RLAI

Enigma Dimitri

October 29, 2024

1 Introduction

Exercise 1.1: Self-Play Suppose, instead of playing against a random opponent, the reinforcement learning algorithm described above played against itself, with both sides learning. What do you think would happen in this case? Would it learn a different policy for selecting moves?

Answer

It would of course learn a different policy for selecting movies because the behavior of the opponent would totally not be the same. Now what I think would happen is that both sides would learn how to counter the other and pretty much all runs would end up in ties. □

Exercise 1.2: Symmetries Many tic-tac-toe positions appear different but are really the same because of symmetries. How might we amend the learning process described above to take advantage of this? In what ways would this change improve the learning process? Now think again. Suppose the opponent did not take advantage of symmetries. In that case, should we? Is it true, then, that symmetrically equivalent positions should necessarily have the same value?

Answer

In theory we might think that if we code the symmetries to be the exact same state and change the actions accordingly then we would probably need way less examples to actually learn how to play the game. It would also make the algorithm slightly better because in average the best cases will also have more examples. Now in practice if the opponent do not use the symmetries then we should also not use them as his game may be rigged toward a particular axis. In this sense, if we abuse symmetries, we may miss which axis is the weakest for the opponent. Hence, in this case, symmetrically equivalent positions should not have the same value by default. □

Exercise 1.3: Greedy Play Suppose the reinforcement learning player was *greedy*, that is, it always played the move that brought it to the position that it rated the best. Might it learn to play better, or worse, than a nongreedy player? What problems might occur?

Answer

It would be bad for the algorithm. Indeed, convinced that it's moves are always the best, it would never know if some others may actually lead to a better reward. The algorithm would be stuck in a confirmation bias and actually never truly see what is the objective best maneuver. The nongreedy player would at the opposite explore different paths and sometimes find new solutions that are actually more efficient. □

Exercise 1.4: Learning from Exploration Suppose learning updates occurred after *all* moves, including exploratory moves. If the step-size parameter is appropriately reduced over time (but not the tendency to explore), then the state values would converge to a different set of probabilities. What (conceptually) are the two sets of probabilities computed when we do, and when we do not, learn from exploratory moves? Assuming that we do continue to make exploratory moves, which set of probabilities might be better to learn? Which would result in more wins?

Answer

I am going to wildly guess on this one. If we learn from every moves then in theory I guess we are going to learn the set of probabilities that are best because the influence of exploratory moves will fade over time (as the step-size is reduced). This will still be less optimal than the standard case because we'll take into account bad moves done by exploration. If we keep making exploratory moves, probably the classic probabilities will be better because we'll not use exploratory moves (which are bad) to influence the actual important decisions. □

Exercise 1.5: Other Improvements Can you think of other ways to improve the reinforcement learning player? Can you think of any better way to solve the tic-tac-toe problem as posed?

Answer

I think of two things right now, the first one is to decrease the exploration probability over time because if the opponent doesn't change his strategy, at some point we will not need to explore anything anymore because he will not adapt to our learning. The second thing is to use intermediate rewards to not only reward the whole game sequence but specific states of the game leading toward the win. □

---

2 Multi-armed Bandits

Exercise 2.1 In ε-greedy action selection, for the case of two actions and $\varepsilon = 0.5$, what is the probability that the greedy action is selected?

Answer

$$\begin{aligned}
\Pr\{\text{Greedy Selected}\} &= \Pr\{(\text{Greedy Selected} \cap \text{Greedy Play}) \cup (\text{Greedy Selected} \cap \text{Random Play})\} \\
&= \Pr\{\text{Greedy Selected} \cap \text{Greedy Play}\} + \Pr\{\text{Greedy Selected} \cap \text{Random Play}\} \\
&= \Pr\{\text{Greedy Play}\} \times \Pr\{\text{Greedy Selected}|\text{Greedy Play}\} \\
&+ \Pr\{\text{Random Play}\} \times \Pr\{\text{Greedy Selected}|\text{Random Play}\} \\
&= (1 - \varepsilon) + \varepsilon/k
\end{aligned}$$

For $\varepsilon = 0.5$ and $k = 2$, we get a probability of $0.5 + 0.25 = 0.75$. □

---

*Exercise 2.** Reproduce the results of **Figure 2.1** as a proof of correct implementation.

Answer

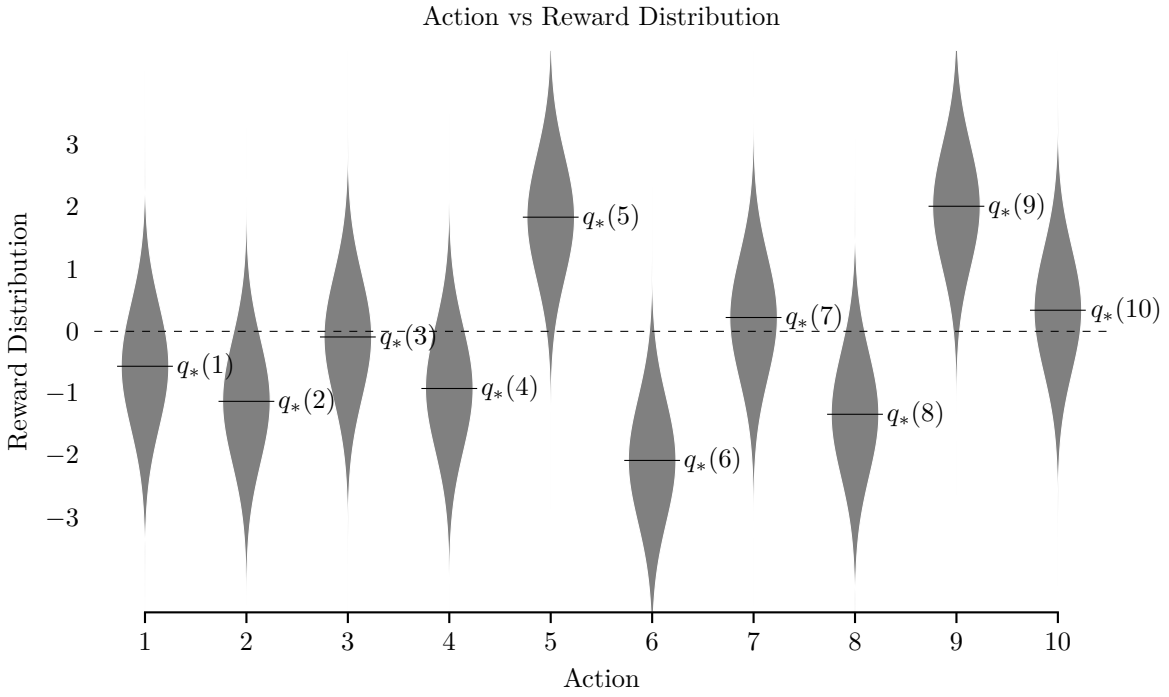

Figure 2.1: An example bandit problem from the 10-armed testbed. The true value $q_*(a)$ of each of the ten actions was selected according to a normal distribution with mean zero and unit variance, and then the actual rewards were selected according to a mean $q_*(a)$ unit variance normal distribution, as suggested by these gray distributions.

□

---

*Exercise 2.** Reproduce the results of **Figure 2.2** as a proof of correct implementation.

Answer

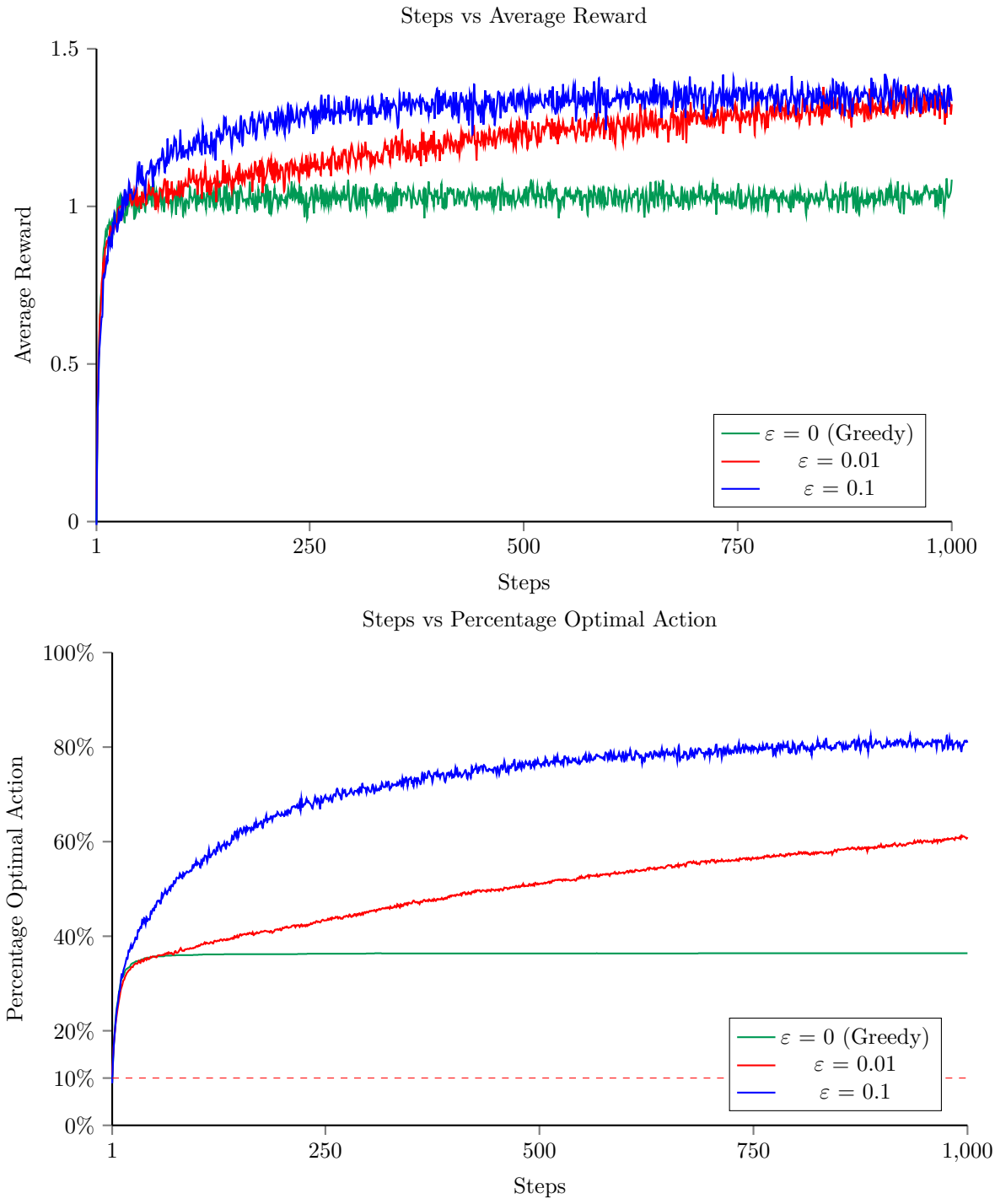

Figure 2.2: Average performance of ε-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates.

□

Exercise 2.2: Bandit Example Consider a k-armed bandit problem with $k = 4$ actions, denoted 1, 2, 3, and 4. Consider applying to this problem a bandit algorithm using ε-greedy action selection, sample-average action-value estimates, and initial estimates of $Q_1(a) = 0$, for all a. Suppose the initial sequence of actions and rewards is $A_1 = 1$, $R_1 = 1$, $A_2 = 2$, $R_2 = 1$, $A_3 = 2$, $R_3 = 2$, $A_4 = 2$, $R_4 = 2$, $A_5 = 3$, $R_5 = 0$. On some of these time steps the ε case may have occurred, causing an action to be selected at random. On which time steps did this definitely occur? On which time steps could this possibly have occurred?

Answer:

Let's write the values at each step to understand what happens.
For $t = 1$:
$Q_1(1) = 0$,
$Q_1(2) = 0$,
$Q_1(3) = 0$,
$Q_1(4) = 0$.
Here we have an egality of values so the play $A_1 = 1$ can be greedy or not.
For $t = 2$:
$Q_2(1) = 1$,
$Q_2(2) = 0$,
$Q_2(3) = 0$,
$Q_2(4) = 0$.
Here we have a clear winner which is the first arm. $A_2 = 2$ so we did not select the best arm so the selection is for sure random here.
For $t = 3$:
$Q_3(1) = 1$,
$Q_3(2) = 1$,
$Q_3(3) = 0$,
$Q_3(4) = 0$.
Here we have two winners which are the first and second arms. $A_3 = 2$ so the play can be greedy or not.
For $t = 4$:
$Q_4(1) = 1$,
$Q_4(2) = 1.5$,
$Q_4(3) = 0$,
$Q_4(4) = 0$.
Here we have a clear winner which is the second arm. $A_4 = 2$ so the play can be greedy or not.
For $t = 5$:
$Q_5(1) = 1$,
$Q_5(2) = 1.67$,
$Q_5(3) = 0$,
$Q_5(4) = 0$.
Here we have a clear winner which is the second arm. $A_5 = 3$ so we did not select the best arm so the selection is for sure random here. □

Exercise 2.3 In the comparison shown in Figure 2.2, which method will perform best in the long run in terms of cumulative reward and probability of selecting the best action? How much better will it be? Express your answer quantitatively.

Answer:

In the long run, both methods will converge in the sense that both methods will know which action is the optimal best action. So in the long run, for $\varepsilon = 0.1$ we will have $\Pr\{A_*\} = \Pr\{\text{Greedy}\} = 0.9$. Similarly for $\varepsilon = 0.01$ we will have $\Pr\{A_*\} = \Pr\{\text{Greedy}\} = 0.99$. For the rewards, if we call q_* the

optimal mean, the cumulative reward will look respectively like $t \times 0.9 \times q_*$ (the random play being canceled out by the fact that the average of the means is 0) and $t \times 0.99 \times q_*$ so the case $\varepsilon = 0.01$ will be $0.99/0.9 = 1.1$ better.

$$Q_{n+1} = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i. \tag{2.6}$$

□

Exercise 2.4 If the step-size parameters, α_n, are not constant, then the estimate Q_n is a weighted average of previously received rewards with a weighting different from that given by (2.6). What is the weighting on each prior reward for the general case, analogous to (2.6), in terms of the sequence of step-size parameters?

Answer:

$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha_n \Big[R_n - Q_n \Big] \\
&= \alpha_n R_n + (1-\alpha_n) Q_n \\
&= \alpha_n R_n + (1-\alpha_n)(\alpha_{n-1} R_{n-1} + (1-\alpha_{n-1}) Q_{n-1}) \\
&= \alpha_n R_n + \sum_{i=1}^{n-1} \alpha_i R_i \prod_{j=i+1}^{n} (1-\alpha_j) + Q_1 \prod_{i=1}^{n} (1-\alpha_i).
\end{aligned}$$

The weighting is therefore α_n for $i = n$ and $\alpha_i \prod_{j=i+1}^{n}(1-\alpha_j)$ for $1 \leq i < n$. For Q_1 the weighting is $\prod_{i=1}^{n}(1-\alpha_i)$.

Now let's prove that the sum of all of these elements is 1. We are going to prove that recursively, for the initial values of $n = 1$ and $n = 2$ it is very easy to see ($\alpha_1 + (1-\alpha_1) = 1$ and $\alpha_2 + \alpha_1(1-\alpha_2) + (1-\alpha_1)(1-\alpha_2) = 1$). So let's assume that we have:

$$u_n \doteq \alpha_n + \sum_{i=1}^{n-1} \alpha_i \prod_{j=i+1}^{n} (1-\alpha_j) + \prod_{i=1}^{n} (1-\alpha_i) = 1.$$

Let's prove that:

$$u_{n+1} \doteq \alpha_{n+1} + \sum_{i=1}^{n} \alpha_i \prod_{j=i+1}^{n+1} (1-\alpha_j) + \prod_{i=1}^{n+1} (1-\alpha_i) = 1.$$

We have:

$$\begin{aligned}
u_{n+1} &= \alpha_{n+1} + (1-\alpha_{n+1}) \sum_{i=1}^{n-1} \alpha_i \prod_{j=i+1}^{n} (1-\alpha_j) + (1-\alpha_{n+1}) \prod_{i=1}^{n} (1-\alpha_i) + \alpha_n (1-\alpha_{n+1}) \\
&= \alpha_{n+1} + (1-\alpha_{n+1})(u_n - \alpha_n) + +\alpha_n (1-\alpha_{n+1}) \\
&= \alpha_{n+1} + (1-\alpha_{n+1})(1-\alpha_n) + +\alpha_n (1-\alpha_{n+1}) \\
&= 1.
\end{aligned}$$

□

Exercise 2.5 (programming) Design and conduct an experiment to demonstrate the difficulties that sample-average methods have for nonstationary problems. Use a modified version of the 10-armed testbed in which all the $q_*(a)$ start out equal and then take independent random walks (say by adding a normally distributed increment with mean zero and standard deviation 0.01 to all the $q_*(a)$ on each step). Prepare plots like Figure 2.2 for an action-value method using sample averages, incrementally

computed, and another action-value method using a constant step-size parameter, $\alpha = 0.1$. Use $\varepsilon = 0.1$ and longer runs, say of $10,000$ steps.

Answer

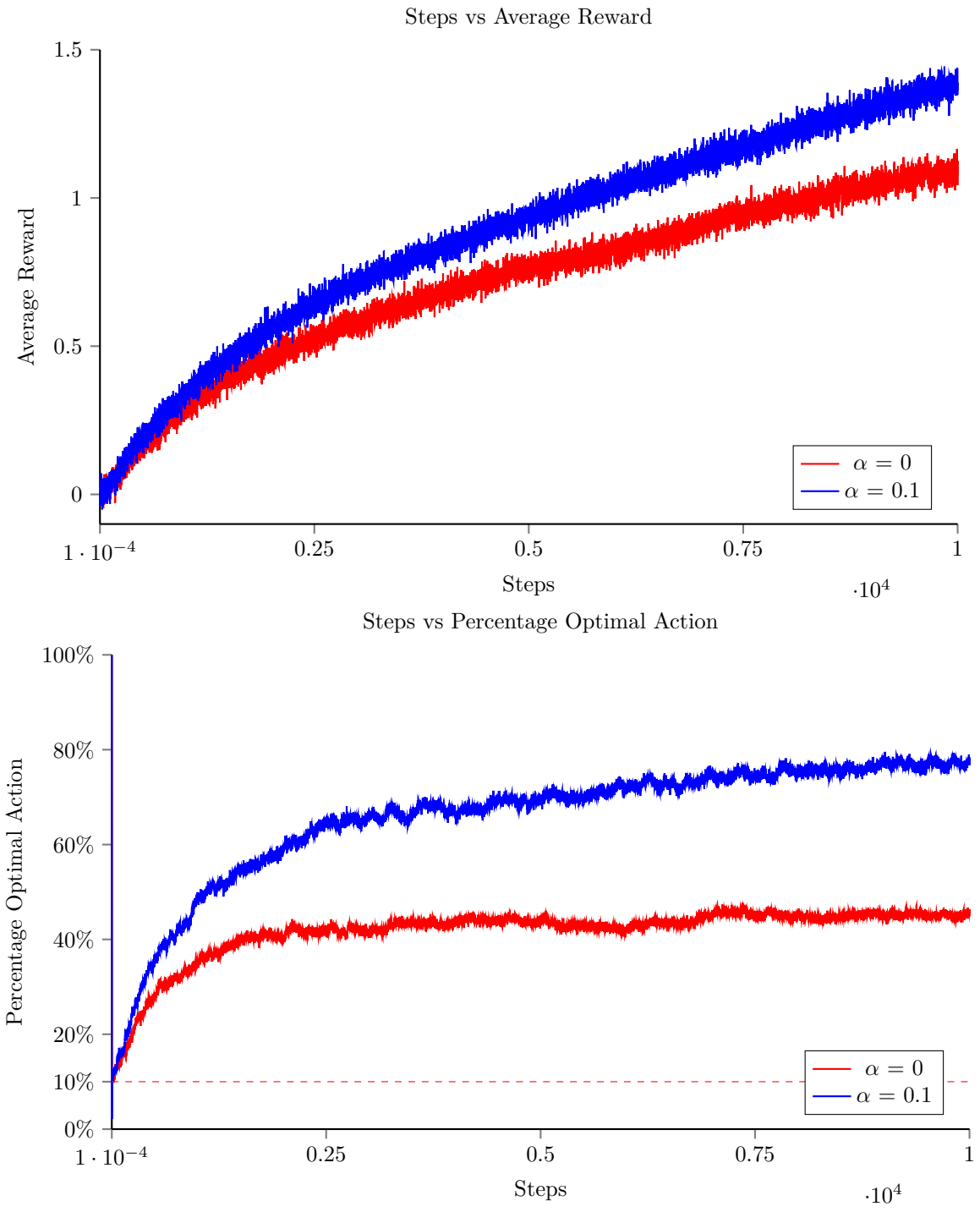

Figure 2.2b: Average performance of 0.1-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems.

□

*Exercise 2.** Reproduce the results of **Figure 2.3** as a proof of correct implementation.

Answer

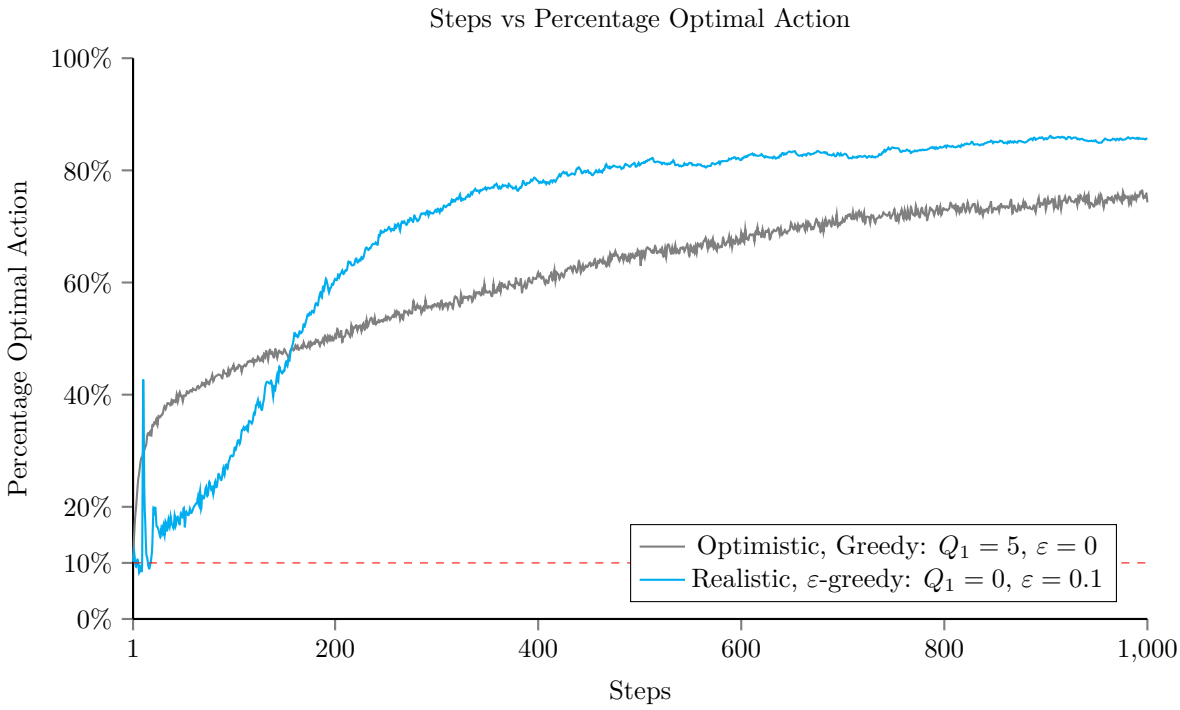

Figure 2.3: The effect of optimistic initial action-value estimates on the 10-armed testbed. Both methods used a constant step-size parameter, $\alpha = 0.1$.

□

Exercise 2.6 Mysterious Spikes The results shown in Figure 2.3 should be quite reliable because they are averages over 2000 individual, randomly chosen 10-armed bandit tasks. Why, then, are there oscillations and spikes in the early part of the curve for the optimistic method? In other words, what might make this method perform particularly better or worse, on average, on particular early steps?

Answer:

The reason is very simple, the initial value is in average way higher than the potential outcomes of our arms so at the beginning we will try arms, realize that their actual rewards is lower than our initial estimate so pick another one until we tried every arms. At this moment in average we'll identify which one is the better by averaging 5 and the first reward so on step 11 we pick the best arm hence the spike. But then again we realize that the actual mean is lower that what we think so we try the other arms until the effect of the initial estimate is finally diminished enough to correctly identify the best arm. Let's give an example with 3 arms which are constant equal to 0.1, 0.2 and 0.3. On step 0 we randomly pick one, let's say the third one, the value then becomes for the third arm $(5+0.3)/2 = 2.65$ which is bad compared to the values of the first and second arms. We pick one of the best arms, let's say the second one, the value then becomes for the second arm $(5 + 0.2)/2 = 2.6$ which is bad compared to the value of the first arm so we'll pick the first one. The value becomes for the first arm $(5 + 0.1)/2 = 2.55$. Now we have "good estimates" so we pick the actual best arm which is the third one so the value for the third one becomes $(5 + 0.3 + 0.3)/3 = 1.87$ which is lesser than the first and

second arms, etc. □

Exercise 2.7: Unbiased Constant-Step-Size Trick In most of this chapter we have used sample averages to estimate action values because sample averages do not produce the initial bias that constant step sizes do (see the analysis leading to (2.6)). However, sample averages are not a completely satisfactory solution because they may perform poorly on nonstationary problems. Is it possible to avoid the bias of constant step sizes while retaining their advantages on nonstationary problems? One way is to use a step size of

$$\beta_n \doteq \alpha/\bar{o}_n, \tag{2.8}$$

to process the nth reward for a particular action, where $\alpha > 0$ is a conventional constant step size, and $\bar{o}_n$ is a trace of one that starts at 0:

$$\bar{o}_n \doteq \bar{o}_{n-1} + \alpha(1 - \bar{o}_{n-1}), \text{ for } n \geq 0, \text{ with } \bar{o}_0 \doteq 0. \tag{2.8}$$

Carry out an analysis like that in (2.6) to show that Q_n is an exponential recency-weighted average without initial bias.

Answer:

Using the result of *Exercise 2.4*:

$$\begin{aligned}
Q_{n+1} &= \beta_n R_n + \sum_{i=1}^{n-1} \beta_i R_i \prod_{j=i+1}^{n} (1-\beta_j) + Q_1 \prod_{i=1}^{n} (1-\beta_i) \\
&= \beta_n R_n + \sum_{i=1}^{n-1} \beta_i R_i \prod_{j=i+1}^{n} (1-\beta_j) + Q_1 \prod_{i=1}^{n-1} (1-\beta_i)(1-\beta_1) \\
&= \beta_n R_n + \sum_{i=1}^{n-1} \beta_i R_i \prod_{j=i+1}^{n} (1-\beta_j) + Q_1 \prod_{i=1}^{n-1} (1-\beta_i)(1-\alpha/\bar{o}_1) \\
&= \beta_n R_n + \sum_{i=1}^{n-1} \beta_i R_i \prod_{j=i+1}^{n} (1-\beta_j) + Q_1 \prod_{i=1}^{n-1} (1-\beta_i)(1-\alpha/(\bar{o}_0 + \alpha(1-\bar{o}_0)) \\
&= \beta_n R_n + \sum_{i=1}^{n-1} \beta_i R_i \prod_{j=i+1}^{n} (1-\beta_j) \\
&= \beta_n R_n + \sum_{i=1}^{n-1} \beta_i R_i \prod_{j=i+1}^{n} (1-\alpha/\bar{o}_j) \\
&= \beta_n R_n + \sum_{i=1}^{n-1} \beta_i R_i \prod_{j=i+1}^{n} ((\bar{o}_j - \alpha)/\bar{o}_j) \\
&= \beta_n R_n + \sum_{i=1}^{n-1} \beta_i R_i \prod_{j=i+1}^{n} (1-\alpha)\bar{o}_{j-1}/\bar{o}_j \\
&= \beta_n R_n + \sum_{i=1}^{n-1} \beta_i R_i (1-\alpha)^{n-i} \bar{o}_i/\bar{o}_n = \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i/\bar{o}_n.
\end{aligned}$$

□

---

*Exercise 2.** Reproduce the results of **Figure 2.4** as a proof of correct implementation.

Answer

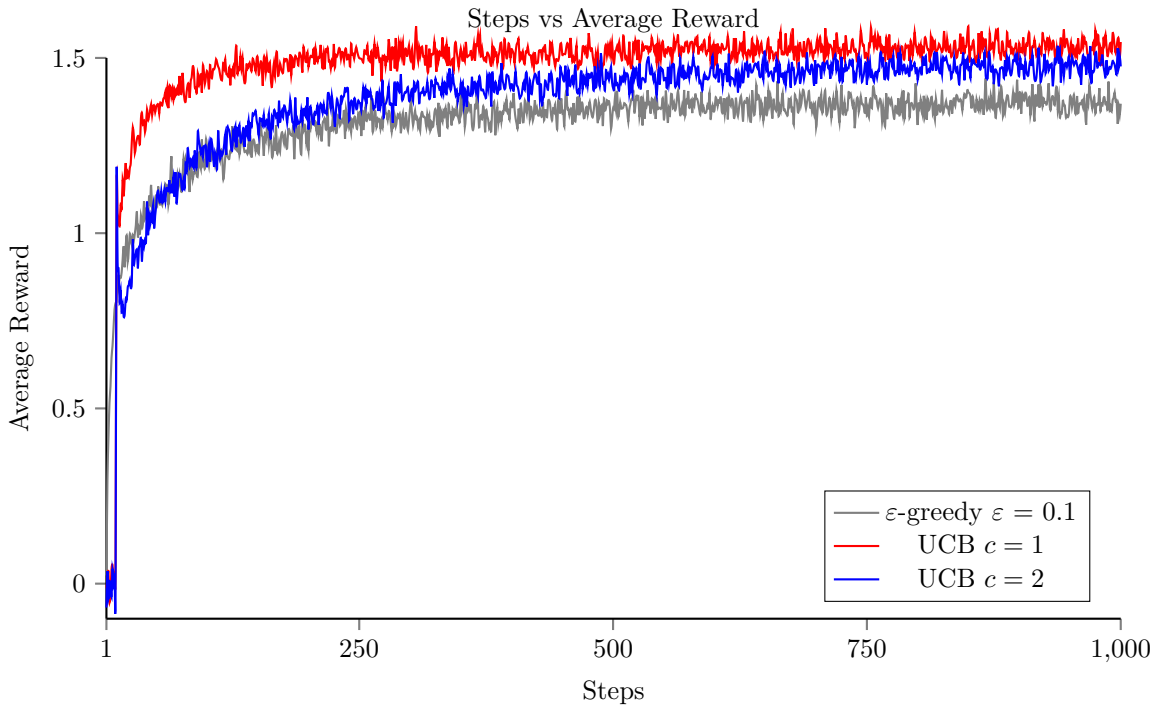

Figure 2.4: Average performance of UCB action selection on the 10-armed testbed. As shown, UCB generally performs better than ε-greedy action selection, except in the first k steps, when it selects randomly among the as-yet-untried actions.

□

Exercise 2.8: UCB Spikes In Figure 2.4 the UCB algorithm shows a distinct spike in performance on the 11th step. Why is this? Note that for your answer to be fully satisfactory it must explain both why the reward increases on the 11th step and why it decreases on the subsequent steps. Hint: if $c = 1$, then the spike is less prominent.

Answer:

This is explained almost like the scenario in which we changed the initial value. Indeed, for the first 10 steps we explore every possibility, then at the 11th step we use our previous knowledge to use the best action which in average will be the actual best. But then because the second term of the formula $c\sqrt{\ln t/N_t(a)}$ is big (for $c = 2$) then the disillusion of picking the best action again will drop the value too much (because $N_t(a)$ will be bigger and because c also) and give credit to unplayed actions. If c is 1 the phenomenon is less extreme because we give less credit to exploration. To give a small numerical example, let's see the case where we have two constant actions of mean 0 and 0.5. At $t = 3$ we would have the value of interest of 0 being $c\sqrt{\ln 3}$ and the value of interest of 1 being $0.5+c\sqrt{\ln 3}$ so we would pick the actual optimal action. Then on time $t = 4$, the value of interest of 0 would be $c\sqrt{\ln 4}$ and the value of interest of 1 would be $0.5 + c\sqrt{\ln 4}/2$. Visually then, we would have this order based on the value of c:

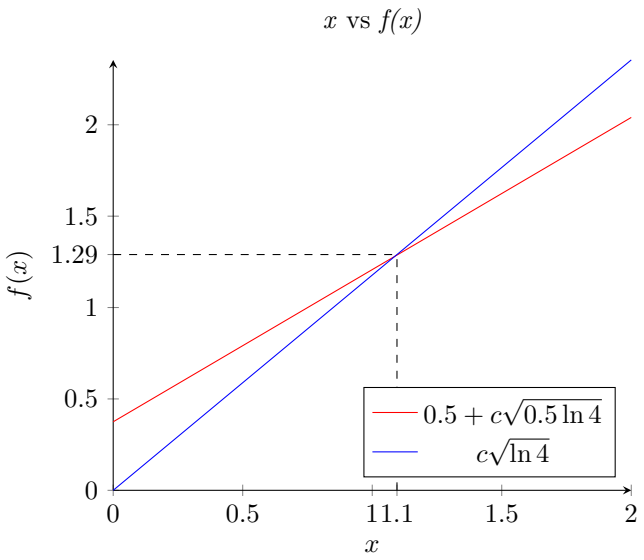

Figure 2.4b: Study of the behavior of the UCB maximization.

We can see in this case that for $c = 2$ we would select the bad action whereas for $c = 1$ we would actually select the correct one. □

Exercise 2.9 Show that in the case of two actions, the soft-max distribution is the same as that given by the logistic, or sigmoid, function often used in statistics and artificial neural networks.

Answer:

$$\Pr\{A_t = 1\} = \frac{e^{H_t(1)}}{e^{H_t(1)} + e^{H_t(2)}} = \frac{1}{1 + e^{H_t(2) - H_t(1)}} = \frac{1}{1 + e^{-(H_t(1) - H_t(2))}} = \sigma(H_t(1) - H_t(2)),$$
$$\Pr\{A_t = 2\} = \sigma(H_t(2) - H_t(1)).$$

□

---

*Exercise 2.** Reproduce the results of **Figure 2.5** as a proof of correct implementation.

Answer

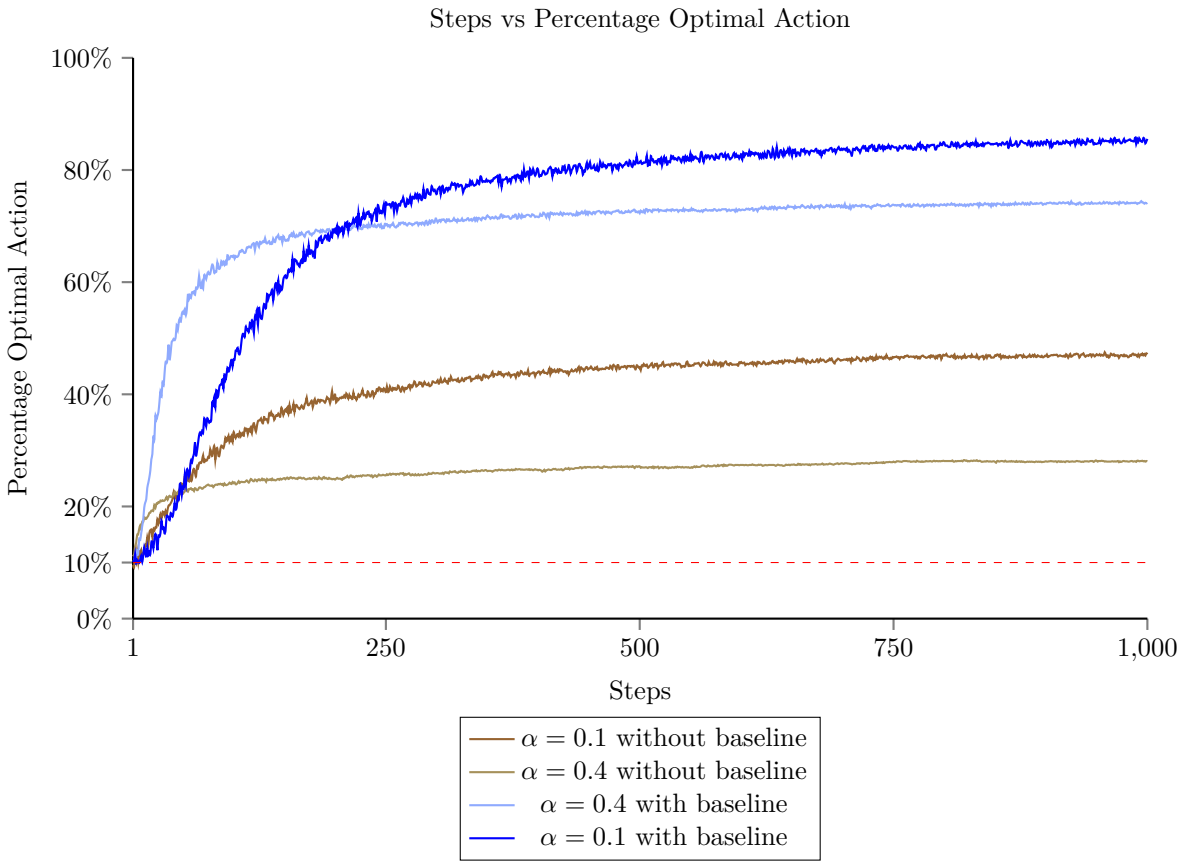

Figure 2.5: Average performance of the gradient bandit algorithm with and without a reward baseline on the 10-armed testbed when the $q_*(a)$ are chosen to be near +4 rather than near zero.

□

Exercise 2.10 Suppose you face a 2-armed bandit task whose true action values change randomly from time step to time step. Specifically, suppose that, for any time step, the true values of actions 1 and 2 are respectively 0.1 and 0.2 with probability 0.5 (case A), and 0.9 and 0.8 with probability 0.5 (case B). If you are not able to tell which case you face at any step, what is the best expectation of success you can achieve and how should you behave to achieve it? Now suppose that on each step you are told whether you are facing case A or case B (although you still don't know the true action values). This is an associative search task. What is the best expectation of success you can achieve in this task, and how should you behave to achieve it?

Answer

If you don't know in which scenario you are, whatever the strategy will always lead to the same reward in average: 0.5. Now if you know in which scenario you are then you will have to play 1 in case B and 2 in case A and in then you would have a reward of $0.5 \times 0.9 + 0.5 \times 0.2 = 0.55$. □

---

*Exercise 2.** Reproduce the results of **Figure 2.6** as a proof of correct implementation.

Answer

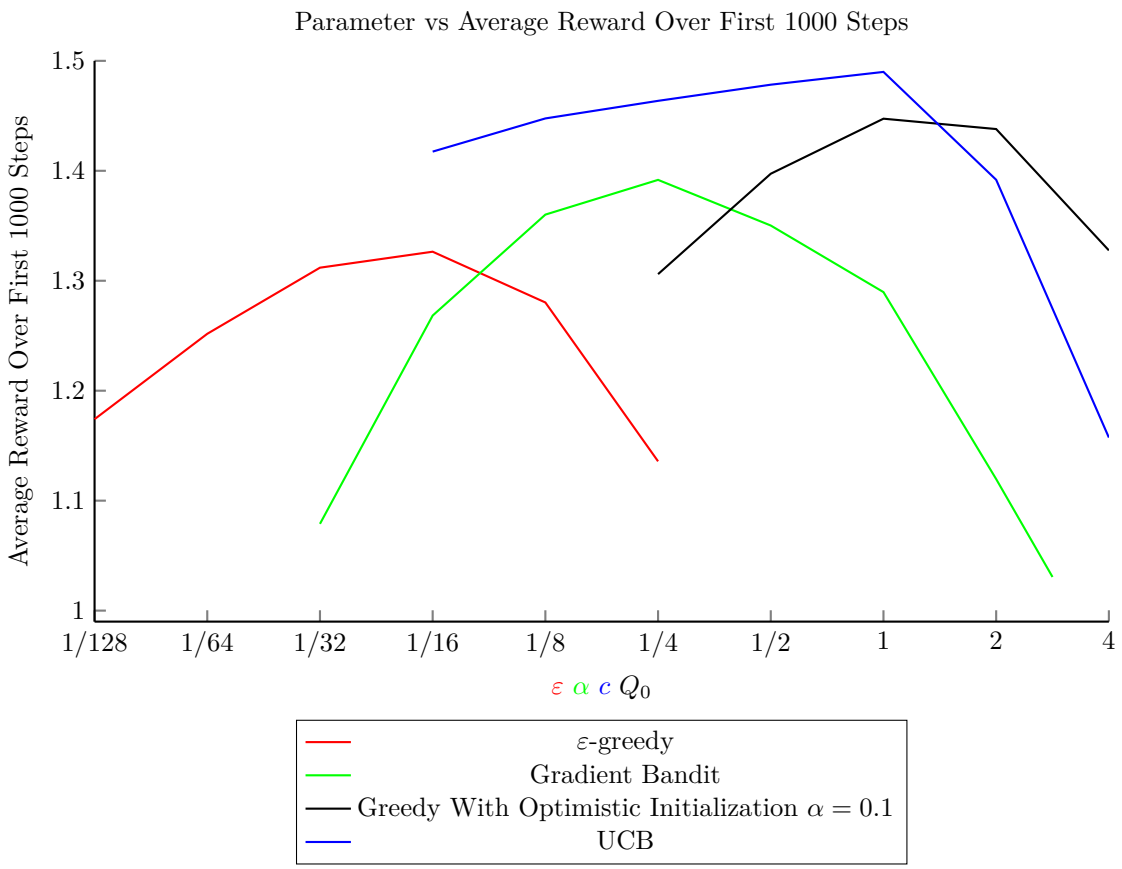

Figure 2.6: A parameter study of the various bandit algorithms presented in this chapter. Each point is the average reward obtained over 1000 steps with a particular algorithm at a particular setting of its parameter.

□

Exercise 2.11 (programming) Make a figure analogous to Figure 2.6 for the nonstationary case outlined in Exercise 2.5. Include the constant-step-size ε-greedy algorithm with $\alpha = 0.1$. Use runs of $200,000$ steps and, as a performance measure for each algorithm and parameter setting, use the average reward over the last $100,000$ steps.

Answer

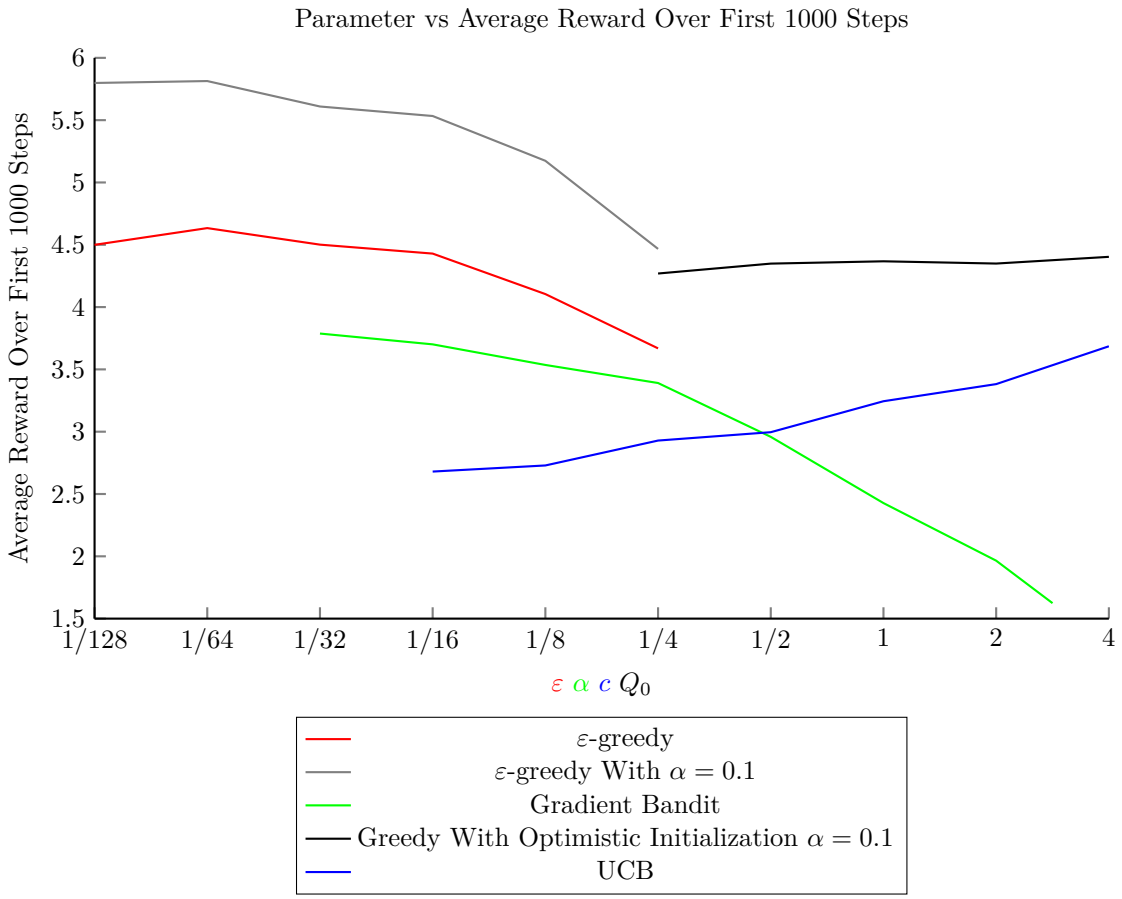

Figure 2.6b: A parameter study of the various bandit algorithms presented in this chapter. Each point is the average reward obtained over 100000 steps (with a total of 200000 steps) with a particular algorithm at a particular setting of its parameter.

□
