16 Model-Based Methods

This chapter discusses both maximum likelihood and Bayesian approaches for learning the underlying dynamics and reward through interaction with the environment. Maximum likelihood methods involve counting state transitions and recording the amount of reward received to estimate the model parameters. We will discuss a few approaches for planning using models that are continuously updated. Even if we solve the estimated problem exactly, we generally have to rely on heuristic exploration strategies to arrive at a suitable solution. Bayesian methods involve computing a posterior distribution over model parameters. Solving for the optimal exploration strategy is generally intractable, but we can often obtain a sensible approximation through posterior sampling.

16.1 Maximum Likelihood Models

As introduced in section 15.6 and implemented in algorithm 15.9, reinforcement learning involves using information about past state transitions and rewards to inform decisions. This section describes how to obtain a *maximum likelihood estimate* of the underlying problem. This maximum likelihood estimate can be used to generate a value function estimate that can be used with an exploration strategy to generate actions.

We record the transition counts $N(s, a, s')$, indicating the number of times a transition from s to s' was observed when taking action a. The maximum likelihood estimate of the transition function given these transition counts is

$$T(s' \mid s, a) \approx N(s, a, s') / N(s, a) \tag{16.1}$$

where $N(s, a) = \sum_{s'} N(s, a, s')$. If $N(s, a) = 0$, then we have no information from which to estimate a transition. In such a case, we can default to a uniform

distribution or assume that the transition is impossible by setting the transition probabilities to zero.

The reward function can also be estimated. As we receive rewards, we update $\rho(s, a)$, the sum of all rewards obtained when taking action a in state s. The maximum likelihood estimate of the reward function is the mean reward:

$$R(s, a) \approx \rho(s, a) / N(s, a) \tag{16.2}$$

If $N(s, a) = 0$, then our estimate of $R(s, a)$ is 0. If we have prior knowledge about the transition probabilities or rewards, then we can initialize $N(s, a, s')$ and $\rho(s, a)$ to values other than 0.

Algorithm 16.1 updates N and ρ after observing the transition from s to s' after taking action a and receiving reward r. Algorithm 16.2 converts the maximum likelihood model into an MDP representation. Example 16.1 illustrates this process. We can use this maximum likelihood model to select actions while interacting with the environment and improving the model.

16.2 Update Schemes

As we update our maximum likelihood estimate of the model, we also need to update our plan. This section discusses several update schemes in response to our continuously changing model. A major consideration is computational efficiency because we will want to perform these updates fairly frequently while interacting with the environment.

16.2.1 Full Updates

Algorithm 16.3 solves the maximum likelihood model using the linear programming formulation from section 7.7, though we could have used value iteration or some other algorithm. After each step, we obtain a new model estimate and re-solve.

16.2.2 Randomized Updates

Recomputing an optimal policy with each state transition is typically computationally expensive. An alternative is to perform a Bellman update on the estimated model at the previously visited state, as well as a few randomly chosen states.[1] Algorithm 16.4 implements this approach.

[1] This approach is related to the *Dyna* approach suggested by R. S. Sutton, "Dyna, an Integrated Architecture for Learning, Planning, and Reacting," *SIGART Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

```
mutable struct MaximumLikelihoodMDP
    S # state space (assumes 1:nstates)
    A # action space (assumes 1:nactions)
    N # transition count N(s,a,s′)
    ρ # reward sum ρ(s, a)
    γ # discount
    U # value function
    planner
end

function lookahead(model::MaximumLikelihoodMDP, s, a)
    S, U, γ = model.S, model.U, model.γ
    n = sum(model.N[s,a,:])
    if n == 0
        return 0.0
    end
    r = model.ρ[s, a] / n
    T(s,a,s′) = model.N[s,a,s′] / n
    return r + γ * sum(T(s,a,s′)*U[s′] for s′ in S)
end

function backup(model::MaximumLikelihoodMDP, U, s)
    return maximum(lookahead(model, s, a) for a in model.A)
end

function update!(model::MaximumLikelihoodMDP, s, a, r, s′)
    model.N[s,a,s′] += 1
    model.ρ[s,a] += r
    update!(model.planner, model, s, a, r, s′)
    return model
end
```

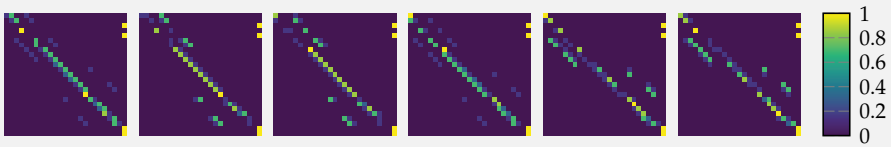
Algorithm 16.1. A method for updating the transition and reward model for maximum likelihood reinforcement learning with discrete state and action spaces. We increment `N[s,a,s′]` after observing a transition from `s` to `s′` after taking action `a`, and we add `r` to `ρ[s,a]`. The model also contains an estimate of the value function `U` and a planner. This algorithm block also includes methods for performing backup and lookahead with respect to this model.

```
function MDP(model::MaximumLikelihoodMDP)
    N, ρ, S, A, γ = model.N, model.ρ, model.S, model.A, model.γ
    T, R = similar(N), similar(ρ)
    for s in S
        for a in A
            n = sum(N[s,a,:])
            if n == 0
                T[s,a,:] .= 0.0
                R[s,a] = 0.0
            else
                T[s,a,:] = N[s,a,:] / n
                R[s,a] = ρ[s,a] / n
            end
        end
    end
    return MDP(T, R, γ)
end
```

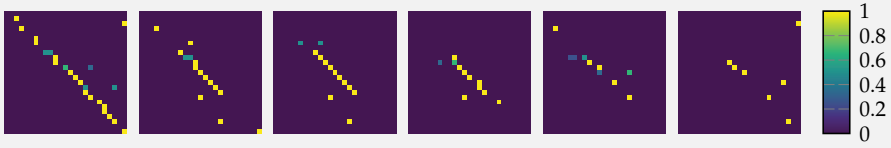
Algorithm 16.2. A method for converting a maximum likelihood model to an `MDP` problem.

Example 16.1. Applying maximum likelihood estimation to the hex world problem.

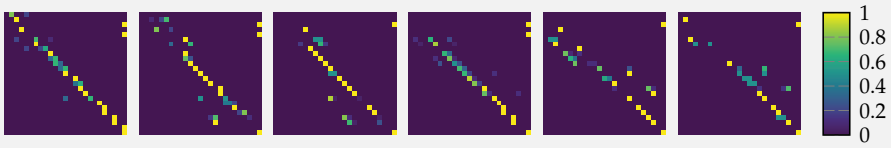
We would like to apply maximum likelihood model estimation to the hex world problem. The true transition matrices look like this:



There are six transition matrices, one for each action. The rows correspond to the current state, and the columns correspond to the next state. There are 26 states. The intensity in the images relate to the probability of making the corresponding transition. In a reinforcement learning context, we do not know these transition probabilities ahead of time. However, we can interact with the environment and record the transitions we observe. After 10 simulations of 10 steps each from random initial states, maximum likelihood estimation results in the following matrices:



After 1000 simulations, our estimate becomes



```
struct FullUpdate end

function update!(planner::FullUpdate, model, s, a, r, s′)
    𝒫 = MDP(model)
    U = solve(𝒫).U
    copy!(model.U, U)
    return planner
end
```

Algorithm 16.3. A method that performs a full update of the value function of `U` using the linear programming formulation from section 7.7.

```
struct RandomizedUpdate
    m # number of updates
end

function update!(planner::RandomizedUpdate, model, s, a, r, s′)
    U = model.U
    U[s] = backup(model, U, s)
    for i in 1:planner.m
        s = rand(model.𝒮)
        U[s] = backup(model, U, s)
    end
    return planner
end
```

Algorithm 16.4. Maximum likelihood model-based reinforcement learning with updates at randomized states. This approach performs a Bellman update at the previously visited state, as well as at `m` additional states chosen randomly.

16.2.3 *Prioritized Updates*

An approach called *prioritized sweeping*[2] (algorithm 16.5) uses a priority queue to help identify which states are most in need of updating. A transition from s to s' is followed by an update of $U(s)$ based on our updated transition and reward models. We then iterate over all state-action pairs (s^-, a^-) that can immediately transition into s. The priority of any such s^- is increased to $T(s \mid s^-, a^-) \times |U(s) - u|$, where u was the value of $U(s)$ before the update. Hence, the larger the change in $U(s)$ and the more likely the transition to s, the higher the priority of states leading to s. The process of updating the highest-priority state in the queue continues for a fixed number of iterations or until the queue becomes empty.

[2] A. W. Moore and C. G. Atkeson, "Prioritized Sweeping: Reinforcement Learning with Less Data and Less Time," *Machine Learning*, vol. 13, no. 1, pp. 103–130, 1993.

16.3 *Exploration*

Regardless of the update scheme, some form of exploration strategy generally must be followed to avoid the pitfalls of pure exploitation mentioned in the previous chapter. We can adapt the exploration algorithms presented in that chapter

```
struct PrioritizedUpdate
    m  # number of updates
    pq # priority queue
end

function update!(planner::PrioritizedUpdate, model, s)
    N, U, pq = model.N, model.U, planner.pq
    𝒮, 𝒜 = model.𝒮, model.𝒜
    u = U[s]
    U[s] = backup(model, U, s)
    for s⁻ in 𝒮
        for a⁻ in 𝒜
            n_sa = sum(N[s⁻,a⁻,s′] for s′ in 𝒮)
            if n_sa > 0
                T = N[s⁻,a⁻,s] / n_sa
                priority = T * abs(U[s] - u)
                if priority > 0
                    pq[s⁻] = max(get(pq, s⁻, 0.0), priority)
                end
            end
        end
    end
    return planner
end

function update!(planner::PrioritizedUpdate, model, s, a, r, s′)
    planner.pq[s] = Inf
    for i in 1:planner.m
        if isempty(planner.pq)
            break
        end
        update!(planner, model, dequeue!(planner.pq))
    end
    return planner
end
```

Algorithm 16.5. The prioritized sweeping algorithm maintains a priority queue `pq` of states that determines which are to be updated. With each update, we set the previous state to have infinite priority. We then perform `m` Bellman updates of the value function `U` at the highest-priority states.

for use in multistate problems. Algorithm 16.6 provides an implementation of the ϵ-greedy exploration strategy.

```
function (π::EpsilonGreedyExploration)(model, s)
    𝒜, ϵ = model.𝒜, π.ϵ
    if rand() < ϵ
        return rand(𝒜)
    end
    Q(s,a) = lookahead(model, s, a)
    return argmax(a→Q(s,a), 𝒜)
end
```

Algorithm 16.6. The ϵ-greedy exploration strategy for maximum likelihood model estimates. It chooses a random action with probability ϵ; otherwise, it uses the model to extract the greedy action.

A limitation of the exploration strategies discussed in the previous chapter is that they do not reason about exploring actions from states besides the current one. For instance, we might want to take actions that bring ourselves into an area of the state space that has not been explored. Several algorithms have been suggested for addressing this issue, which also provide probabilistic bounds on the quality of the resulting policy after a finite number of interactions.[3]

One such algorithm is known as *R-MAX* (algorithm 16.7).[4] Its name comes from assigning maximal reward to underexplored state-action pairs. State-action pairs with fewer than m visitations are considered underexplored. Instead of using the maximum likelihood estimate for the reward (equation (16.2)), we use

$$R(s,a) = \begin{cases} r_{\max} & \text{if } N(s,a) < m \\ \rho(s,a)/N(s,a) & \text{otherwise} \end{cases} \tag{16.3}$$

where $r_{\max}$ is the maximum achievable reward.

The transition model in R-MAX is also modified so that underexplored state-action pairs result in staying in the same state:

$$T(s' \mid s,a) = \begin{cases} (s' = s) & \text{if } N(s,a) < m \\ N(s,a,s')/N(s,a) & \text{otherwise} \end{cases} \tag{16.4}$$

Hence, underexplored states have value $r_{\max}/(1-\gamma)$, providing an incentive to explore them. This exploration incentive relieves us of needing a separate exploration mechanism. We simply choose our actions greedily with respect to the value function derived from our transition and reward estimates. Example 16.2 demonstrates ϵ-greedy and R-MAX exploration.

[3] M. Kearns and S. Singh, "Near-Optimal Reinforcement Learning in Polynomial Time," *Machine Learning*, vol. 49, no. 2/3, pp. 209–232, 2002.

[4] R. I. Brafman and M. Tennenholtz, "R-MAX—A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning," *Journal of Machine Learning Research*, vol. 3, pp. 213–231, 2002.

```
mutable struct RmaxMDP
    𝒮 # state space (assumes 1:nstates)
    𝒜 # action space (assumes 1:nactions)
    N # transition count N(s,a,s′)
    ρ # reward sum ρ(s, a)
    γ # discount
    U # value function
    planner
    m    # count threshold
    rmax # maximum reward
end

function lookahead(model::RmaxMDP, s, a)
    𝒮, U, γ = model.𝒮, model.U, model.γ
    n = sum(model.N[s,a,:])
    if n < model.m
        return model.rmax / (1-γ)
    end
    r = model.ρ[s, a] / n
    T(s,a,s′) = model.N[s,a,s′] / n
    return r + γ * sum(T(s,a,s′)*U[s′] for s′ in 𝒮)
end

function backup(model::RmaxMDP, U, s)
    return maximum(lookahead(model, s, a) for a in model.𝒜)
end

function update!(model::RmaxMDP, s, a, r, s′)
    model.N[s,a,s′] += 1
    model.ρ[s,a] += r
    update!(model.planner, model, s, a, r, s′)
    return model
end

function MDP(model::RmaxMDP)
    N, ρ, 𝒮, 𝒜, γ = model.N, model.ρ, model.𝒮, model.𝒜, model.γ
    T, R, m, rmax = similar(N), similar(ρ), model.m, model.rmax
    for s in 𝒮
        for a in 𝒜
            n = sum(N[s,a,:])
            if n < m
                T[s,a,:] .= 0.0
                T[s,a,s] = 1.0
                R[s,a] = rmax
            else
                T[s,a,:] = N[s,a,:] / n
                R[s,a] = ρ[s,a] / n
            end
        end
    end
    return MDP(T, R, γ)
end
```

Algorithm 16.7. The R-MAX exploration strategy modifies the transition and reward model from maximum likelihood estimation. It assigns the maximum reward `rmax` to any underexplored state-action pair, defined as being those that have been tried fewer than `m` times. In addition, all underexplored state-action pairs are modeled as transitioning to the same state. This `RmaxMDP` can be used as a replacement for the `MaximumLikelihoodMDP` introduced in algorithm 16.1.

Example 16.2. Demonstration of ϵ-greedy and R-MAX exploration.

We can apply ϵ-greedy exploration to maximum likelihood model estimates constructed while interacting with the environment. The code that follows initializes the counts, rewards, and utilities to zero. It uses full updates to the value function with each step. For exploration, we choose a random action with probability 0.1. The last line runs a simulation (algorithm 15.9) of problem $\mathcal{P}$ for 100 steps starting in a random initial state:

```
N = zeros(length(𝒮), length(𝒜), length(𝒮))
ρ = zeros(length(𝒮), length(𝒜))
U = zeros(length(𝒮))
planner = FullUpdate()
model = MaximumLikelihoodMDP(𝒮, 𝒜, N, ρ, γ, U, planner)
π = EpsilonGreedyExploration(0.1)
simulate(𝒫, model, π, 100, rand(𝒮))
```

Alternatively, we can use R-MAX with an exploration threshold of $m = 3$. We can act greedily with respect to the R-MAX model:

```
rmax = maximum(𝒫.R(s,a) for s in 𝒮, a in 𝒜)
m = 3
model = RmaxMDP(𝒮, 𝒜, N, ρ, γ, U, planner, m, rmax)
π = EpsilonGreedyExploration(0)
simulate(𝒫, model, π, 100, rand(𝒮))
```

16.4 Bayesian Methods

In contrast with the maximum likelihood methods discussed so far, Bayesian methods balance exploration and exploitation without having to rely on heuristic exploration policies. This section describes a generalization of the Bayesian methods covered in section 15.5. In *Bayesian reinforcement learning*, we specify a prior distribution over all model parameters θ.[5] These model parameters may include the parameters governing the distribution over immediate rewards, but this section focuses on the parameters governing the state transition probabilities.

[5] A survey of this topic is provided by M. Ghavamzadeh, S. Mannor, J. Pineau, and A. Tamar, "Bayesian Reinforcement Learning: A Survey," *Foundations and Trends in Machine Learning*, vol. 8, no. 5–6, pp. 359–483, 2015. It covers methods for incorporating priors over reward functions, which are not discussed here.

The structure of the problem can be represented using the dynamic decision network shown in figure 16.1, wherein the model parameters are made explicit. The shaded nodes indicate that the states are observed but the model parameters are not. We generally assume that the model parameters are time invariant with $\theta_{t+1} = \theta_t$. However, our belief about θ evolves with time as we transition to new states.

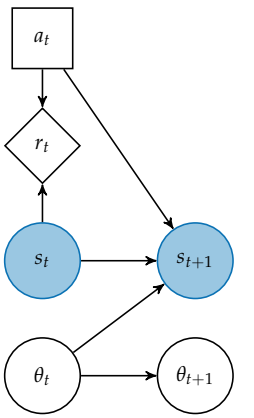


Figure 16.1. A dynamic decision network for an MDP with model uncertainty.

The belief over transition probabilities can be represented using a collection of Dirichlet distributions, one for each source state and action. Each Dirichlet distribution represents the distribution over s' for a given s and a. If $\boldsymbol{\theta}_{(s,a)}$ is an $|\mathcal{S}|$-element vector representing the distribution over the next state, then the prior distribution is given by

$$\text{Dir}(\boldsymbol{\theta}_{(s,a)} \mid \mathbf{N}(s,a)) \tag{16.5}$$

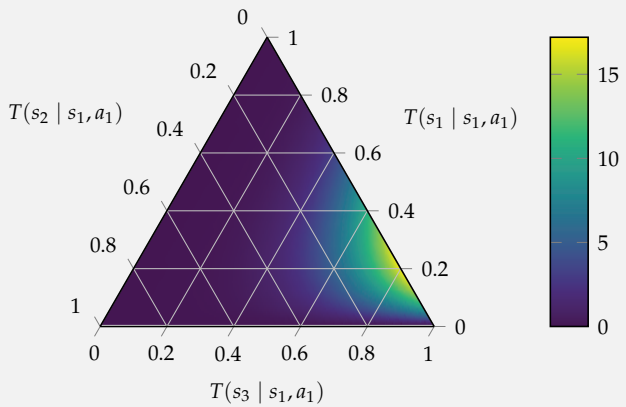
where $\mathbf{N}(s,a)$ is the vector of counts associated with transitions starting in state s taking action a. It is common to use a uniform prior with all components set to 1, but prior knowledge of the transition dynamics can be used to initialize the counts differently. Example 16.3 illustrates how these counts are used by the Dirichlet distribution to represent the distribution over possible transition probabilities.

The distribution over θ is the result of the product of the Dirichlet distributions:

$$b(\theta) = \prod_s \prod_a \text{Dir}\left(\boldsymbol{\theta}_{(s,a)} \mid \mathbf{N}(s,a)\right) \tag{16.6}$$

Algorithm 16.8 provides an implementation of the Bayesian update for this type of posterior model. For problems with larger or continuous spaces, we can use other posterior representations.

Suppose our agent randomly explores an environment with three states. The agent takes action a_1 from state s_1 five times. It transitions to s_3 four times and remains in s_1 once. The counts associated with s_1 and a_1 are $\mathbf{N}(s_1, a_1) = [1, 0, 4]$. If we want to assume a uniform prior over resulting states, we would increment the counts by 1 to get $\mathbf{N}(s_1, a_1) = [2, 1, 5]$. The transition function from s_1 taking action a_1 is a three-valued categorical distribution because there are three possible successor states. Each successor state has an unknown transition probability. The space of possible transition probabilities is the set of three-element vectors that sum to 1. The Dirichlet distribution represents a probability distribution over these possible transition probabilities. Here is a plot of the density function:



Example 16.3. A posterior Dirichlet distribution over transition probabilities from a particular state when taking a particular action. An agent learning the transition function in an unknown MDP may choose to maintain such a distribution over each state-action pair.

```
mutable struct BayesianMDP
    𝒮 # state space (assumes 1:nstates)
    𝒜 # action space (assumes 1:nactions)
    D # Dirichlet distributions D[s,a]
    R # reward function as matrix (not estimated)
    γ # discount
    U # value function
    planner
end

function lookahead(model::BayesianMDP, s, a)
    𝒮, U, γ = model.𝒮, model.U, model.γ
    n = sum(model.D[s,a].alpha)
    if n == 0
        return 0.0
    end
    r = model.R(s,a)
    T(s,a,s′) = model.D[s,a].alpha[s′] / n
    return r + γ * sum(T(s,a,s′)*U[s′] for s′ in 𝒮)
end

function update!(model::BayesianMDP, s, a, r, s′)
    α = model.D[s,a].alpha
    α[s′] += 1
    model.D[s,a] = Dirichlet(α)
    update!(model.planner, model, s, a, r, s′)
    return model
end
```

Algorithm 16.8. A Bayesian update method when the posterior distribution over transition models is represented as a product of Dirichlet distributions. We assume in this implementation that the reward model `R` is known, though we can use Bayesian methods to estimate expected reward from experience. The matrix `D` associates Dirichlet distributions with every state-action pair to model uncertainty in the transition to their successor states.

16.5 Bayes-Adaptive Markov Decision Processes

We can formulate the problem of acting optimally in an MDP with an unknown model as a higher-dimensional MDP with a known model. This MDP is known as a *Bayes-adaptive Markov decision process*, which is related to the partially observable Markov decision process discussed in part IV.

The state space in the Bayes-adaptive MDP is the Cartesian product $\mathcal{S} \times \mathcal{B}$, where $\mathcal{B}$ is the space of possible beliefs over the model parameters θ. Although $\mathcal{S}$ is discrete, $\mathcal{B}$ is often a high-dimensional continuous state space.[6] A state in a Bayes-adaptive MDP is a pair (s, b) consisting of the current state s in the base MDP and a belief state b. The action space and reward function are the same as in the base MDP.

[6] It is continuous in the case of Dirichlet distributions over transition probabilities, as shown in example 16.3.

The transition function in a Bayes-adaptive MDP is $T(s', b' \mid s, b, a)$, which is the probability of transitioning to a state s' with a belief state b', given that the agent starts in s with belief b and takes action a. The new belief state b' can be deterministically computed according to Bayes' rule. If we let this deterministic function be denoted as τ so that $b' = \tau(s, b, a, s')$, then we can decompose the Bayes-adaptive MDP transition function as

$$T(s', b' \mid s, b, a) = \delta_{\tau(s,b,a,s')}(b')\, P(s' \mid s, b, a) \tag{16.7}$$

where $\delta_x(y)$ is the *Kronecker delta function*[7] such that $\delta_x(y) = 1$ if $x = y$, and 0 otherwise.

[7] This function is named after the German mathematician Leopold Kronecker (1823–1891).

The second term can be computed using integration:

$$P(s' \mid s, b, a) = \int_\theta b(\theta) P(s' \mid s, \theta, a)\, \mathrm{d}\theta = \int_\theta b(\theta) \theta_{(s,a,s')}\, \mathrm{d}\theta \tag{16.8}$$

This equation can be evaluated analytically in a manner similar to equation (15.1). In the case where our belief b is represented by the factored Dirichlet in equation (16.6), we have

$$P(s' \mid s, b, a) = N(s, a, s') / \sum_{s''} N(s, a, s'') \tag{16.9}$$

We can generalize the Bellman optimality equation (equation (7.16)) for MDPs with a known model to the case in which the model is unknown:

$$U^*(s, b) = \max_a \left(R(s, a) + \gamma \sum_{s'} P(s' \mid s, b, a) U^*\left(s', \tau\left(s, b, a, s'\right)\right) \right) \tag{16.10}$$

Unfortunately, we cannot simply directly apply policy iteration or value iteration because b is continuous. We can, however use the approximation methods of chapter 8 or the online methods of chapter 9. Part IV presents methods that better use the structure of the Bayes-adaptive MDP.

16.6 Posterior Sampling

An alternative to solving for the optimal value function over the belief space is to use *posterior sampling*,[8] which was originally introduced in the context of exploration in bandit problems in section 15.4.[9] Here, we draw a sample θ from the current belief b and then solve for the best action, assuming that θ is the true model. We then update our belief, draw a new sample, and solve the corresponding MDP. Example 16.4 provides an example instance of this.

An advantage of posterior sampling is that we do not have to decide on heuristic exploration parameters. However, solving the MDP at every step can be expensive. A method for sampling a discrete MDP from the posterior is implemented in algorithm 16.9.

[8] M. J. A. Strens, "A Bayesian Framework for Reinforcement Learning," in *International Conference on Machine Learning (ICML)*, 2000.

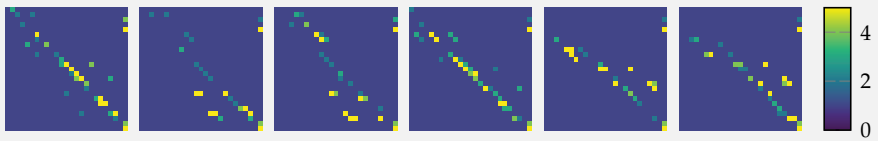
[9] In that section, we sampled from a posterior distribution over the probability of payoffs and then assumed that the sampled probabilities were correct when selecting an action.

```
struct PosteriorSamplingUpdate end

function Base.rand(model::BayesianMDP)
    S, A = model.S, model.A
    T = zeros(length(S), length(A), length(S))
    for s in S
        for a in A
            T[s,a,:] = rand(model.D[s,a])
        end
    end
    return MDP(T, model.R, model.γ)
end

function update!(planner::PosteriorSamplingUpdate, model, s, a, r, s′)
    P = rand(model)
    U = solve(P).U
    copy!(model.U, U)
end
```

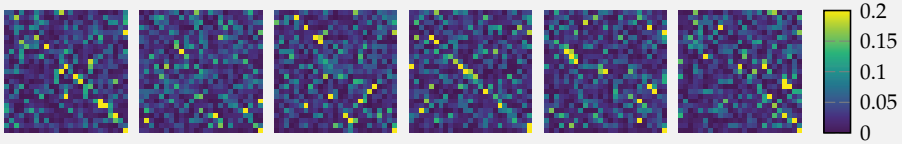
Algorithm 16.9. The update method for posterior sampling. After updating the parameters of the Bayesian posterior, we sample an MDP problem from that posterior. This implementation assumes a discrete state and action space with a Dirichlet modeling our uncertainty in the transition probabilities from each state-action pair. To generate the transition model, we iterate over every state and action and sample from the associated Dirichlet distribution. Once we have a sampled problem `P`, we solve it using the linear programming formulation and store the resulting value function `U`.

Example 16.4. Application of Bayesian model estimation and posterior sampling to the hex world problem.

We want to apply Bayesian model estimation to hex world. We start with associating uniform Dirichlet priors with every state-action pair. After 100 simulations of length 10 and adding our transition counts to our pseudo-counts in our prior, the parameters of our posterior distributions over our successor states appear as follows:



We can sample from this distribution to produce the model shown here. Notice that it has many more nonzero transition probabilities than the maximum likelihood models shown in example 16.1.



16.7 Summary

- Model-based methods learn the transition and reward models through interaction with the environment.
- Maximum likelihood models use transition counts to maintain an estimate of the transition probabilities to successor states and to track the mean reward associated with state-action pairs.
- Maximum likelihood models must be paired with an exploration strategy, such as those introduced in the previous chapter in the context of bandits.
- Although we can replan with each step of experience, doing so exactly can be costly.
- Prioritized sweeping can focus replanning by updating the values of states that appear to need it the most in our evolving model of the environment.
- Bayesian model-based methods maintain a probability distribution over possible problems, allowing principled reasoning about exploration.
- In Bayes-adaptive MDPs, their states augment the original MDP with the probability distribution over the possible MDP models.
- Posterior sampling reduces the high computational complexity of solving a Bayes-adaptive MDP by solving an MDP sampled from the belief state rather than reasoning about all possible MDPs.

16.8 Exercises

Exercise 16.1. Suppose we have an agent interacting in an environment with three states and two actions with unknown transition and reward models. We perform one sequence of direct interaction with the environment. Table 16.1 tabulates the state, action, reward, and resulting state. Use maximum likelihood estimation to estimate the transition and reward functions from this data.

Table 16.1. Transition data.

s	a	r	s'
s_2	a_1	2	s_1
s_1	a_2	1	s_2
s_2	a_2	1	s_1
s_1	a_2	1	s_2
s_2	a_2	1	s_3
s_3	a_2	2	s_2
s_2	a_2	1	s_3
s_3	a_2	2	s_3
s_3	a_1	2	s_2
s_2	a_1	2	s_3

Solution: We first tabulate the number of transitions from each state and action $N(s, a)$, the rewards received $\rho(s, a)$, and the maximum likelihood estimate of the reward function $\hat{R}(s, a) = \rho(s, a)/N(s, a)$ as follows:

s	a	$N(s,a)$	$\rho(s,a)$	$\hat{R}(s,a) = \frac{\rho(s,a)}{N(s,a)}$
s_1	a_1	0	0	0
s_1	a_2	2	2	1
s_2	a_1	2	4	2
s_2	a_2	3	3	1
s_3	a_1	1	2	2
s_3	a_2	2	4	2

In the next set of tables, we compute the number of observed transitions $N(s,a,s')$ and the maximum likelihood estimate of the transition model $\hat{T}(s' \mid s,a) = N(s,a,s')/N(s,a)$. When $N(s,a) = 0$, we use a uniform distribution over the resulting states.

s	a	s'	$N(s,a,s')$	$\hat{T}(s' \mid s,a) = \frac{N(s,a,s')}{N(s,a)}$
s_1	a_1	s_1	0	1/3
s_1	a_1	s_2	0	1/3
s_1	a_1	s_3	0	1/3
s_1	a_2	s_1	0	0
s_1	a_2	s_2	2	1
s_1	a_2	s_3	0	0
s_2	a_1	s_1	1	1/2
s_2	a_1	s_2	0	0
s_2	a_1	s_3	1	1/2
s_2	a_2	s_1	1	1/3
s_2	a_2	s_2	0	0
s_2	a_2	s_3	2	2/3
s_3	a_1	s_1	0	0
s_3	a_1	s_2	1	1
s_3	a_1	s_3	0	0
s_3	a_2	s_1	0	0
s_3	a_2	s_2	1	1/2
s_3	a_2	s_3	1	1/2

Exercise 16.2. Provide a lower bound and an upper bound on the number of updates that could be performed during an iteration of prioritized sweeping.

Solution: A lower bound on the number of updates performed in an iteration of prioritized sweeping is 1. This could occur during our first iteration using a maximum likelihood model, where the only nonzero entry in our transition model is $T(s' \mid s,a)$. Since no

state-action pairs (s^-, a^-) transition to s, our priority queue would be empty, and thus the only update performed would be for $U(s)$.

An upper bound on the number of updates performed in an iteration of prioritized sweeping is $|\mathcal{S}|$. Suppose that we just transitioned to s', and $\hat{T}(s' \mid s, a) > 0$ for all s and a. If we do not provide a maximum number of updates, we will perform $|\mathcal{S}|$ updates. If we provide a maximum number of updates $m < |\mathcal{S}|$, the upper bound is reduced to m.

Exercise 16.3. In performing Bayesian reinforcement learning of the transition model parameters for a discrete MDP with state space $\mathcal{S}$ and action space $\mathcal{A}$, how many independent parameters are there when using Dirichlet distributions to represent uncertainty over the transition model?

Solution: For each state and action, we specify a Dirichlet distribution over the transition probability parameters, so we will have $|\mathcal{S}||\mathcal{A}|$ Dirichlet distributions. Each Dirichlet is specified using $|\mathcal{S}|$ independent parameters. In total, we have $|\mathcal{S}|^2|\mathcal{A}|$ independent parameters.

Exercise 16.4. Consider the problem statement in exercise 16.1, but this time we want to use Bayesian reinforcement learning with a prior distribution represented by a Dirichlet distribution. Assuming a uniform prior, what is the posterior distribution over the next state, given that we are in state s_2 and take action a_1?

Solution: $\text{Dir}(\theta_{(s_2,a_1)} \mid [2, 1, 2])$