and never again selects an action that will lead to that position. This is why, after the initial exploration of the "slippery" positions, the greedy agent never returns to them, but instead selects actions that result in higher utility. Since all Q-values, i.e. all $Q(s, a)$, were initialized to zero, the agent continues to repeatedly obtain zero rewards.

After several hundred iterations, we see that the agent finds a local optimum. (Through simulations, we see that one such scenario is when the agent gets "stuck" in an infinite loop, oscillating back and forth between two "safe" states in order to avoid incurring negative rewards by entering into a "slippery" position and dropping the packages. Another observed scenario is that the agent never moves away from the initial position, because it always selects the optimal action of going south, and hits the wall. Yet another observed scenario is that the agent gets stuck in position (5, 2), again always selecting the optimal action of going south, and hitting the wall.)

Hence, being purely greedy in this case means always selects the optimal action that will not result in a lower utility. That is, since the greedy agent is being deterministic and not exploring the environment, it never "learns" how to attain positive rewards, which explains what we observe.

(b) By definition of $\epsilon - greedy$ Q-Learning, the $\epsilon - greedy$ policy takes a random action with the probability of $\epsilon \in [0,\ 1]$, and takes the estimated optimal, or *greedy*, action with a probability of $(1 - \epsilon)$ at each step. Setting ϵ to 0.1, means that the agent will select an action at random with probability of 0.1 at each state s, and will behave greedy, i.e. select the optimal action resulting in maximum utility, with probability of 0.9 at each state.

Hence, we observe that the agent receives negative rewards in the beginning while exploring the environment, and then continues to receive positive rewards. The agent follows the optimal policy 90% of the time, but the randomness introduced by ϵ causes it not to behave optimally 10% of the time, which is why we occasionally observe the total reward decreasing as the agent gets penalized for exploring a state with a lower utility value. We see that randomness allows the agent to get out of infinite loops around a local optimum. But we also see that randomness does not allow the agent to converge to ever-increasing positive rewards, because it has exhaustively searched through all states, the ϵ randomness interferes with taking the optimal policy.

(c) The higher the value of ϵ, the larger the probability that the agent will select an action at random at any given state, thus increasing the chances for exploration, but decreasing the agent's ability to follow the optimal policy. A high value of ϵ means that the agent will explore non-optimal options more often and learn from them, instead of being always deterministic in its choice of action, as in the purely-greedy algorithm. Setting ϵ to 0.5 means that the agent is choosing an action at random 50% of the time, and only following the optimal policy the other 50%. This explains why we see the agent incurring mostly negative rewards.

2. Problem 2: WorldWithThief

(a) Setting ϵ to 0.05 means that we introduce some randomness in our Q-Learning algorithm to allow the agent to not get stuck onto a local optimum, but explore the world and find a global one. However, since we are now operating in a world with thief *but* have no knowledge of his position. The agent takes a random action 5% of the time, and follows the optimal policy 95% of the time, but it still does not learn very well and incurs negative rewards, because the unknown position of the thief is interfering with its knowledge of the environment, i.e. the reward associated with each state appears "random" to the agent and prevents it from learning.

(b) We are now operating in a world with thief *and* have knowledge of his position, the possible states have increased times the number of possible positions the thief can occupy at any given state, which is equal to the number of rows - 5. The agent takes a random action 5% of the time, and follows the optimal policy 95% of the time. Since the position of the thief is known, the $\epsilon - greedy$ agent will capture a much more accurate utility value for each state. The reward associated with each state and playing a role in the utility of that state no longer appears "random" to the agent, but can be learned. We see that initially, during the exploration stage, the agent incurs negative rewards, but they quickly start exhibiting an increasing trend, and reach very high values towards the end of the simulation. Because of the increased state space information, the agent, now being aware of the thief's position, has more informed estimates of the utility for each state, and is able to choose a much more accurate policy at each given state.

(c) The learning rate determines to what extent the newly acquired information will override the old information. A rate of $\alpha = 0$ will make the agent not learn anything, while a rate of $\alpha = 1$ would make the agent consider only the most recent information.

The learning rate reflects the relative confidence in the old and the new information, and hence a high α means that we are placing higher importance on the utility of our most recent state, $Q(s, a)$, in estimating the utility of a future state, $Q(s', a')$.

Conversely, a low value of α means we are not confident that our estimate of the future state utility is sufficiently accurate, and hence are placing lesser importance on the utility of our most recent state in predicting the utility of a future state.

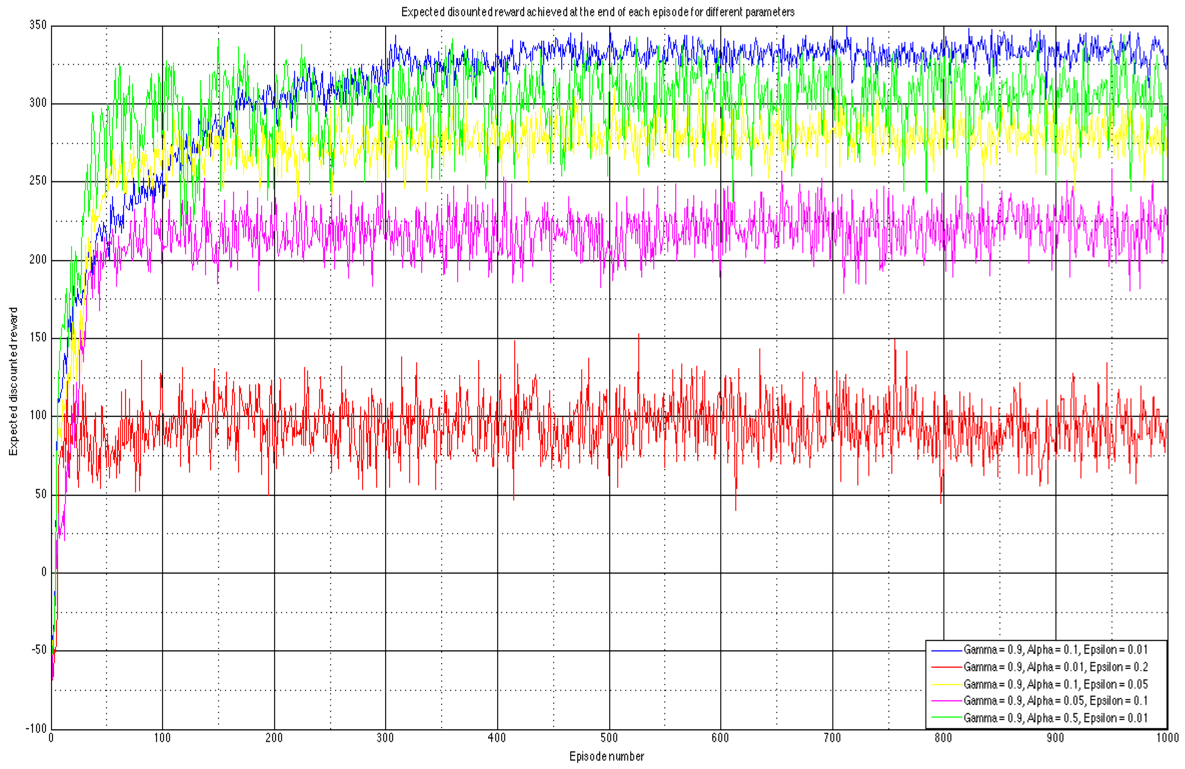
Through experimentation, we arrive at the optimal value of α for our Q-Learning agent being 0.1, as demonstrated in the plots below.

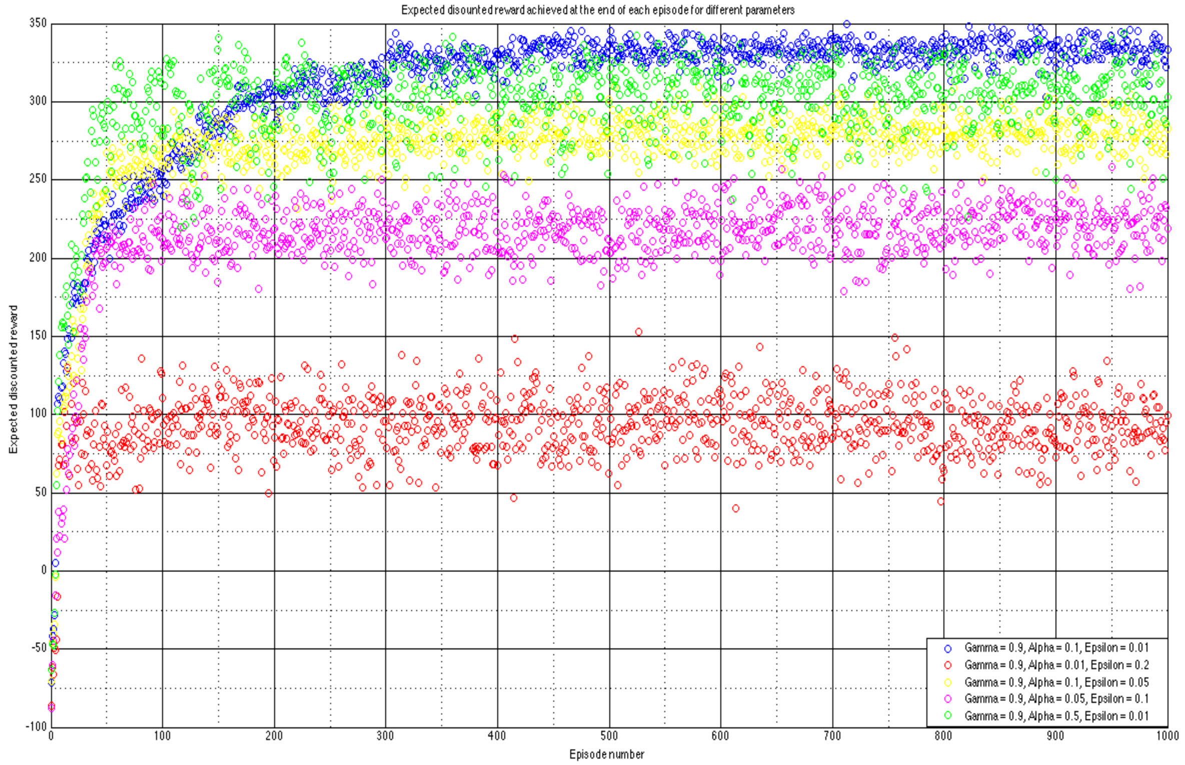
As discussed earlier, the value of ϵ determines the amount of randomness we introduce in the selection of action a at each state s, i.e. it defines the tradeoff between exploitation and exploration. If we choose $\epsilon > 0.5$, then our agent will select a random action more often than an optimal one, and vice versa. Hence, a larger ϵ results in more randomness in action selection at each state, which may not lead to optimal solutions - while a smaller ϵ results in the agent selecting an optimal action $a*$ leading to maximum utility of the next state, $Q^*(s, a^*)$, more often than acting at random.

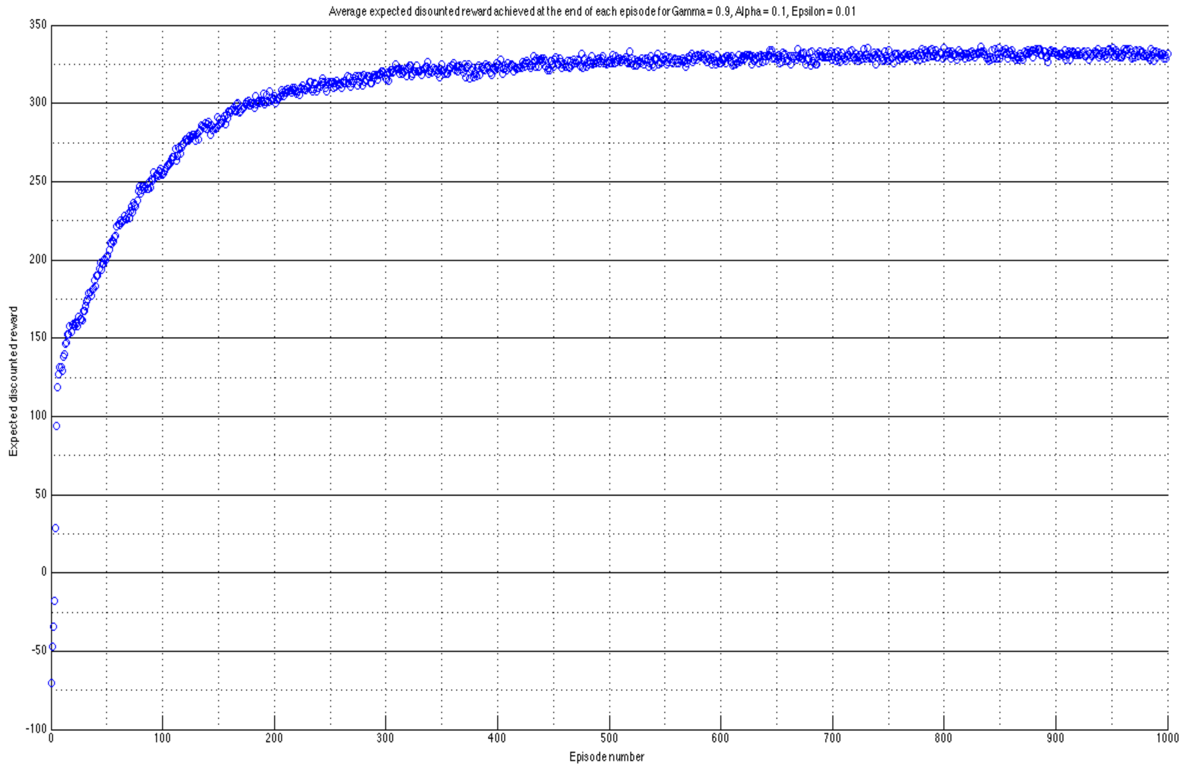
Through experimentation, we determine the optimal value of ϵ for our Q-Learning agent is 0.01, meaning that we introduce just as much randomness in action selection as required to not converge to local optima, but explore until a globally-optimal solution is reached.

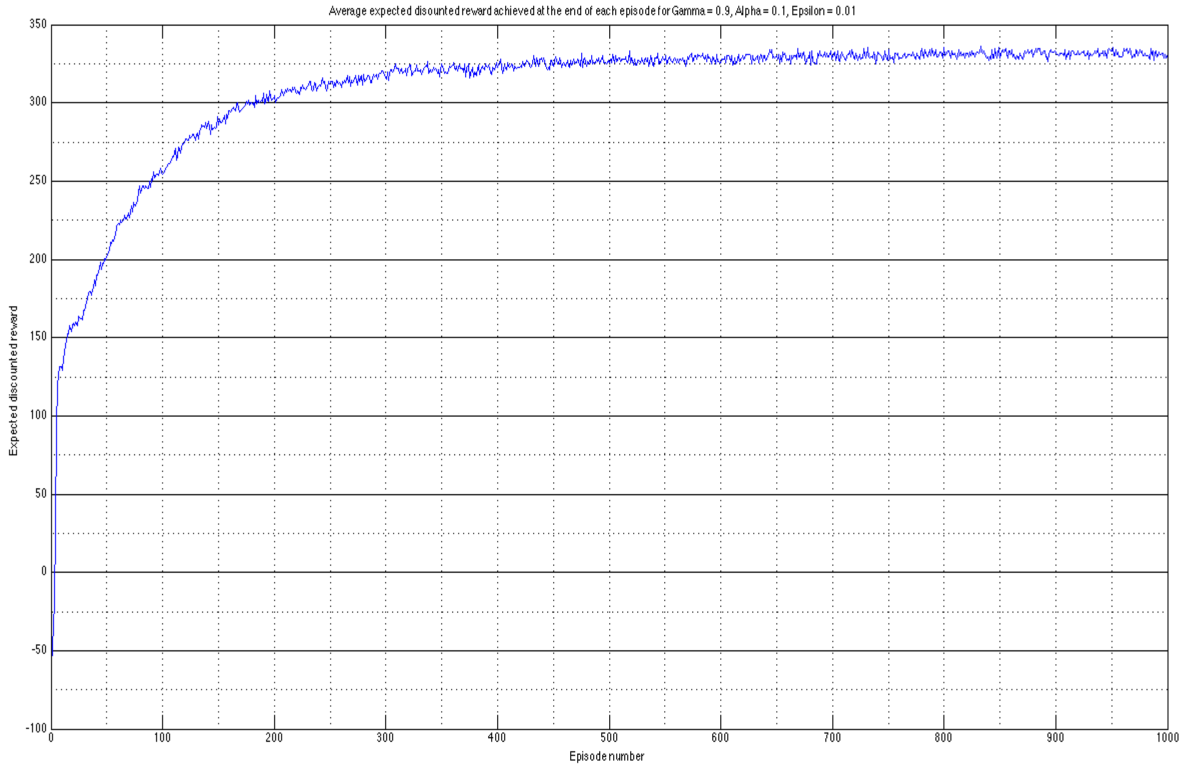
3. After averaging the expected discounted rewards over ten simulations with the same optimal parameters, $\gamma = 0.9, \alpha = 0.1, \epsilon = 0.01$, we observe that the noise we noticed when we plotted out individual simulation results has decreased significantly. The effect of the ϵ randomness on the expected reward is "flattened out" by the averaging of our results.

   The first set of plots below demonstrate our results for several different sets of parameters. We notice that even for the optimal set of parameters, the ϵ randomness introduces a lot of noise to the rewards values, even towards the end of the simulation when the results tend to convergence. The second set of plots demonstrates the average result of ten simulations for *only* the *optimal* set of parameters. We notice that the noise observed previously has significantly decreased, which is expected due to averaging. This implies that averaging several iterations of the same optimal policy might be a good way to filter out the random noise, inevitably caused by the definition of $\epsilon - greedy$ Q-Learning.


Expected disounted reward achieved at the end of each episode for different parameters
Expected discounted reward
350
300
250
200
150
100
50
0
-50
-100
0
100
200
300
400
500
600
700
800
900
1000
Episode number
Gamma = 0.9, Alpha = 0.1, Epsilon = 0.01
Gamma = 0.9, Alpha = 0.01, Epsilon = 0.2
Gamma = 0.9, Alpha = 0.1, Epsilon = 0.05
Gamma = 0.9, Alpha = 0.05, Epsilon = 0.1
Gamma = 0.9, Alpha = 0.5, Epsilon = 0.01


Expected disounted reward achieved at the end of each episode for different parameters
Expected discounted reward
350
300
250
200
150
100
50
0
-50
-100
0
100
200
300
400
500
600
700
800
900
1000
Episode number
Gamma = 0.9, Alpha = 0.1, Epsilon = 0.01
Gamma = 0.9, Alpha = 0.01, Epsilon = 0.2
Gamma = 0.9, Alpha = 0.1, Epsilon = 0.05
Gamma = 0.9, Alpha = 0.05, Epsilon = 0.1
Gamma = 0.9, Alpha = 0.5, Epsilon = 0.01


Average expected disounted reward achieved at the end of each episode for Gamma = 0.9, Alpha = 0.1, Epsilon = 0.01
Expected discounted reward
350
300
250
200
150
100
50
0
-50
-100
0
100
200
300
400
500
600
700
800
900
1000
Episode number


Average expected disounted reward achieved at the end of each episode for Gamma = 0.9, Alpha = 0.1, Epsilon = 0.01
Expected discounted reward
350
300
250
200
150
100
50
0
-50
-100
0
100
200
300
400
500
600
700
800
900
1000
Episode number