Algorithms and Uncertainty, Winter 2024/25 Lecture 8 (5 pages)

Markov Decision Processes

Thomas Kesselheim Last Update: November 2, 2025

As a motivation, consider the following game: There are n envelopes. Envelope i contains prize $v_i \geq 0$ with probability $q_i \in [0,1]$. With probability $1 - q_i$ it is empty. You may open envelopes and keep the prizes as long as you do not open an empty envelope.

What is the best strategy to play this game? One might be tempted to act myopically: Open the envelope of highest expected reward $q_i \cdot v_i$. This is, for example, a bad idea in the following setting:

$$v_1 = 1000, \quad q_1 = \frac{1}{100}, \qquad v_i = q_i = 1 \text{ for } i > 1 \ .$$

Here, we would open envelope 1 first but with 99 % chance, we do not get anything. It is much better to first open envelopes $2, \dots, n$ and only then to take the chance and open envelope 1.

Today's goal will be to introduce a general model for such stochastic decision problems, to describe optimal policies and give algorithms to compute them.

1 Markov Decision Processes

A Markov Decision Process is defined by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a reward function that defines a reward $r_a(s)$ for taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ and a random transition function, which is defined by probabilities $p_a(s, s')$: If we are in state $s \in \mathcal{S}$ and we take action $a \in \mathcal{A}$, then we move on to state $s' \in \mathcal{S}$ with probability $p_a(s, s') \in [0, 1]$.

The process works as follows. We start from state $s_1 \in \mathcal{S}$, choose one action $a \in \mathcal{A}$. We immediately get reward $r_a(s) \in \mathbb{R}$ and then continue to a random state s', which is given by the probability distribution $p_a(s_1, \cdot)$. This way, a sequence $s_1, s_2, \dots$ evolves. We move from s_t to s_{t+1} by the probability distribution $p_a(s_t, \cdot)$. So, the probabilities only depend on the current state and the current action but not on which states we have seen before. This makes the process *Markovian*.

Generally, rewards may also be random, just as in our example above. To capture this, set $r_a(s)$ to the *expected* reward that you get when taking action a in state s.

On the one hand, this generalizes a deterministic finite automaton. Here, for each a and s, there is exactly one s' for which $p_a(s, s') = 1$ and $p_a(s, s') = 0$ otherwise. On the other hand, it is also a generalization of a Markov chain. Here, $\mathcal{A}$ has only one element (an action like "continue") and then we move through states without having a real choice.

Example 8.1. *Let us define the Markov decision process for the motivating example with the envelopes. In the state, we have to keep track which envelopes were opened so far and if any of them was empty. This is done by* $\mathcal{S} = 2^{[n]} \cup \{\text{STOP}\}$*, where* $[n] = \{1, \dots, n\}$*.*

Each action corresponds to opening an envelope. Therefore, $\mathcal{A} = [n]$*. Let us define the state transitions. For* $s \in 2^{[n]}$*, we set* $p_a(s, s \cup \{a\}) = q_a$ *and* $p_a(s, \text{STOP}) = 1 - q_a$ *for all* $a \in [n]$*. Furthermore,* $p_a(\text{STOP}, \text{STOP}) = 1$ *to ensure that we remain in state* STOP *once an envelope was empty. All other probabilities are set to* 0*.*

The reward from opening an envelope is the expected *prize inside, so* $r_a(s) = q_a \cdot v_a$ *for* $s \in 2^{[n]}$ *with* $a \notin s$*.*

In principle, in any state, any action may be performed. So, we cannot forbid a policy to open an envelope twice. We can only penalize this is terms of rewards, for example by setting $r_a(s) = -\infty$ *if* $s \in 2^{[n]}$*,* $a \in s$*. Also in state* STOP*, the actions are still available although no*

more envelopes may be opened. By setting $r_a(\text{STOP}) = 0$ for all $a \in \mathcal{A}$, effectively these actions do not matter.

2 Policies and Their Structure

An algorithm making decisions in a Markov decision process is called a *policy*. Formally, a policy π assigns to each sequences of states $s_1, \ldots, s_t \in \mathcal{S}$ an action $\pi(s_1, \ldots, s_t) \in \mathcal{A}$. This action then causes a transition to the next state. For a policy π, we let $s_1^\pi, s_2^\pi, \ldots$ denote the random sequence of states $s_1^\pi, s_2^\pi, \ldots$, caused by the actions $a_1^\pi, a_2^\pi, \ldots$, where $a_t^\pi = \pi(s_1^\pi, \ldots, s_t^\pi)$ and s_{t+1}^π is the state reached from s_t^π when action a_t^π is chosen.

Generally, we can move through a Markov decision process for unbounded time. Our motivating example as well as many other important applications can be captured by a finite time horizon. That is, there is some T such that we do not care what happens after time T. In this case, we can write the expected reward of policy π when starting at s_1 as

$$V(\pi, s_1, T) = \mathbf{E}\left[\sum_{t=1}^{T} r_{a_t^\pi}(s_t^\pi)\right] \ .$$

We also define $V^*(s_1, T)$ as the highest expected reward that one can achieve starting from s_1 in T steps, that is, $V^*(s_1, T) = \max_{\text{policy } \pi} V(\pi, s_1, T)$. (Note that there are only finitely many histories $s_1, \ldots, s_{t-1}$ for $t \leq T$ and therefore only finitely many different policies, so the maximum is well-defined.)

Consider an optimal policy π, that is $V(\pi, s_1, T) = V^*(s_1, T)$. As a_1^π is deterministic, we might as well write

$$V(\pi, s_1, T) = r_{a_1^\pi}(s_1) + \mathbf{E}\left[\sum_{t=2}^{T} r_{a_t^\pi}(s_t^\pi)\right] = r_{a_1^\pi}(s_1) + \sum_{s' \in \mathcal{S}} p_{a_1^\pi}(s_1, s')\mathbf{E}\left[\sum_{t=2}^{T} r_{a_t^\pi}(s_t^\pi) \,\middle|\, s_2^\pi = s'\right] \ .$$

Let us inspect the expectation on the right-hand side. We claim that

$$\mathbf{E}\left[\sum_{t=2}^{T} r_{a_t^\pi}(s_t^\pi) \,\middle|\, s_2^\pi = s'\right] = V^*(s', T-1) \ .$$

The reason is simple: Both is the maximum expected reward that we would receive from a Markov decision process running for $T-1$ steps, starting from s'. On the left-hand side, we actually start from s_1 but this does not make a difference for the remaining steps. Importantly, rewards in the current step only depend on the current state and action, not on the past ones.

We skip the fleshed out formal argument here. One possible way is to assume that either side is strictly larger than the other and observe that one could either add or remove s_1 from the beginning of the history.

Consequently, we can define $V^*(s, T)$ for $T \geq 1$ recursively as

$$V^*(s, T) = \max_{a \in \mathcal{A}} \left(r_a(s) + \sum_{s' \in \mathcal{S}} p_a(s, s') V^*(s', T-1) \right) \tag{1}$$

and $V^*(s, 0) = 0$. These observations now lead us to the following theorem.

Theorem 8.2. *An optimal policy for a time horizon of T steps can be computed in time $O(T \cdot |\mathcal{S}|^2 \cdot |\mathcal{A}|)$. Moreover the computed policy is Markovian.*

Besides stating that an optimal policy can indeed be computed, this theorem gives us also an insight into the structure of optimal policies.

Definition 8.3. *A policy π is* Markovian *if actions only depend on the current state and the number of remaining steps. For a Markovian policy, we write $\pi(s, T')$ for the action being taken when there are T' steps remaining.*

Proof of Theorem 8.2. We can compute an optimal policy by dynamic programming using the following algorithm.

- Initialize $V^*(s, 0) = 0$ for all $s \in \mathcal{S}$
- For $T' = 1, \dots, T$
  - For all $s \in \mathcal{S}$
    - Set $V^*(s, T') = \max_{a \in \mathcal{A}} \left(r_a(s) + \sum_{s' \in \mathcal{S}} p_a(s, s') V^*(s', T' - 1) \right)$
    - Let $\pi(s, T')$ be any action $a \in \mathcal{A}$ that maximizes $r_a(s) + \sum_{s' \in \mathcal{S}} p_a(s, s') V^*(s', T' - 1)$

The policy π we compute is Markovian because $\pi(s, T')$ only depends on s and T'. It is optimal because by construction $V(\pi, s, T') = V^*(s, T')$ for all s and T'.

Regarding the running time, we observe that we compute $T \cdot |\mathcal{S}|$ values of V^* in total, each computation takes $|\mathcal{S}| \cdot |\mathcal{A}|$ steps. □

Example 8.4. *Let us come back to our initial example with the envelopes. The time horizon will be $T = n$ because, in principle, all envelopes can be opened. Suppose we have only two envelopes with, where*

$$v_1 = 1000, \quad q_1 = \frac{1}{100}, \qquad v_2 = q_2 = 1 \ .$$

Our intuition tells us that we should open envelope 2 first and then envelope 1. This gives us an expected reward of $1 + 10 = 11$. By dynamic programming, we get the following table, where each column corresponds to one state, each line to one time step and the entries to the maximum expected reward:

T'	$\emptyset$	$\{1\}$	$\{2\}$	$\{1, 2\}$	STOP
0	0	0	0	**0**	**0**
1	10	**1**	**10**	$-\infty$	**0**
2	**11**	$-\infty$	$-\infty$	$-\infty$	0

Note that only the bold numbers are actually reachable from $s_1 = \emptyset$ with $T = 2$. We compute $V^*(\emptyset, 2)$ as follows. We could choose 1 as the first action. In this case, the expected reward is $10 + \frac{1}{100} \cdot 1 + \frac{99}{100} \cdot 0 = 10.01$. Or, we could choose 2 as the first action. Then, the expected reward is $1 + 1 \cdot 10 = 11$.

3 Deriving the Optimal Policy

As we have just seen, in the initial example with the envelopes, the optimal policy can be computed via dynamic programming. The downside is that the number of states is $2^n + 1$. So, it is huge. And, as we will see, we can understand policies in a much more structured way.

Consider an optimal policy π^*. Note that π^* defines nothing but an order (i.e. a permutation) in which to open the envelopes as long as none was empty. We observe that if in the order

that π^* uses, i comes directly before j (meaning that it opens envelope i always before j), then the following has to hold. Let s be the set of indices of envelopes opened before i. Applying Equation (1) twice, we get

$$\begin{aligned} V^*(s,T) = V(\pi^*,s,T) &= q_i v_i + q_i V^*(s \cup \{i\}, T-1) \\ &= q_i v_i + q_i(q_j v_j + q_j V^*(s \cup \{i,j\}, T-2)) \ . \end{aligned}$$

It would also be feasible policy π to first open j, then i, and then continue according to π^*. This would give expected reward

$$V(\pi,s,T) = q_j v_j + q_j(q_i v_i + q_i V^*(s \cup \{i,j\}, T-2)) \ ,$$

because we only have to swap i and j in the previous expression. As π^* is optimal, we have $V(\pi^*,s,T) \geq V(\pi,s,T)$, and therefore

$$q_i v_i + q_i(q_j v_j + q_j V^*(s \cup \{i,j\}, T-2)) \geq q_j v_j + q_j(q_i v_i + q_i V^*(s \cup \{i,j\}, T-2)) \ ,$$

which is equivalent to

$$\frac{q_i v_i}{1-q_i} \geq \frac{q_j v_j}{1-q_j} \ .$$

Consequently, it is optimal to open to envelopes by non-increasing $\frac{q_i v_i}{1-q_i}$. The optimal policy is unique up to breaking these ties (and choosing the actions from STOP and any unreachable states, which are irrelevant).

4 Bonus: A Stochastic Model for Ski Rental

In our first lecture we introduced a stochastic model for the Ski Rental problem. We defined it such that there are T days; each day is a skiing day with probability q. We can buy skis once at a cost of B or rent them for a day at a cost of 1. This is, indeed also a very simple Markov decision process.

One potential way to model it is as follows. We have three states SKIING, NOT-SKIING, BOUGHT. There are two actions, RENT and BUY.

- In state SKIING, action RENT has a reward to -1 and makes us transition to SKIING with probability q and to NON-SKIING with probability $1-q$. Action BUY has a reward of $-B$ and makes us transition to BOUGHT.
- In state NON-SKIING, both actions have no reward. They make us transition to SKIING with probability q and to NON-SKIING with probability $1-q$.
- In state BOUGHT, every action always keeps us in BOUGHT and has no reward.

There is a small technicality: The starting state is always deterministic. However, we would like it to be random. To this end, we add a day 0 as a non-skiing day.

Crucially, the optimal policy is Markovian. This means that its choices only depend on the current state and how many steps are left. It is irrelevant how we came to this point.

To derive the optimal policy, we only have to understand under which circumstances it chooses BUY in state SKIING. Generally, it is possible to derive this from Equation (1): We would like to understand for which choices of T is the maximum (meaning the smaller cost) attained by $a =$ BUY if $s =$ SKIING. However, one easily gets lost in notation.

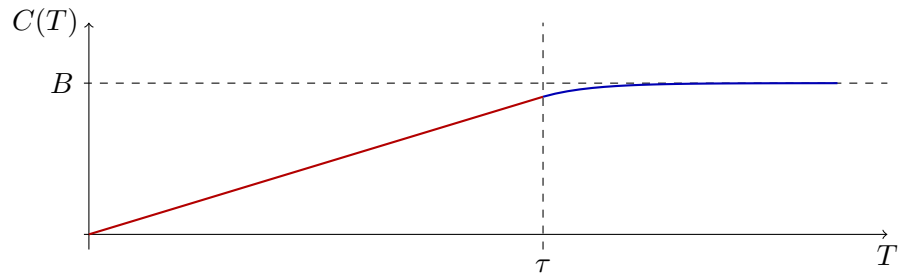

Figure 1: Values of $C(T)$ for $B = 10$, $q = \frac{1}{10}$. Right of τ, the function asymptotically approaches B (but never actually reaches it).

We switch from maximizing rewards to minimizing costs and let $C(T)$ denote the expected cost on a sequence of T days (not counting day zero). The key observation is that if the policy does not buy skis on the first day, it will face the exact same decision process with $T-1$ steps. For this reason

$$C(T) = \begin{cases} q(C(T-1)+1) + (1-q)C(T-1) & \text{if optimal policy rents in } (\textsc{skiing}, T) \\ qB + (1-q)C(T-1) & \text{otherwise} \end{cases}$$

The algorithm ALG* is defined to choose the cheaper of the two options. So

$$C(T) = q \min\{C(T-1)+1, B\} + (1-q)C(T-1) \ .$$

Let us understand this recursion. We start from $C(0) = 0$. Initially, we will have $C(T-1)+1 \le B$ and so $C(T) = C(T-1) + q$, which means $C(T) = qT$ for the first values of T. At some point τ we have $C(T-1)+1 > B$ for the first time. More precisely, this is the smallest τ such that $q(\tau-1)+1 > B$, which is $\tau = \left\lfloor \frac{B-1}{q} + 2 \right\rfloor$.

Note that $C(T-1)+1 > B$ to be true for larger T. That is, for all $T \ge \tau$, we have $C(T) = qB + (1-q)C(T-1)$.

One can solve this recursion further from here (see Figure 1 for the solution). However, this is not necessary to derive the optimal policy. Recall that the minimum in the recursion comes from the choice of the policy on the first day. We now know that in $(\textsc{skiing}, T)$, the optimal policy will buy the skis if $T \ge \tau$ and rent them otherwise. An interesting consequence is that the optimal policy will never first rent and then buy.