15 *Exploration and Exploitation*

Reinforcement learning agents[1] must balance *exploration* of the environment with *exploitation* of knowledge obtained through its interactions. Pure exploration will allow the agent to build a comprehensive model, but the agent will likely have to sacrifice the gathering of reward. Pure exploitation has the agent continually choosing the action it thinks best to accumulate reward, but there may be other, better actions that could be taken. This chapter introduces the challenges associated with the exploration-exploitation tradeoff by focusing on a problem with a single state. We conclude with introducing exploration in MDPs with multiple states.

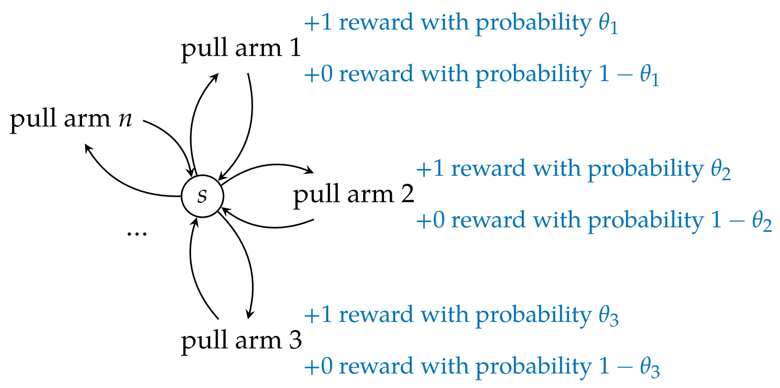
[1] A review of the field of reinforcement learning is provided in M. Wiering and M. van Otterlo, eds., *Reinforcement Learning: State of the Art*. Springer, 2012.

15.1 *Bandit Problems*

Early analyses of the exploration-exploitation tradeoff were focused on slot machines, also called *one-armed bandits*.[2] The name comes from older slot machines having a single pull lever and that the machine tends to take the gambler's money. Many real world problems can be framed as *multi-armed bandit problems*, such as the allocation of clinical trials and adaptive network routing. Many different bandit problem formulations exist in the literature, but this chapter will focus on what is called a *binary bandit*, *Bernoulli bandit*, or *binomial bandit*. In these problems, arm a pays off 1 with probability θ_a and 0 otherwise. Pulling an arm costs nothing, but we only have h pulls.

A bandit problem can be framed as an h-step Markov decision process with a single state, n actions, and an unknown reward function $R(s, a)$, as shown in figure 15.1. Recall that $R(s, a)$ is the expected reward when taking action a in s, but individual rewards realized in the environment may come from a probability distribution.

[2] These bandit problems were explored during World War II and proved exceptionally challenging to solve. According to Peter Whittle, "efforts to solve [bandit problems] so sapped the energies and minds of Allied analysts that the suggestion was made that the problem be dropped over Germany as the ultimate instrument of intellectual sabotage." J. C. Gittins, "Bandit Processes and Dynamic Allocation Indices," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 41, no. 2, pp. 148–177, 1979.



pull arm 1
$+1$ reward with probability θ_1
$+0$ reward with probability $1 - \theta_1$
pull arm n
s
...
pull arm 2
$+1$ reward with probability θ_2
$+0$ reward with probability $1 - \theta_2$
pull arm 3
$+1$ reward with probability θ_3
$+0$ reward with probability $1 - \theta_3$

Figure 15.1. The multi-armed bandit problem is a single-state Markov decision process where actions can only differ in the likelihood that they produce reward.

Algorithm 15.1 defines the simulation loop for a bandit problem. At each step, we evaluate our exploration policy π on our current model of the payoff probabilities to generate an action a. The next section will discuss a way to model payoff probabilities, and the remainder of the chapter will outline several different exploration strategies. After obtaining a, we simulate a pull of that arm, returning binary reward r. The model is then updated using the observed a and r. The simulation loop is repeated to horizon h.

```
struct BanditProblem
    θ # vector of payoff probabilities
    R # reward sampler
end

function BanditProblem(θ)
    R(a) = rand() < θ[a] ? 1 : 0
    return BanditProblem(θ, R)
end

function simulate(𝒫::BanditProblem, model, π, h)
    for i in 1:h
        a = π(model)
        r = 𝒫.R(a)
        update!(model, a, r)
    end
end
```

Algorithm 15.1. Simulation of a bandit problem. A bandit problem is defined by a vector `θ` of payoff probabilities, one per action. We also define a function `R` that simulates the generation of a stochastic binary reward in response to the selection of an action. Each step of a simulation involves generating an action `a` from the exploration policy `π`. The exploration policy generally consults the model in the selection of the action. The selection of that action results in a randomly generated reward, which is then used to update the model. Simulations are run to horizon `h`.

15.2 Bayesian Model Estimation

We would like to track our belief over the win probability θ_a for arm a. The beta distribution (section 4.2) is often used for representing such a belief. Assuming a uniform prior of $\text{Beta}(1, 1)$, the posterior for θ_a after w_a wins and ℓ_a losses is $\text{Beta}(w_a + 1, \ell_a + 1)$. The posterior probability of winning is

$$\rho_a = P(\text{win}_a \mid w_a, \ell_a) = \int_0^1 \theta \times \text{Beta}(\theta \mid w_a + 1, \ell_a + 1)\, d\theta = \frac{w_a + 1}{w_a + \ell_a + 2} \tag{15.1}$$

Algorithm 15.2 provides an implementation. Example 15.1 illustrates how to compute these posterior distributions from counts of wins and losses.

```julia
struct BanditModel
    B # vector of beta distributions
end

function update!(model::BanditModel, a, r)
    α, β = StatsBase.params(model.B[a])
    model.B[a] = Beta(α + r, β + (1-r))
    return model
end
```

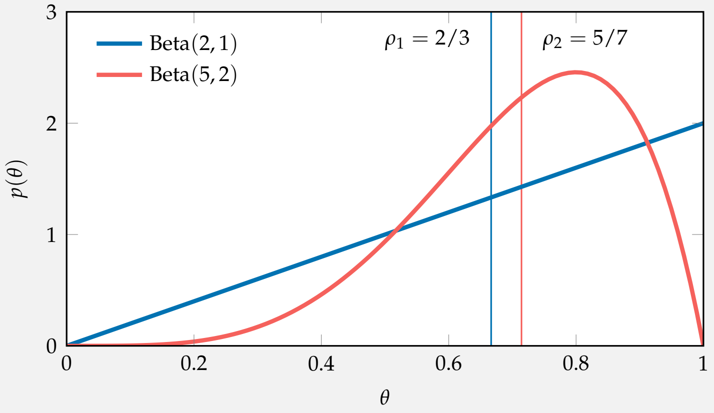
Algorithm 15.2. The Bayesian update function for bandit models. After observing reward `r` after taking action `a`, we update the beta distribution associated with that action by incrementing the appropriate parameter.

A *greedy action* is one that maximizes our expected immediate reward—or, in other words, the posterior probability of winning in the context of our binary bandit problem. There may be multiple greedy actions. We do not always want to select a greedy action because we may miss out on discovering another action that may actually provide higher reward in expectation. We can use the information from the beta distributions associated with the different actions to drive our exploration of non-greedy actions.

15.3 Undirected Exploration Strategies

There are several *ad hoc exploration* strategies that are commonly used to balance exploration with exploitation. This section discusses a type of ad hoc exploration called *undirected exploration*, where we do not use information from previous outcomes to guide exploration of non-greedy actions.

Suppose we have a two-armed bandit that we have pulled six times. The first arm has 1 win and 0 losses, and the other arm has 4 wins and 1 loss. Assuming a uniform prior, the posterior distribution for θ_1 is $\text{Beta}(2, 1)$ and the posterior distribution for θ_2 is $\text{Beta}(5, 2)$.



These posteriors assign non-zero likelihood to win probabilities between 0 and 1. The density at 0 is 0 for both arms because they both received at least one win. Similarly, the density at 1 for arm 2 is 0 because it received at least one loss. The payoff probabilities $\rho_1 = 2/3$ and $\rho_2 = 5/7$ are shown above with vertical lines. We believe that the second arm has the best chance of producing a payout.

Example 15.1. An example of posterior probability distributions and expected payouts for a multi-armed bandit.

One of the most common undirected exploration strategies is *ϵ-greedy exploration* (algorithm 15.3). This strategy chooses a random arm with probability ϵ. Otherwise, we choose a greedy arm, $\arg\max_a \rho_a$. This ρ_a is the posterior probability of a win with action a using the Bayesian model in the previous section. Alternatively, we can use the maximum likelihood estimate, but with enough pulls, the difference between the two approaches is small. Larger values of ϵ lead to more exploration, thereby resulting in faster identification of the best arm, but more pulls are wasted on suboptimal arms. Example 15.2 demonstrates this exploration strategy and the evolution of our beliefs.

The ϵ-greedy method maintains a constant amount of exploration, despite there being far more uncertainty early in the interaction with the bandit than later. One common adjustment is to decay ϵ over time, such as with an exponential decay schedule with the following update:

$$\epsilon \leftarrow \alpha\epsilon \tag{15.2}$$

for some $\alpha \in (0, 1)$ typically close to 1.

```julia
mutable struct EpsilonGreedyExploration
    ϵ # probability of random arm
    α # exploration decay factor
end

function (π::EpsilonGreedyExploration)(model::BanditModel)
    if rand() < π.ϵ
        π.ϵ *= π.α
        return rand(eachindex(model.B))
    else
        return argmax(mean.(model.B))
    end
end
```

Algorithm 15.3. The ϵ-green exploration strategy. With probability `ϵ`, it will return a random action after decaying `ϵ` by a factor `α`. Otherwise, it will return a greedy action.

Another strategy is *explore-then-commit exploration* (algorithm 15.4), where we select actions uniformly at random for the first k timesteps. From that point on, we choose a greedy action.[3] Large values for k reduce the risk of committing to a suboptimal action, but we waste more time exploring potentially suboptimal actions.

[3] A. Garivier, T. Lattimore, and E. Kaufmann, "On Explore-Then-Commit Strategies," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Example 15.2. Application of the ϵ-greedy exploration strategy to a two-armed bandit problem.

We would like to apply the ϵ-greedy exploration strategy to a two-armed bandit. We can construct the model with a uniform prior and the exploration policy with $\epsilon = 0.3$ and $\alpha = 0.99$:

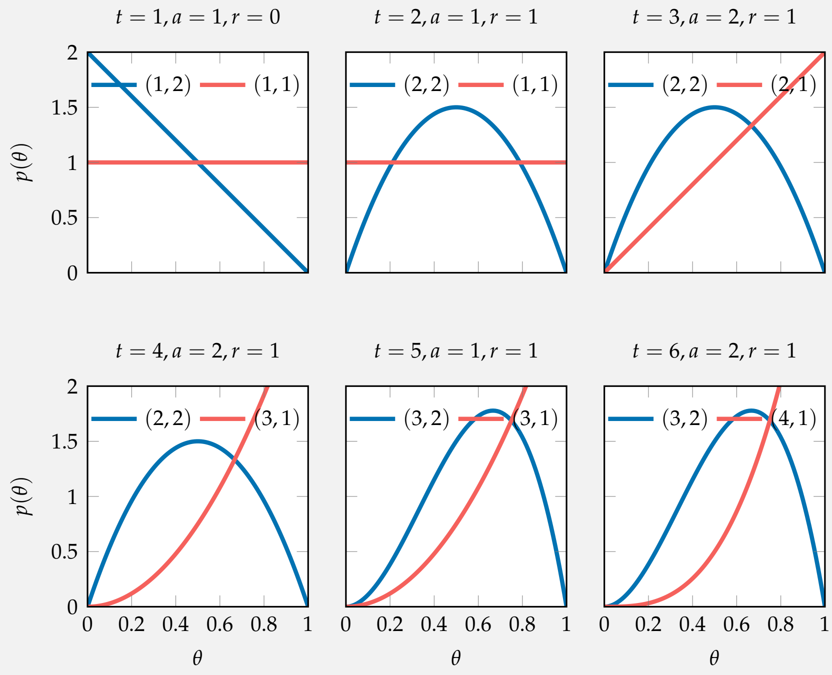
```
model(fill(Beta(),2))
π = EpsilonGreedyExploration(0.3, 0.99)
```

To obtain our first action, we call `π(model)`, which returns 1 based on the current state of the random number generator. We observe a loss, with $r = 0$, and then call

```
update!(model, 1, 0)
```

which updates the beta distributions within the model to reflect that we took action 1 and received a reward of 0.

The plots below show the evolution of the payoff beliefs after each of six steps of execution using our exploration strategy. Blue corresponds to the first action, and red corresponds to the second action.



```
mutable struct ExploreThenCommitExploration
    k # pulls remaining until commitment
end

function (π::ExploreThenCommitExploration)(model::BanditModel)
    if π.k > 0
        π.k -= 1
        return rand(eachindex(model.B))
    end
    return argmax(mean.(model.B))
end
```

Algorithm 15.4. The explore-then-commit exploration strategy. If `k` is strictly positive, it will return a random action after decrementing `k`. Otherwise, it will return a greedy action.

15.4 Directed Exploration Strategies

Directed exploration uses information gathered from previous pulls to guide exploration of the non-greedy actions. For example, the *softmax strategy* (algorithm 15.5) pulls arm a with probability proportional to $\exp(\lambda\rho_a)$, where the *precision parameter* $\lambda \geq 0$ controls the amount of exploration. We have uniform random selection as $\lambda \to 0$ and greedy selection as $\lambda \to \infty$. As more data is accumulated, we may want to increase λ by a multiplicative factor to reduce exploration.

```
mutable struct SoftmaxExploration
    λ # precision parameter
    α # precision factor
end

function (π::SoftmaxExploration)(model::BanditModel)
    weights = exp.(π.λ * mean.(model.B))
    π.λ *= π.α
    return rand(Categorical(normalize(weights, 1)))
end
```

Algorithm 15.5. The softmax exploration strategy. It selects action a with probability proportional to $\exp(\lambda\rho_a)$. The precision parameter `λ` is scaled by a factor `α` at each step.

A variety of exploration strategies are grounded in the idea of *optimism under uncertainty*. If we are optimistic about the outcomes of our actions to the extent that our data statistically allows, we will be implicitly driven to balance exploration and exploitation. One such approach is *quantile exploration* (algorithm 15.6),[4] where we choose the arm with the highest α-*quantile* (section 2.2.2) for the payoff probability. Values for $\alpha > 0.5$ result in optimism under uncertainty, incentivizing the exploration of actions that have not been tried as often. Larger values for α

[4] This general strategy is related to *upper confidence bound exploration*, *interval exploration* or *interval estimation*, referring to the upper bound of a confidence interval. L. P. Kaelbling, *Learning in Embedded Systems*. MIT Press, 1993.

result in more exploration. Example 15.3 shows quantile estimation and compares it with the other exploration strategies.

```
mutable struct QuantileExploration
    α # quantile (e.g. 0.95)
end

function (π::QuantileExploration)(model::BanditModel)
    return argmax([quantile(B, π.α) for B in model.B])
end
```

Algorithm 15.6. Quantile exploration, which returns the action with the highest α quantile.

An alternative to computing the upper confidence bound for our posterior distribution exactly is to use what is called *UCB1 exploration* (algorithm 15.7).[5] In this strategy, we select the action a that maximizes

$$\rho_a + c\sqrt{\frac{\log N}{N(a)}} \tag{15.3}$$

where $N(a)$ is the number of times we have taken action a and $N = \sum_a N(a)$.[6] The parameter $c \geq 0$ controls the amount of exploration that is encouraged through the second term. Larger values of c lead to more exploration. This strategy is often used with maximum likelihood estimates of the payoff probabilities, but we can adapt it to the Bayesian context by having $N(a)$ be the sum of the beta distribution parameters associated with a.

[5] UCB stands for upper confidence bound. This is one of many different strategies discussed by P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-Time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2–3, pp. 235–256, 2002.

[6] This equation is derived from the Chernoff-Hoeffding bound.

Another general approach to exploration is to use *posterior sampling* (algorithm 15.8), also referred to as *randomized probability matching* or *Thompson sampling*.[7] It is simple to implement and does not have any parameters to tune. The idea is to sample from the posterior distribution over the rewards associated with the different actions. The action with the largest sampled value is selected.

[7] W. R. Thompson, "On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933. For a recent tutorial, see D. Russo, B. V. Roy, A. Kazerouni, I. Osband, and Z. Wen, "A Tutorial on Thompson Sampling," *Foundations and Trends in Machine Learning*, vol. 11, no. 1, pp. 1–96, 2018.

15.5 Optimal Exploration Strategies

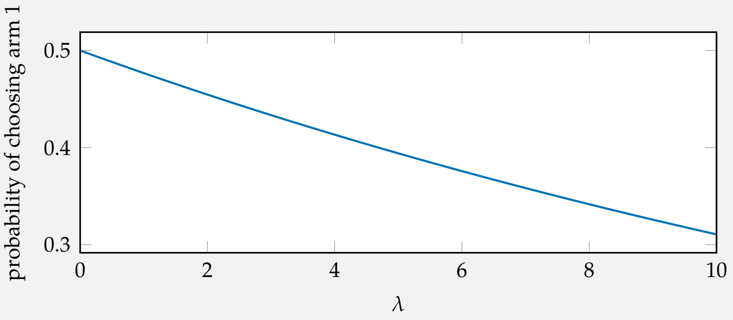
The beta distribution associated with arm a is parameterized by counts (w_a, ℓ_a). Together these counts $w_1, \ell_1, \ldots, w_n, \ell_n$ represent our belief about payoffs, and thus represent a *belief state*. These $2n$ numbers can describe n continuous probability distributions over possible payoff probabilities.

Example 15.3. Examples of exploration strategies used on the two-armed bandit problem from example 15.1.

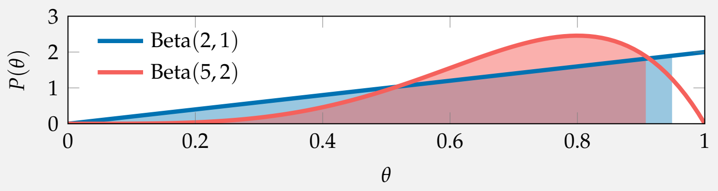
Consider using exploration strategies given the information obtained in the two-armed bandit problem of example 15.1, where the posterior distribution for θ_1 is $\text{Beta}(2, 1)$ and the posterior distribution for θ_2 is $\text{Beta}(5, 2)$. The second arm has the higher payoff probability.

An ϵ-greedy strategy with $\epsilon = 0.2$ has a 20 % chance of choosing randomly between the arms and an 80 % chance of choosing the second arm. Hence, the overall probability of choosing the first arm is 0.1 and the probability of choosing the second arm is 0.9.

A softmax strategy with $\lambda = 1$ assigns a weight of $\exp(\rho_1) = \exp(2/3) \approx 1.948$ to the first arm and a weight of $\exp(\rho_2) = \exp(5/7) \approx 2.043$ to the second. The probability of choosing the first arm is $1.948/(1.948 + 2.043) \approx 0.488$, and the probability of choosing the second arm is 0.512. The plot below shows how the probability of choosing the first arm varies with λ.



Quantile exploration with $\alpha = 0.9$ computes the payoff probability that is greater than 90 % of the probability mass associated with each posterior distribution. The 0.9 quantile for θ_1 is 0.949 and for θ_2 is 0.907, as shown below. The first arm (blue) has the higher quantile and would be pulled next.



```
mutable struct UCB1Exploration
    c # exploration constant
end

function bonus(π::UCB1Exploration, B, a)
    N = sum(b.α + b.β for b in B)
    Na = B[a].α + B[a].β
    return π.c * sqrt(log(N)/Na)
end

function (π::UCB1Exploration)(model::BanditModel)
    B = model.B
    ρ = mean.(B)
    u = ρ .+ [bonus(π, B, a) for a in eachindex(B)]
    return argmax(u)
end
```

Algorithm 15.7. The UCB1 exploration strategy with exploration constant `c`. From the pseudocount parameters in `B`, we compute equation (15.3) for each action. We then return the action that maximizes that quantity.

```
struct PosteriorSamplingExploration end

(π::PosteriorSamplingExploration)(model::BanditModel) = argmax(rand.(model.B))
```

Algorithm 15.8. The posterior sampling exploration strategy. It has no free parameters. It simply samples from the beta distributions associated with each action and then returns the action associated with the largest sample.

We can construct an MDP whose states are vectors of length $2n$ that represent the agent's belief over the n-armed bandit problem. Dynamic programming can be used to solve this MDP to obtain an optimal policy π^* that specifies which arm to pull given the counts.

Let $Q^*(w_1, \ell_1, \ldots, w_n, \ell_n, a)$ represent the expected payoff after pulling arm a and thereafter acting optimally. The optimal utility function and optimal policy can be written in terms of Q^*:

$$U^*(w_1, \ell_1, \ldots, w_n, \ell_n) = \max_a Q^*(w_1, \ell_1, \ldots, w_n, \ell_n, a) \tag{15.4}$$

$$\pi^*(w_1, \ell_1, \ldots, w_n, \ell_n) = \arg\max_a Q^*(w_1, \ell_1, \ldots, w_n, \ell_n, a) \tag{15.5}$$

We can decompose Q^* into two terms:

$$\begin{aligned} Q^*(w_1, \ell_1, \ldots, w_n, \ell_n, a) = & \frac{w_a + 1}{w_a + \ell_a + 2}(1 + U^*(\ldots, w_a + 1, \ell_a, \ldots)) \\ & + \left(1 - \frac{w_a + 1}{w_a + \ell_a + 2}\right) U^*(\ldots, w_a, \ell_a + 1, \ldots) \end{aligned} \tag{15.6}$$

The first term is associated with a win for arm a, and the second term is associated with a loss. The value $(w_a + 1)/(w_a + \ell_a + 2)$ is the posterior probability of a win, which comes from equation (15.1).[8] The first U^* in the equation above records a win, whereas the second U^* records a loss.

[8] This probability can be adjusted if we have a nonuniform prior.

We can compute Q^* for the entire belief space, as we have assumed a finite horizon h. We start with all terminal belief states with $\sum_a(w_a + \ell_a) = h$, where $U^* = 0$. We can then work backward to states with $\sum_a(w_a + \ell_a) = h - 1$ and apply equation (15.6). This process is repeated until we reach our initial state. Such an optimal policy is computed in example 15.4.

Although this dynamic programming solution is optimal, the number of belief states is $O(h^{2n})$. We can formulate an infinite horizon, discounted version of the problem that can be solved efficiently using the *Gittins allocation index*:[9]

$$g(w_{1:n}, \ell_{1:n}) = \frac{1}{1-\gamma}\,\mathbb{E}\left[\sum_{k=1}^{\infty} \gamma^{(k-1)} r^{(k)}\right] \tag{15.7}$$

[9] J. C. Gittins, "Bandit Processes and Dynamic Allocation Indices," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 41, no. 2, pp. 148–177, 1979. J. Gittins, K. Glazebrook, and R. Weber, *Multi-Armed Bandit Allocation Indices*, 2nd ed. Wiley, 2011.

where $r^{(k)}$ is the payout obtained at the kth step and γ is the discount factor.

The allocation index can be stored as a lookup table that specifies a scalar allocation index value given the number of pulls and the number of wins associated with an arm.[10] The arm that has the highest allocation index is the one that should be pulled next.

[10] A survey of algorithms for computing this lookup table are provided in J. Chakravorty and A. Mahajan, "Multi-Armed Bandits, Gittins Index, and Its Calculation," in *Methods and Applications of Statistics in Clinical Trials*, vol. 2, Wiley, 2014, pp. 416–435.

15.6 Exploration with Multiple States

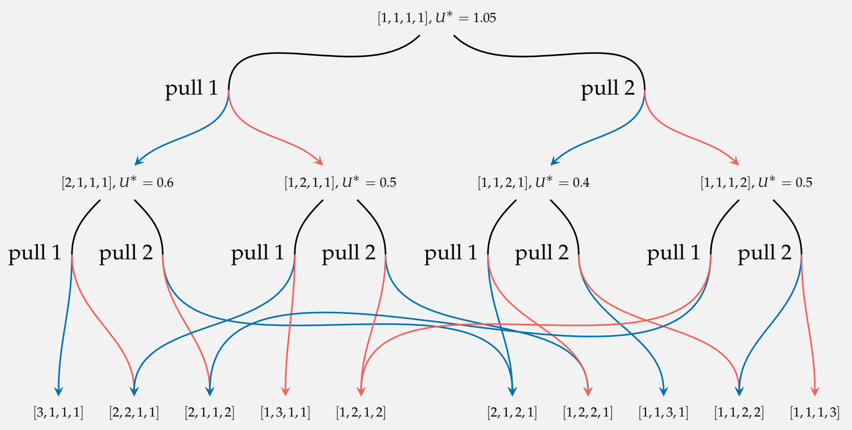
In the general reinforcement learning context with multiple states, we must use observations about state transitions to inform our decisions. We can modify the simulation process in algorithm 15.1 to account for state transitions and update our model appropriately. Algorithm 15.9 provides an implementation. There are many ways to model the problem and perform exploration as we will discuss over the next few chapters, but the simulation structure is exactly the same.

15.7 Summary

- The exploration-exploitation tradeoff is a balance between exploring the state-action space for higher rewards and exploiting the already known favorable state-actions.
- Multi-armed bandit problems involve a single state where the agent receives stochastic rewards for taking different actions.

Example 15.4. Computing the optimal policy for a two-armed, two-step horizon bandit problem.

Below we have constructed the state-action tree for a two-arm bandit problem with a two-step horizon. State vectors are shown as $[w_1, \ell_1, w_2, \ell_2]$, blue arrows indicate wins, and red arrows indicate losses.



Unsurprisingly, the policy is symmetric with respect to arms 1 and 2. We find that the first arm does not matter, and it is best to pull a winning arm twice and not pull a losing arm twice.

The optimal value functions were computed using:

$$Q^*([2,1,1,1],1) = \frac{3}{5}(1+0) + \frac{2}{5}(0) = 0.6$$
$$Q^*([2,1,1,1],2) = \frac{2}{4}(1+0) + \frac{2}{4}(0) = 0.5$$
$$Q^*([1,2,1,1],1) = \frac{2}{5}(1+0) + \frac{3}{5}(0) = 0.4$$
$$Q^*([1,2,1,1],2) = \frac{2}{4}(1+0) + \frac{2}{4}(0) = 0.5$$
$$Q^*([1,1,1,1],1) = \frac{2}{4}(1+0.6) + \frac{2}{4}(0.5) = 1.05$$

```
function simulate(𝒫::MDP, model, π, h, s)
    for i in 1:h
        a = π(model, s)
        s′, r = 𝒫.TR(s, a)
        update!(model, s, a, r, s′)
        s = s′
    end
end
```

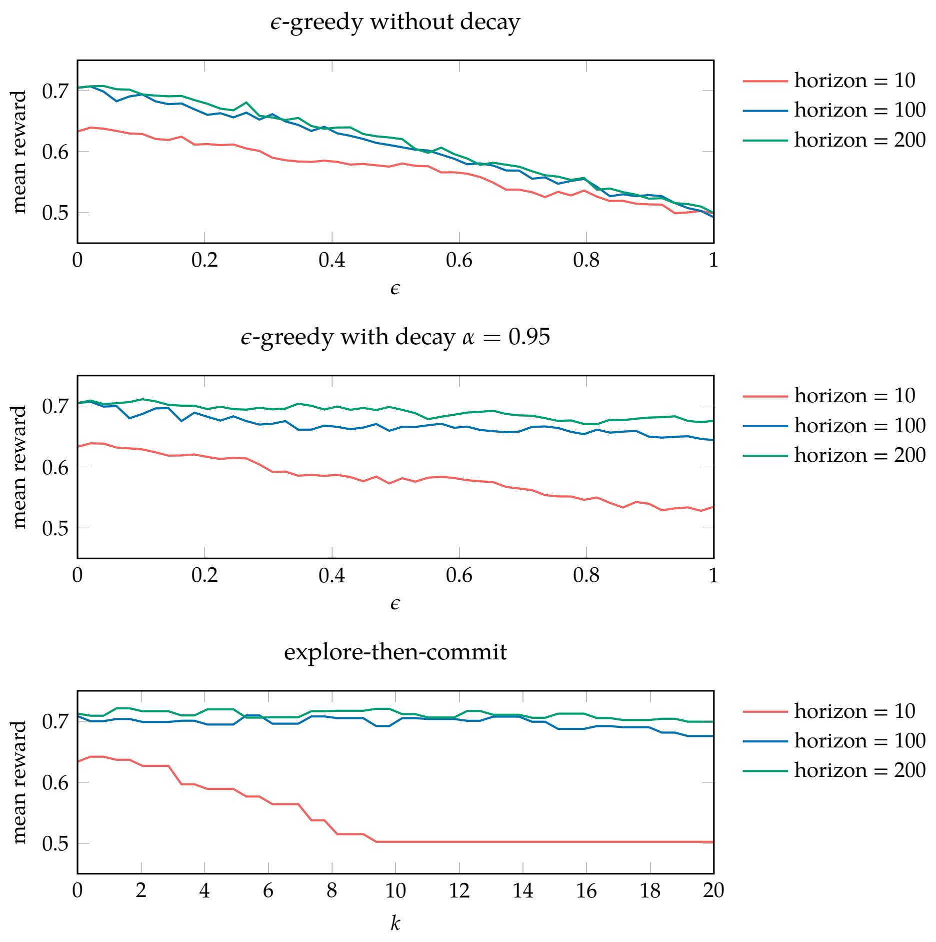
Algorithm 15.9. The simulation loop for reinforcement learning problems. The exploration policy π generates the next action based on information in the model and the current state s. The MDP problem 𝒫 is treated as the ground truth and is used to sample the next state and reward. The state transition and reward is used to update the model. The simulation is run to horizon h.

- A beta distribution can be used to maintain a belief over multi-armed bandit rewards.
- Undirected exploration strategies, including ϵ-greedy and explore-then-commit, are simple to implement but do not use information from previous outcomes to guide exploration of non-greedy actions.
- Directed exploration strategies, including softmax, quantile, UCB1, and posterior sampling exploration, use information from past actions to better explore promising actions.
- Dynamic programming can be used to derive optimal exploration strategies for finite horizons, but these strategies can be expensive to compute.

15.8 Exercises

Exercise 15.1. Consider 3-armed bandit problems where each arm has a win probability drawn uniformly between 0 and 1. Compare the ϵ-greedy, ϵ-greedy with α decay, and explore-then-commit exploration strategies. Qualitatively, what values for ϵ, α, and k produce the highest expected reward on randomly generated bandit problems?

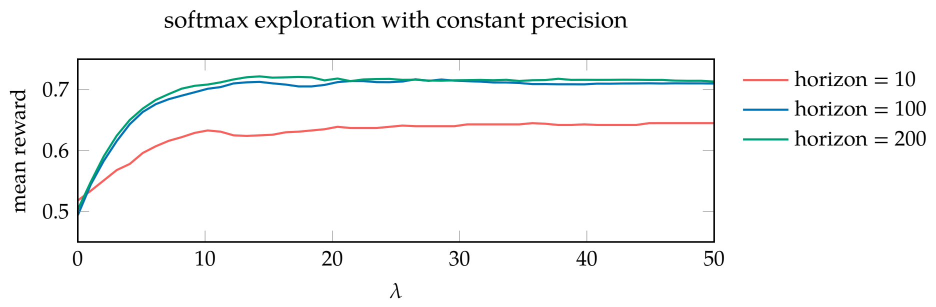
Solution: Below we plot the expected reward-per-step for the three different strategies. The effectiveness of the parameterization depends on the problem horizon, so several different depths are shown as well.

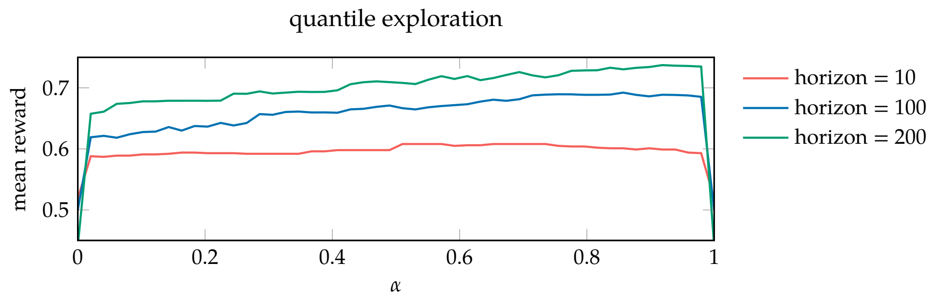


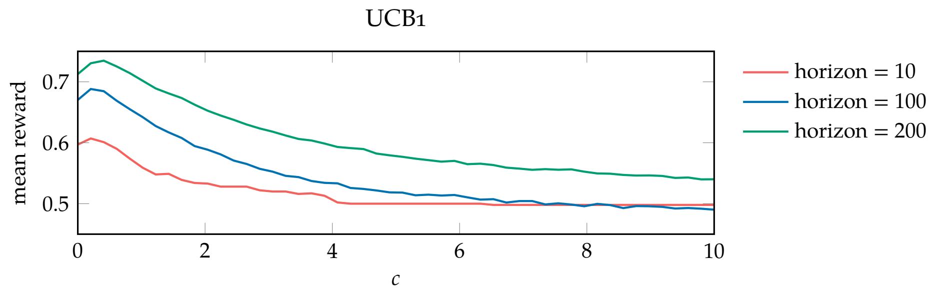
Purely greedy policies ($\epsilon = 0$) outperform other ϵ-greedy strategies for shorter horizons. Having a small non-zero chance of choosing randomly is most valuable for very long horizons. Including a moderate α decay improves performance, especially for large values of ϵ on problems with longer horizons. These policies had the highest overall expected reward per iteration. The explore-then-commit policy behaves in a manner similar to ϵ-greedy with α decay. It performs best with k at roughly 20 % of the horizon.

Exercise 15.2. Consider again the 3-armed bandit problems where each arm has a win probability drawn uniformly between 0 and 1. Compare the softmax, quantile, and UCB1 exploration strategies. Qualitatively, what values for λ, α, and c produce the highest expected reward on randomly generated bandit problems?

Solution: Below we plot the expected reward-per-step for the three different strategies. Again, the effectiveness of the parameterization depends on the problem horizon, so several different depths are shown as well.







The softmax strategy performs best for large values of λ, which prioritize pulling arms with higher expected reward according to the current belief. Upper confidence bound exploration performs better with longer horizons, independently of its parameterization. The size of the confidence bound α does not significantly affect performance except for values very close to 0 or 1. The UCB1 strategy performs best with small positive values of the exploration scalar c. The expected reward decays as c increases. All three policies can be tuned to produce similar maximal expected rewards.

Exercise 15.3. Give an example of a multi-armed bandit problem.

Solution: There are many different multi-armed bandit problems. Consider, for example, a news company that would like to maximize interaction (clicks) on articles on its website. The company may have several articles to display, but must select on article to display at a given time. This problem is a multi-armed bandit problem because a user will either click article i with probability θ_i or not click with probability $1 - \theta_i$. Exploration would consist of displaying different articles on the website and observing the number of clicks and exploitation would consist of displaying the article likely to lead to the highest number of clicks. This problem is related to *A/B testing*, where companies test different versions of a website to determine which version yields the most interaction.

Exercise 15.4. Given a single-armed bandit with a prior of $\theta \sim \text{Beta}(7, 2)$, provide bounds on the posterior probability of winning after 10 additional pulls.

Solution: A lower bound on our posterior probability of winning $\underline{\rho}$ can be computed assuming all pulls result in a loss, e.g. $\underline{\ell} = 10$ and $\underline{w} = 0$. We can similarly compute an upper bound $\overline{\rho}$ assuming all pulls result in a win, e.g. $\overline{w} = 10$ and $\overline{\ell} = 0$. Thus, the bounds are

$$\underline{\rho} = \frac{\underline{w} + 7}{\underline{w} + \underline{\ell} + 9} = \frac{0 + 7}{0 + 10 + 9} = \frac{7}{19}$$
$$\overline{\rho} = \frac{\overline{w} + 7}{\overline{w} + \overline{\ell} + 9} = \frac{10 + 7}{10 + 0 + 9} = \frac{17}{19}$$

Exercise 15.5. We have a bandit with arms a and b, and we use an ϵ-greedy exploration strategy with $\epsilon = 0.3$ and an exploration decay factor of $\alpha = 0.9$. We generate a random number x between 0 and 1 to determine if we explore $(x < \epsilon)$ or exploit $(x > \epsilon)$. Given we have $\rho_a > \rho_b$, which arm is selected if $x = 0.2914$ in the first iteration? Which arm is selected if $x = 0.1773$ in the ninth iteration?

Solution: Since $x < \epsilon_1$ in the first iteration, we explore and choose a with probability 0.5 and b with probability 0.5. At the ninth iteration, $\epsilon_9 = \alpha^8 \epsilon_1 \approx 0.129$. Since $x > \epsilon_9$, we exploit and select a.

Exercise 15.6. We have a four-armed bandit and we want to use a softmax exploration strategy with precision parameter $\lambda = 2$ and a prior belief $\theta_a \sim \text{Beta}(2, 2)$ for each arm a. Suppose we pull each arm four times with the result that arms 1, 2, 3, and 4 pay off 1, 2, 3 and 4 times, respectively. List the posterior distributions over θ_a and calculate the probability that we select arm 2.

Solution: The posterior distribution for each arm is respectively: Beta(3, 5), Beta(4, 4), Beta(5, 3), Beta(6, 2). The probability of selecting arm 2 can be computed in the following steps

$$\begin{aligned}
P(a = i) &\propto \exp(\lambda \rho_i) \\
P(a = i) &= \frac{\exp(\lambda \rho_i)}{\sum_a \exp(\lambda \rho_a)} \\
P(a = 2) &= \frac{\exp(2 \times \frac{4}{8})}{\exp(2 \times \frac{3}{8}) + \exp(2 \times \frac{4}{8}) + \exp(2 \times \frac{5}{8}) + \exp(2 \times \frac{6}{8})} \\
P(a = 2) &\approx 0.2122
\end{aligned}$$

Exercise 15.7. Rewrite equation (15.6) for an arbitrary Beta(α, β) prior.

Solution: We can rewrite the equation more generally as follows

$$\begin{aligned}
Q^*(w_1, \ell_1, \ldots, w_n, \ell_n, a) = &\frac{w_a + \alpha}{w_a + \ell_a + \alpha + \beta}(1 + U^*(\ldots, w_a + 1, \ell_a, \ldots)) \\
&+ \left(1 - \frac{w_a + \alpha}{w_a + \ell_a + \alpha + \beta}\right) U^*(\ldots, w_a, \ell_a + 1, \ldots)
\end{aligned}$$

Exercise 15.8. Recall example 15.4. Instead of having a payoff of 1 for each arm, let us assume arm 1 gives a payoff of 1 while arm 2 gives a payoff of 2. Calculate the new action value functions for both arms.

Solution: For arm 1, we have:

$$\begin{aligned}
Q^*([2, 1, 1, 1], 1) &= \frac{3}{5}(1 + 0) + \frac{2}{5}(0) = 0.6 \\
Q^*([2, 1, 1, 1], 2) &= \frac{2}{4}(2 + 0) + \frac{2}{4}(0) = 1 \\
Q^*([1, 2, 1, 1], 1) &= \frac{2}{5}(1 + 0) + \frac{3}{5}(0) = 0.4 \\
Q^*([1, 2, 1, 1], 2) &= \frac{2}{4}(2 + 0) + \frac{2}{4}(0) = 1 \\
Q^*([1, 1, 1, 1], 1) &= \frac{2}{4}(1 + 1) + \frac{2}{4}(1) = 1.5
\end{aligned}$$

And for arm 2, we have:

$$\begin{aligned}
Q^*([1,1,2,1],1) &= \frac{2}{4}(1+0) + \frac{2}{4}(0) = 0.5 \\
Q^*([1,1,2,1],2) &= \frac{3}{5}(2+0) + \frac{2}{5}(0) = 1.2 \\
Q^*([1,1,1,2],1) &= \frac{2}{4}(1+0) + \frac{2}{4}(0) = 0.5 \\
Q^*([1,1,1,2],2) &= \frac{2}{5}(2+0) + \frac{3}{5}(0) = 0.8 \\
Q^*([1,1,1,1],2) &= \frac{2}{4}(2+1.2) + \frac{2}{4}(0.8) = 2
\end{aligned}$$

Exercise 15.9. Prove that the number of belief states in an n-armed bandit problem with a horizon of h is $O(h^{2n})$.

Solution: We begin by counting the number of solutions to $w_1 + \ell_1 + \cdots + w_n + \ell_n = k$, where $0 \leq k \leq h$. If $n = 2$ and $k = 6$, one solution is $2 + 0 + 3 + 1 = 6$. For our counting argument, we will use tally marks to represent our integers. For example, we can write a solution like $2 + 0 + 3 + 1 =$ **||++|||+|** $= 6$. For general values for n and k, we would have k tally marks and $2n - 1$ plus signs. Given that many tally marks and plus signs, we can arrange them in any order we want. We can represent a solution as a string of $k + 2n - 1$ characters, where a character is either **|** or **+**, with k of those characters being **|**. To obtain the number of solutions, we count the number of ways we can choose k positions for **|** from the set of $k + 2n - 1$ positions, resulting in

$$\frac{(k + 2n - 1)!}{(2n - 1)!k!} = O(h^{2n-1})$$

solutions. The number of belief states is this expression summed for k from 0 to h, which is $O(h \times h^{2n-1}) = O(h^{2n})$.