10 Markov Decision Processes

> **Note**
>
> This page contains all content from the legacy [PDF notes](#); markov decision processes chapter.
>
> As we phase out the PDF, this page may receive updates not reflected in the static PDF.

Consider a robot learning to navigate through a maze, a game-playing AI developing strategies through self-play, or a self-driving car making driving decisions in real-time. These problems share a common challenge: the agent must make a sequence of decisions where each choice affects future possibilities and rewards. Unlike static prediction tasks where we learn a one-time mapping from inputs to outputs, these problems require reasoning about the consequences of actions over time.

This sequential and dynamical nature demands mathematical tools beyond the more static supervised or unsupervised learning approaches. The most general framework for such problems is *reinforcement learning* (RL), where an agent learns to take actions in an unknown environment to maximize cumulative rewards over time.

In this chapter, we'll first study *Markov decision processes* (MDPs), which provide the mathematical foundation for understanding and solving sequential decision making problems like RL. MDPs formalize the interaction between an agent and its environment, capturing the key elements of states, actions, rewards, and transitions.

10.1 Definition and value functions

Formally, a Markov decision process is $\langle \mathcal{S}, \mathcal{A}, \mathrm{T}, \mathrm{R}, \gamma \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and:

- $\mathrm{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a *transition model*, where

$$\mathrm{T}(s, a, s') = Pr(S_t = s' | S_{t-1} = s, A_{t-1} = a) \ ,$$

  specifying a conditional probability distribution;

- $\mathrm{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a *reward function*, where $\mathrm{R}(s, a)$ specifies an immediate reward for taking action a when in state s; and

- $\gamma \in [0, 1]$ is a *discount factor*, which we'll discuss in Section 10.1.2.

The notation $S_t = s'$ uses a capital letter S to stand for a random variable, and small letter s to stand for a concrete value. So S_t here is a random variable that can take on elements of $\mathcal{S}$ as values.

In this class, we assume the rewards are deterministic functions. Further, in this MDP chapter, we assume the state space and action space are discrete and finite.

The following description of a simple machine as Markov decision process provides a concrete example of an MDP.

The machine has three possible operations (*actions*): `wash`, `paint`, and `eject` (each with a corresponding button). Objects are put into the machine, and each time you push a button, something is done to the object. However, it's an old machine, so it's not very reliable. The machine has a camera inside that can clearly detect what is going on with the object and will output the state of the object: `dirty`, `clean`, `painted`, or `ejected`.

For each action, this is what is done to the object:

Wash

- If you perform the `wash` operation on any object—whether it's dirty, clean, or painted—it will end up `clean` with probability 0.9 and `dirty` otherwise.
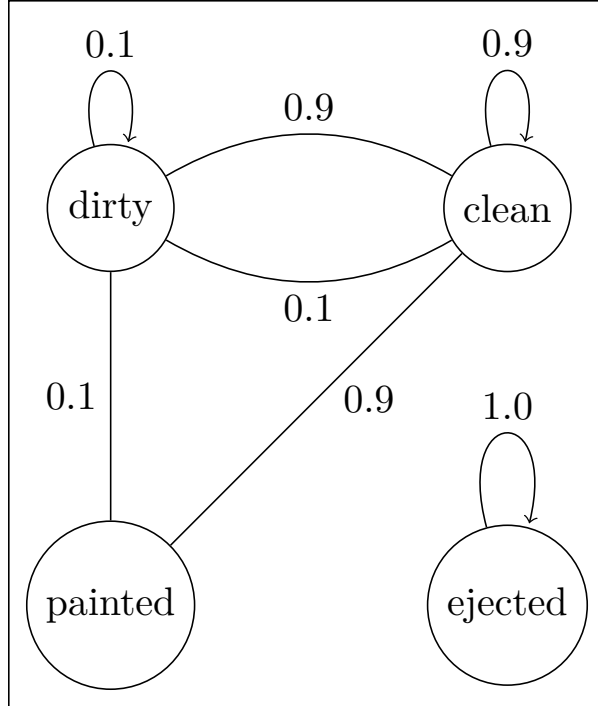
Paint

- If you perform the `paint` operation on a clean object, it will become nicely `painted` with probability 0.8. With probability 0.1, the paint misses but the object stays clean, and with probability 0.1, the machine dumps rusty dust all over the object, making it `dirty`.
- If you perform the `paint` operation on a `painted` object, it stays `painted` with probability 1.0.
- If you perform the `paint` operation on a `dirty` object, it stays `dirty` with probability 1.0.

Eject

- If you perform an `eject` operation on any object, the object comes out of the machine and the process is terminated. The object remains `ejected` regardless of any further actions.

These descriptions specify the transition model T, and the transition function for each action can be depicted as a state machine diagram. For example, here is the diagram for `wash`:

You get reward +10 for ejecting a painted object, reward 0 for ejecting a non-painted object, reward 0 for any action on an "ejected" object, and reward -3 otherwise. The MDP description would be completed by also specifying a discount factor.

A *policy* is a function π that specifies what action to take in each state. The policy is what we will want to learn; it is akin to the strategy that a player employs to win a given game. Below, we take just the initial steps towards this eventual goal. We describe how to evaluate how good a policy is, first in the *finite horizon* case Section 10.1.1 when the total number of transition steps is finite. In the finite horizon case, we typically denote the policy as $\pi_h(s)$, where h is a non-negative integer denoting the number of steps remaining and $s \in \mathcal{S}$ is the current state. Then we consider the *infinite horizon* case Section 10.1.2, when you don't know when the game will be over.

10.1.1 Finite-horizon value functions

The goal of a policy is to maximize the expected total reward, averaged over the stochastic transitions that the domain makes. Let's first consider the case where there is a finite *horizon h*, indicating the total number of steps of interaction that the agent will have with the MDP.

We seek to measure the goodness of a policy. We do so by defining for a given horizon h and MDP policy π_h, the "horizon h *value*" of a state, $V_h^\pi(s)$. We do this by induction on the horizon, which is the *number of steps left to go*.

The base case is when there are no steps remaining, in which case, no matter what state we're in, the value is 0, so

$$V_0^\pi(s) = 0 \ . \tag{10.1}$$

Then, the value of a policy in state s at horizon $h + 1$ is equal to the reward it will get in state s plus the next state's expected horizon h value, discounted by a factor γ

. So, starting with horizons 1 and 2, and then moving to the general case, we have:

$$V_1^\pi(s) = R(s, \pi_1(s)) + 0 \tag{10.2}$$

$$V_2^\pi(s) = R(s, \pi_2(s)) + \gamma \sum_{s'} T(s, \pi_2(s), s') V_1^\pi(s') \tag{10.3}$$

$$\vdots$$

$$V_h^\pi(s) = R(s, \pi_h(s)) + \gamma \sum_{s'} T(s, \pi_h(s), s') V_{h-1}^\pi(s') \tag{10.4}$$

The sum over s' is an *expectation*: it considers all possible next states s', and computes an average of their $(h-1)$-horizon values, weighted by the probability that the transition function from state s with the action chosen by the policy $\pi_h(s)$ assigns to arriving in state s', and discounted by γ.

> **? Study Question**
>
> What is the value of
>
> $$\sum_{s'} T(s, a, s')$$
>
> for any given state–action pair (s, a)?

> **? Study Question**
>
> Convince yourself that the definitions in Equation 10.1 and Equation 10.3 are special cases of the more general formulation in Equation 10.4.

Then we can say that a policy π is better than policy $\bar{\pi}$ for horizon h if and only if for all $s \in \mathcal{S}$, $V_h^\pi(s) \geq V_h^{\bar{\pi}}(s)$ and there exists at least one $s \in \mathcal{S}$ such that $V_h^\pi(s) > V_h^{\bar{\pi}}(s)$.

10.1.2 Infinite-horizon value functions

More typically, the actual finite horizon is not known, i.e., when you don't know when the game will be over! This is called the *infinite horizon* version of the problem. How does one evaluate the goodness of a policy in the infinite horizon case?

If we tried to simply take our definitions above and use them for an infinite horizon, we could get in trouble. Imagine we get a reward of 1 at each step under one policy and a reward of 2 at each step under a different policy. Then the reward as the number of steps grows in each case keeps growing to become infinite in the limit of more and more steps. Even though it seems intuitive that the second policy should be better, we can't justify that by saying $\infty < \infty$.

One standard approach to deal with this problem is to consider the *discounted* infinite horizon. We will generalize from the finite-horizon case by adding a discount factor.

In the finite-horizon case, we valued a policy based on an expected finite-horizon value:

$$\mathbb{E}\left[\sum_{t=0}^{h-1} \gamma^t R_t \mid \pi, s_0\right] , \qquad (10.5)$$

where R_t is the reward received at time t.

> **Note**
>
> What is $\mathbb{E}[\cdot]$? This mathematical notation indicates an *expectation*, i.e., an average taken over all the random possibilities which may occur for the argument. Here, the expectation is taken over the *conditional probability* $Pr(R_t = r \mid \pi, s_0)$, where R_t is the random variable for the reward, subject to the policy being π and the state being s_0. Since π is a function, this notation is shorthand for conditioning on all of the random variables implied by policy π and the stochastic transitions of the MDP.
>
> A very important point is that $R(s, a)$ is always deterministic (in this class) for any given s and a. Here R_t represents the set of all possible $R(s_t, a)$ at time step t; this R_t is a random variable because the state we're in at step t is itself a random variable, due to prior stochastic state transitions up to but not including at step t and prior (deterministic) actions dictated by policy π.

Now, for the infinite-horizon case, we select a discount factor $0 \leq \gamma \leq 1$, and evaluate a policy based on its expected *infinite horizon value*:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid \pi, s_0\right] = \mathbb{E}\left[R_0 + \gamma R_1 + \gamma^2 R_2 + \ldots \mid \pi, s_0\right] . \qquad (10.6)$$

Note that the t indices here are not the number of steps to go, but actually the number of steps forward from the starting state (there is no sensible notion of "steps to go" in the infinite horizon case).

> **Note**
>
> Equation 10.5 and Equation 10.6 are a conceptual stepping stone. Our main objective is to get to Equation 10.8, which can also be viewed as including γ in Equation 10.4, with the appropriate definition of the infinite-horizon value.

There are two good intuitive motivations for discounting. One is related to economic theory and the present value of money: you'd generally rather have some money today than that same amount of money next week (because you could use it now or invest it). The other is to think of the whole process terminating, with probability $1 - \gamma$ on each step of the interaction. (At every step, your expected future lifetime, given that you have survived until now, is $1/(1 - \gamma)$.) This value is the expected amount of reward the agent would gain under this terminating model.

Let us now evaluate a policy in terms of the expected discounted infinite-horizon value that the agent will get in the MDP if it executes that policy. We define the infinite-horizon value of a state s under policy π as

$$
\begin{aligned}
V_\infty^\pi(s) &= \mathbb{E}[R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots \mid \pi, S_0 = s] \\
&= \mathbb{E}[R_0 + \gamma(R_1 + \gamma(R_2 + \gamma \ldots))) \mid \pi, S_0 = s] \ .
\end{aligned} \tag{10.7}
$$

Because the expectation of a linear combination of random variables is the linear combination of the expectations, we have

$$
\begin{aligned}
V_\infty^\pi(s) &= \mathbb{E}[R_0 \mid \pi, S_0 = s] + \gamma \mathbb{E}[R_1 + \gamma(R_2 + \gamma \ldots))) \mid \pi, S_0 = s] \\
&= R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V_\infty^\pi(s') \ .
\end{aligned} \tag{10.8}
$$

The equation defined in [Equation 10.8](#) is known as the Bellman Equation, which breaks down the value function into the immediate reward and the (discounted) future value function. You could write down one of these equations for each of the $n = |\mathcal{S}|$ states. There are n unknowns $V^\pi(s)$. These are linear equations, and standard software (e.g., using Gaussian elimination or other linear algebraic methods) will, in most cases, enable us to find the value of each state under this policy.

10.2 Finding policies for MDPs

Given an MDP, our goal is typically to find a policy that is optimal in the sense that it gets as much total reward as possible, in expectation over the stochastic transitions that the domain makes. We build on what we have learned about evaluating the goodness of a policy ([Section 10.1.2](#)), and find optimal policies for the finite horizon case ([Section 10.2.1](#)), then the infinite horizon case ([Section 10.2.2](#)).

10.2.1 Finding optimal finite-horizon policies

How can we go about finding an optimal policy for an MDP? We could imagine enumerating all possible policies and calculating their value functions as in the previous section and picking the best one – but that's too much work!

The first observation to make is that, in a finite-horizon problem, the best action to take depends on the current state, but also on the horizon: imagine that you are in a situation where you could reach a state with reward 5 in one step or a state with reward 100 in two steps. If you have at least two steps to go, then you'd move

toward the reward 100 state, but if you only have one step left to go, you should go in the direction that will allow you to gain 5!

For the finite-horizon case, we define $Q_h^*(s, a)$ to be the expected value of

- starting in state s,

- executing action a, and

- continuing for $h - 1$ more steps executing an optimal policy for the appropriate horizon on each step.

Similar to our definition of V_h^* for evaluating a policy, we define the Q_h^* function recursively according to the horizon. The only difference is that, on each step with horizon h, rather than selecting an action specified by a given policy, we select the value of a that will maximize the expected Q_h^* value of the next state.

$$Q_0^*(s, a) = 0$$
$$Q_1^*(s, a) = R(s, a) + 0$$
$$Q_2^*(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_1^*(s', a')$$
$$\vdots$$
$$Q_h^*(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{h-1}^*(s', a')$$

where (s', a') denotes the next time-step state/action pair. We can solve for the values of Q_h^* with a simple recursive algorithm called *finite-horizon value iteration* that just computes Q_h^* starting from horizon 0 and working backward to the desired horizon h. Given Q_h^*, an optimal π_h^* can be found as follows:

$$\pi_h^*(s) = \arg\max_a Q_h^*(s, a) \ .$$

which gives the *immediate* best action(s) to take when there are h steps left; then $\pi_{h-1}^*(s)$ gives the best action(s) when there are $(h - 1)$ steps left, and so on. In the case where there are multiple best actions, we typically can break ties randomly.

Additionally, it is worth noting that in order for such an optimal policy to be computed, we assume that the reward function $R(s, a)$ is bounded on the set of all possible (state, action) pairs. Furthermore, we will assume that the set of all possible actions is finite.

> **? Study Question**
>
> The optimal value function is unique, but the optimal policy is not. Think of a situation in which there is more than one optimal policy.

We can also define the action-value function for a fixed policy π, denoted by $Q_h^\pi(s, a)$. This quantity represents the expected sum of discounted rewards obtained by taking action a in state s and thereafter following the policy π over the remaining horizon of $h - 1$ steps.

Similar to $V_h^\pi(s)$, $Q_h^\pi(s, a)$ satisfies the Bellman recursion/equations introduced earlier. In fact, for a deterministic policy π:

$$Q_h^\pi(s, \pi(s)) = V_h^\pi(s).$$

However, since our primary goal in dealing with action values is typically to identify an *optimal* policy, we will not dwell extensively on $(Q_h^\pi(s, a))$. Instead, we will place more emphasis on the optimal action-value functions $Q_h^*(s, a)$.

10.2.2 Finding optimal infinite-horizon policies

In contrast to the finite-horizon case, the best way of behaving in an infinite-horizon discounted MDP is not time-dependent. That is, the decisions you make at time $t = 0$ looking forward to infinity, will be the same decisions that you make at time $t = T$ for any positive T, also looking forward to infinity.

An important theorem about MDPs is: in the infinite-horizon case, there exists a stationary optimal policy π^* (there may be more than one) such that for all $s \in \mathcal{S}$ and all other policies π, we have

$$V^\pi(s) \leq V^{\pi^*}(s) \ .$$

There are many methods for finding an optimal policy for an MDP. We have already seen the finite-horizon value iteration case. Here we will study a very popular and useful method for the infinite-horizon case, *infinite-horizon value iteration*. It is also important to us, because it is the basis of many *reinforcement-learning* methods.

We will again assume that the reward function $R(s, a)$ is bounded on the set of all possible (state, action) pairs and additionally that the number of actions in the action space is finite. Define $Q^*_\infty(s, a)$ to be the expected infinite-horizon value of being in state s, executing action a, and executing an optimal policy π^* thereafter. Using similar reasoning to the recursive definition of V^π, we can express this value recursively as

$$Q^*_\infty(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q^*_\infty(s', a') \ .$$

This is also a set of equations, one for each (s, a) pair. This time, though, they are not linear (due to the max operation), and so they are not easy to solve. But there is a theorem that says they have a unique solution!

Once we know the optimal action-value function $Q^*_\infty(s, a)$, then we can extract an optimal policy π^* as

$$\pi^*(s) = \arg\max_a Q^*_\infty(s, a)$$

We can iteratively solve for the Q^* values with the infinite-horizon value iteration algorithm, shown below:

Algorithm 10.1 Infinite-Horizon-Value-Iteration

Require: $\mathcal{S}, \mathcal{A}, T, R, \gamma, \epsilon$
1: **Initialization:**
2: **for** each $s \in \mathcal{S}$ and $a \in \mathcal{A}$ **do**
3:    $Q_{\text{old}}(s, a) \leftarrow 0$
4: **end for**
5: **while** not converged **do**
6:    for each $s \in \mathcal{S}$ and $a \in \mathcal{A}$ **do**
7:       $Q_{\text{new}}(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{\text{old}}(s', a')$
8:    end for
9:    if $\max_{s,a} |Q_{\text{old}}(s, a) - Q_{\text{new}}(s, a)| < \epsilon$ then
10:       return Q_{new}
11:    end if

12:    $Q_{old} \leftarrow Q_{new}$
13: **end while**

Theory

There are a lot of nice theoretical results about infinite-horizon value iteration. For some given (not necessarily optimal) Q function, define $\pi_Q(s) = \arg\max_a Q(s, a)$.

- After executing infinite-horizon value iteration with convergence hyper-parameter ϵ,

$$\|V^{\pi_{Q_{new}}} - V^{\pi^*}\|_{\max} < \epsilon .$$

- There is a value of ϵ such that

$$\|Q_{old} - Q_{new}\|_{\max} < \epsilon \implies \pi_{Q_{new}} = \pi^*$$

- As the algorithm executes, $\|V^{\pi_{Q_{new}}} - V^{\pi^*}\|_{\max}$ decreases monotonically on each iteration.

- The algorithm can be executed asynchronously, in parallel: as long as all (s, a) pairs are updated infinitely often in an infinite run, it still converges to the optimal value.