1228 Probabilistic Graphical Models
H. Zhu and J. Boedecker
May 1, 2026

Markov Decision Problems

Contents

1 Markov decision processes

A *Markov decision process* (MDP) can be used to model the interaction of an *agent* and an *environment*. At each time step t of the interaction, the agent receives some representation of the environment's *state* $s_t \in \mathcal{S}$, and follows a *policy* π to take an *action* $a_t \in \mathcal{A}$. In response, the environment emits a real-valued reward signal $r(s_t, a_t)$ and enters a new state $s_{t+1} \in \mathcal{S}$. A graphical representation of this process is shown in figure 1. The set $\mathcal{S}$ and $\mathcal{A}$ are called the state space and action space, respectively. The policy is in general stochastic, with $\pi(a \mid s)$ being the probability of choosing action a in state s. We use $\pi(s)$ to denote the conditional probability over $\mathcal{A}$ if the policy is stochastic, or the action it chooses if it is deterministic. The function $r\colon \mathcal{S} \times \mathcal{A} \to \mathbf{R}$ is called the *reward function*. The process at every step is called a *transition*; at time t, it consists of the tuple (s_t, a_t, s_{t+1}), where $a_t \sim \pi(s_t)$ and $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$. Hence, under policy π, the probability of generating a trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$ of length T can be written explicitly as

$$p(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t).$$

In general, the state and action sets of an MDP can be discrete or continuous. When both sets are finite, we can represent these functions as lookup tables; this is known as a *tabular representation*.

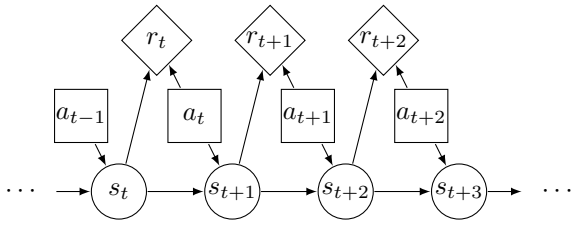

Figure 1 Graphical representation of an MDP.

Note that the field of control theory uses slightly different terminology. In particular, the environment is called the *plant*, and the agent is called the *controller*. States are denoted by $x_t \in \mathcal{X} \subseteq \mathbf{R}^m$, actions are denoted by $u_t \in \mathcal{U} \subseteq \mathbf{R}^n$, and rewards are denoted by costs $c_t \in \mathbf{R}$. In this course we will use the former notation because they are meaningful to a wider audience.

1.1 Episodes and returns

The Markov decision process describes how a trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots)$ is stochastically generated. If the agent can potentially interact with the environment forever, we call it a *continuing task*. Alternatively, the agent is in an *episodic task*, if its interaction terminates once the system enters a terminal state or absorbing state (the next state is always itself with 0 reward). After entering a terminal state, the agent will start a new episode from a new initial state $s_0 \sim p(s_0)$. The episode length is in general random. For example, the amount of time a robot takes to reach its goal may be quite variable, depending on the decisions it makes, and the randomness in the environment. Note that we can convert an episodic MDP to a continuing MDP by redefining the transition model in terminal states to be the initial-state distribution $p(s_0)$. Finally, if the trajectory length T in an episodic task is fixed and known, it is called a *finite horizon problem*.

Let τ be a trajectory of length T, where T may be ∞ if the task is continuing. We define the *return* for the state at time t to be the sum of expected rewards obtained going forwards, where each reward is multiplied by a *discount factor* $\gamma \in [0, 1]$:

$$\begin{aligned} G_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t-1} r_{T-1} \\ &= \sum_{k=0}^{T-t-1} \gamma^k r_{t+k} = \sum_{i=t}^{T-1} \gamma^{i-t} r_i. \end{aligned}$$

For episodic tasks that terminate at time T, we define $G_t = 0$ for $t \geq T$. Clearly, the return satisfies the following recursive relationship:

$$G_t = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \cdots) = r_t + \gamma G_{t+1}. \tag{1.1}$$

The discount factor γ plays two roles. First, it ensures the return G_t is finite even if $T \to \infty$ (*i.e.*, infinite horizon), if we use $\gamma < 1$ and the rewards r_t are

bounded. Second, it puts more weight on short-term rewards, which generally has the effect of encouraging the agent to achieve its goals more quickly. However, if γ is too small, the agent will become too greedy. In the extreme case where $\gamma = 0$, the agent is completely *myopic*, and only tries to maximize its immediate reward. In general, the discount factor reflects the assumption that there is a probability of $1 - \gamma$ that the interaction will end at the next step. For finite horizon problems, where T is known, we can set $\gamma = 1$, since we know the life time of the agent a priori.

1.2 Value functions

Let π be a given policy, we define the *state-value function* V, or *value function* for short, as follows:

$$V^\pi(s) = \underset{\pi}{\mathbf{E}}(G_0 \mid s_0 = s) = \underset{\pi}{\mathbf{E}}\left(\sum_{t=0}^{\infty} \gamma^t r_t \;\middle|\; s_0 = s\right),$$

for all $s \in \mathcal{S}$, where $\mathbf{E}_\pi$ indicating that actions are selected according to π. This is the expected return obtained if we start in state $s \in \mathcal{S}$ and follow π to choose actions in a continuing task. Similarly, we define the *action-value function* Q, also known as the *Q-function*, as follows:

$$Q^\pi(s, a) = \underset{\pi}{\mathbf{E}}(G_0 \mid s_0 = s, a_0 = a) = \underset{\pi}{\mathbf{E}}\left(\sum_{t=0}^{\infty} \gamma^t r_t \;\middle|\; s_0 = s, a_0 = a\right),$$

for all $s \in \mathcal{S}, a \in \mathcal{A}$. The action-value function represents the expected return obtained if we start by taking action a in state s, and then follow π to choose actions thereafter. Finally, we define the *advantage function* as follows:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s),$$

which tells us the benefit of picking action a in state s then switching to policy π, relative to the baseline return of always following π. Note that $A^\pi(s, a)$ can be both positive and negative, and $\mathbf{E}_{\pi(s|a)}\, A^\pi(s, a) = 0$ since

$$V^\pi(s) = \underset{\pi(a|s)}{\mathbf{E}} Q^\pi(s, a).$$

A fundamental property of value functions is that they satisfy recursive relationships similar to that which we have already established for the return (1.1). For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$\begin{aligned} V^\pi(s) &= \underset{\pi}{\mathbf{E}}(G_0 \mid s_0 = s) \\ &= \underset{\pi}{\mathbf{E}}(r_0 + \gamma G_1 \mid s_0 = s) && \text{(by (1.1))} \end{aligned}$$

$$= \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \mathop{\mathbf{E}}_{\pi}(G_1 \mid s_1 = s') \right)$$

$$= \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{\pi}(s') \right). \tag{1.2}$$

The equation (1.2) is called the *Bellman equation* for value function V^{π}. It expresses a relationship between the value of a state and the values of its successor states. Similarly, we have the Bellman equation for action-value function Q^{π} as

$$\begin{aligned} Q^{\pi}(s,a) &= \mathop{\mathbf{E}}_{\pi}(G_0 \mid s_0 = s, a_0 = a) \\ &= \mathop{\mathbf{E}}_{\pi}(r_0 + \gamma G_1 \mid s_0 = s, a_0 = a) && \text{(by (1.1))} \\ &= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \mathop{\mathbf{E}}_{\pi}(G_1 \mid s_1 = s') \\ &= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q^{\pi}(s',a'). \end{aligned}$$

Note that in the above discussion we overload the notation for density function p to also represent the probability measure $\mathbf{P}$ when talking about the transition model $p(s' \mid s,a)$. The intentions behind this is that we would like to use a unified representation for both discrete and continuous MDPs, and to emphasize the fact that the transition model p plays an important role in MDPs. One can distinguish whether p is a density function or a probability measure based on the context, especially according to how the probabilities are calculated ('$\sum$' or '$\int$').

1.3 Optimal value functions and policies

Suppose π^* is a policy such that $V^{\pi^*} \geq V^{\pi}$ for all $s \in \mathcal{S}$ and all policy π, then it is an *optimal policy*. There can be multiple optimal policies for the same MDP, but by definition their value functions must be the same, and are denoted by V^* and Q^*, respectively. We call V^* the *optimal (state-)value function*, and Q^* the *optimal action-value function*.

Intuitively, the value of a state under an optimal policy must equal the expected return for the best action from that state:

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}} Q^*(s,a) && (1.3) \\ &= \max_{a \in \mathcal{A}} \mathop{\mathbf{E}}_{\pi^*}(G_0 \mid s_0 = s, a_0 = a) \\ &= \max_{a \in \mathcal{A}} \mathop{\mathbf{E}}_{\pi^*}(r_0 + \gamma G_1 \mid s_0 = s, a_0 = a) \\ &= \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \mathop{\mathbf{E}}_{\pi^*}(G_1 \mid s_1 = s') \right) \\ &= \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s') \right), && (1.4) \end{aligned}$$

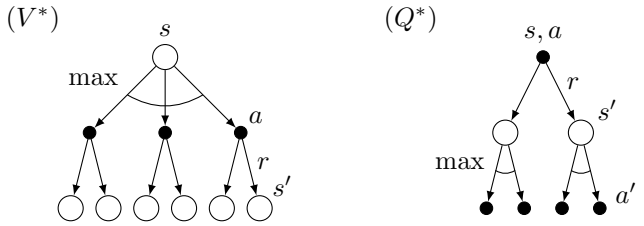

Figure 2 Backup diagrams for V^* and Q^*.

for all $s \in \mathcal{S}$. The last equation formulate the *Bellman optimality equation* for value function V^*. This derivation procedure also informs us about the Bellman optimality equation for action-value function Q^*:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \max_{a' \in \mathcal{A}} Q^*(s',a'), \tag{1.5}$$

for all $s \in \mathcal{S}, a \in \mathcal{A}$. Given a value function (V or Q), the discrepancy between the right- and left-hand sides of (1.4) and (1.5) are called *Bellman error* or *Bellman residual*. The backup diagrams in figure 2 show graphically the spans of future states and actions considered in the Bellman optimality equations for V^* and Q^*. For finite MDPs, the Bellman optimality equations (1.4) and (1.5) has a unique solution π^*. The Bellman optimality equation is actually a system of equations, one for each state, so if there are n states, then there are n equations in n unknowns. If the dynamics p of the environment are known, then in principle one can solve this system of equations for V^* and Q^* using any one of a variety of methods for solving systems of nonlinear equations.

Once one has V^*, it is relatively easy to determine an optimal policy π^*. For each state s, there will be one or more actions at which the maximum is obtained in the Bellman optimality equation. Any policy that assigns nonzero probability only to these actions is an optimal policy. This can be considered as an one-step search. Given the optimal value function V^*, then the actions that appear best after a one-step search will be optimal actions, *i.e.*,

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s') \right),$$

for all $s \in \mathcal{S}$. Another way of saying this is that any policy that is *greedy* with respect to the optimal evaluation function V^* is an optimal policy. The beauty of V^* is that if one uses it to evaluate the short-term consequences of actions — specifically, the one-step consequences — then a greedy policy is actually optimal in the long-term sense in which we are interested because V^* already takes into account the reward consequences of all possible future behavior.

Having Q^* makes choosing optimal actions even easier. With Q^*, the agent does not even have to do a one-step-ahead search: for any state s, it can simply find any

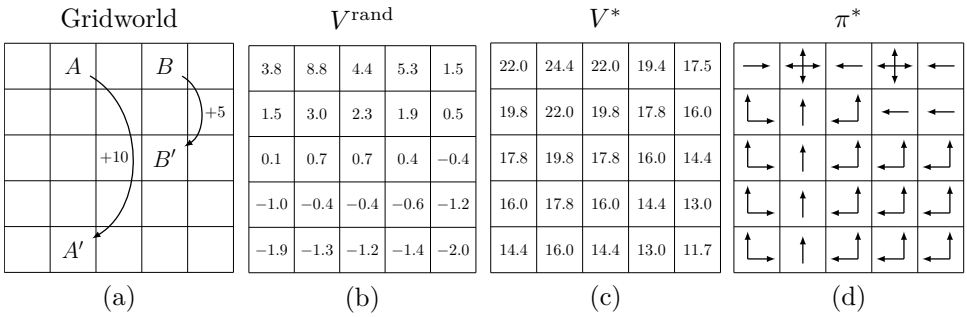

Figure 3 Gridworld example: exceptional reward dynamics (a) and state-value function for the equiprobable random policy (b). The figures (c) and (d) show the optimal value function V^* and optimal policy π^* of this gridworld example.

action that maximizes $Q^*(s, a)$, *i.e.*,

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s, a).$$

The action-value function effectively caches the results of all one-step-ahead searches. It provides the optimal expected long-term return as a value that is locally and immediately available for each state-action pair. Hence, at the cost of representing a function of state-action pairs, instead of just of states, the optimal action-value function allows optimal actions to be selected without having to know anything about possible successor states and their values, that is, without having to know anything about the environment's dynamics.

1.4 Example

We now show a simple example, to make concepts like value functions more concrete. Figure 3(a) shows a rectangular *gridworld* representation of a simple finite MDP. The cells of the grid correspond to the states of the environment. At each cell, four actions are possible: *up*, *down*, *left*, and *right*, which deterministically cause the agent to move one cell in the respective direction on the grid. Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of -1. Other actions result in a reward of 0, except those that move the agent out of the special states A and B. From state A, all four actions yield a reward of $+10$ and take the agent to A'. From state B, all actions yield a reward of $+5$ and take the agent to B'.

Suppose the agent selects all four actions with equal probability in all states. Figure 3(b) shows the value function V^π for this policy, for the discounted reward case with $\gamma = 0.9$. This value function was computed by solving the system of linear equations (1.2). Notice the negative values near the lower edge; these are the result of the high probability of hitting the edge of the grid there under the random policy. State A is the best state to be in under this policy. Note that A's

expected return is less than its immediate reward of 10, because from A the agent is taken to state A' from which it is likely to run into the edge of the grid. State B, on the other hand, is valued more than its immediate reward of 5, because from B the agent is taken to B' which has a positive value. From B' the expected penalty (negative reward) for possibly running into an edge is more than compensated for by the expected gain for possibly stumbling onto A or B.

Suppose we now solve the Bellman optimality equation for V^* in this gridworld task. Figures 3(c) and 3(d) show the optimal value function and the corresponding optimal policies, respectively. Where there are multiple arrows in a cell, all of the corresponding actions are optimal.

2 Dynamic programming

The term *dynamic programming* (DP) refers to a collection of iterative algorithms that can be used to compute optimal policies given a perfect model of the environment as an MDP. The key idea of DP is the use of value functions to organize and structure the search for good policies.

2.1 Policy iteration

2.1.1 Policy evaluation

First we consider how to compute the value function V^π for an arbitrary policy π. This called *policy evaluation* in the DP literature. We also refer to it as the *prediction problem*. According to (1.2), we have

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a \mid s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) V^\pi(s') \right),$$

for all $s \in \mathcal{S}$, where $\pi(a \mid s)$ is the probability of taking action a in state s under policy π. The existence and uniqueness of V^π is guaranteed as long as either $\gamma < 1$ or eventual termination is guaranteed for all states under the policy π.

If the environment's dynamics are completely known, then (1.2) is a system of $|\mathcal{S}|$ simultaneous linear equations in $|\mathcal{S}|$ unknowns ($V^\pi(s)$, for all $s \in \mathcal{S}$). In practice, we perfer to solve the system of linear equations with iterative methods. Consider a sequence of approximate value functions $V^{(0)}, V^{(1)}, V^{(2)}, \ldots$, with $V^{(i)} \colon \mathcal{S} \to \mathbf{R}$. The initial approximation $V^{(0)}$ is chosen arbitrarily (except that the terminal state, if any, must be given value 0), and each successive approximation is obtained by using the Bellman equation for value function (1.2) as an update rule:

$$V^{(i+1)}(s) = \sum_{a\in\mathcal{A}} \pi(a \mid s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) V^{(i)}(s') \right), \tag{2.1}$$

for all $s \in \mathcal{S}$. The sequence $V^{(0)}, V^{(1)}, \ldots, V^{(i)}, \ldots$ can be shown in general to converge to V^π as $i \to \infty$ under the same conditions that guarantee the existence of V^π. This algorithm is called *iterative policy evaluation*.

To write a sequential computer program to implement iterative policy evaluation as given by (2.1), we would have to use two arrays, one for the old values $V^{(i)}$, and one for the new values $V^{(i+1)}$. With two arrays, the new values can be computed one by one from the old values without the old values being changed. Alternatively, you could use one array and update the values 'in place', that is, with each new value immediately overwriting the old one. Then, depending on the order in which the states are updated, sometimes new values are used instead of old ones on the right-hand side of (2.1). This in-place algorithm also converges to V^π; in fact, it usually converges faster than the two-array version, as you might expect, because it uses new data as soon as they are available. We think of the updates as being done in a sweep through the state space. For the in-place algorithm, the order in which states have their values updated during the sweep has a significant influence on the rate of convergence. We usually have the in-place version in mind when we think of DP algorithms. A complete in-place version of iterative policy evaluation is shown in pseudocode in algorithm 1.

Algorithm 1 Iterative policy evaluation.

given the policy π to be evaluated.

initialize $V(s)$ for all $s \in \mathcal{S}$ arbitrarily, if s is not terminal, otherwise 0.

repeat

  for $s \in \mathcal{S}$,

    $V(s) \coloneqq \sum_{a\in\mathcal{A}} \pi(a \mid s) \big(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) V(s')\big)$.

until stop criterion reached.

2.1.2 Policy improvement

After obtaining the value function for some policy π from policy evaluation, we can find a new greedy policy π', according to

$$\pi'(s) = \operatorname*{argmax}_{a\in\mathcal{A}} Q^\pi(s,a) = \operatorname*{argmax}_{a\in\mathcal{A}} \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) V^\pi(s') \right), \tag{2.2}$$

for all $s \in \mathcal{S}$. The new greedy policy takes the action that looks best in the short term — after one step of lookahead — according to V^π. The process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy, is called *policy improvement*.

It can be easily shown that the new greedy policy π' is as good as, or better than the old policy π. Specifically, in the former case, the old policy π must be the optimal policy π^*. Suppose π' is as good as, but not better than π. Then $V^\pi = V^{\pi'}$, and from (2.2) it follows that for all $s \in \mathcal{S}$:

$$V^{\pi'}(s) = \max_{a\in\mathcal{A}} \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) V^{\pi'}(s') \right),$$

which is exactly the Bellman optimality equation. Therefore, $V^{\pi'}$ must be V^*, and both π and π' must be optimal policies.

2.1.3 Policy iteration

Once a policy π, has been improved using V^π to yield a better policy π', we can then compute $V^{\pi'}$ and improve it again to yield an even better π''. We can thus obtain a sequence of monotonically improving policies and value functions:

$$\pi^{(0)} \xrightarrow{\text{E}} V^{\pi^{(0)}} \xrightarrow{\text{I}} \pi^{(1)} \xrightarrow{\text{E}} V^{\pi^{(1)}} \xrightarrow{\text{I}} \pi^{(2)} \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} V^*,$$

where $\xrightarrow{\text{E}}$ denotes a policy evaluation and $\xrightarrow{\text{I}}$ denotes a policy improvement. Each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal). Because a finite MDP has only a finite number of deterministic policies, this process must converge to an optimal policy and the optimal value function in a finite number of iterations.

This way of finding an optimal policy is called *policy iteration*. A complete algorithm is given in algorithm 2. Note that each policy evaluation, itself an iterative computation, is started with the value function for the previous policy. This typically results in a great increase in the speed of convergence of policy evaluation (presumably since the value function changes little from one policy to the next).

Algorithm 2 POLICY ITERATION.

1. Initialization.

initialize $V(s) \in \mathbf{R}$ and $\pi(s) \in \mathcal{A}$ for all $s \in \mathcal{S}$.

2. Policy evaluation.

repeat

  for $s \in \mathcal{S}$,

    $V(s) \coloneqq \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V(s') \right)$.

until stop criterion reached.

3. Policy improvement.

  for $s \in \mathcal{S}$,

    $\pi(s) \coloneqq \operatorname{argmax}_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V(s') \right)$.

until policy is stable.

2.2 Value iteration

One drawback to policy iteration is that each of its iterations involves policy evaluation, which may itself be a protracted iterative computation requiring multiple sweeps through the state set. In fact, the policy evaluation step of policy iteration can be truncated in several ways without losing the convergence guarantees of policy iteration. One important special case is when policy evaluation is stopped after

just one sweep (one update of each state). This algorithm is called *value iteration* (VI). It can be written as a particularly simple update operation that combines the policy improvement and truncated policy evaluation steps:

$$V^{(i+1)}(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{(i)}(s') \right), \tag{2.3}$$

for all $s \in \mathcal{S}$. For arbitrary $V^{(0)}$, the sequence $V^{(0)}, V^{(1)}, \ldots, V^{(i)}, \ldots$ can be shown to converge to V^* under the same conditions that guarantee the existence of V^*.

Another way of understanding value iteration is by reference to the Bellman optimality equation (1.4). Note that value iteration is obtained simply by turning the Bellman optimality equation into an update rule. Also note how the value iteration update is identical to the policy evaluation update (2.1) except that it requires the maximum to be taken over all actions. A complete in-place version of value iteration is shown in pseudocode in Algorithm 3.

Algorithm 3 VALUE ITERATION.

initialize $V(s)$ for all $s \in \mathcal{S}$ arbitrarily, if s is not terminal, otherwise 0.

repeat

  for $s \in \mathcal{S}$,

    $V(s) \coloneqq \max_{a \in \mathcal{A}} \big(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V(s') \big)$.

until stop criterion reached.

output a deterministic policy $\pi \coloneqq \operatorname{argmax}_{a \in \mathcal{A}} \big(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V(s') \big)$.

Bibliography

Markov decision processes were known at least as early as the 1950s [Bel57]. A core body of research on Markov decision processes resulted from Ronald Howard's book [How60].

For a detailed mathematical analysis about some properties Markov decision processes and dynamical programming, one can refer to the book [Put14].

MDPs are closely connected with *reinforcement learning* (RL), one can refer to [Mur23, §35] for an overview about RL. More details can be found in the famous textbook [SB18].

References

[Bel57] R. Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[How60] R. A. Howard. *Dynamic Programming and Markov Processes.* John Wiley & Sons, 1960.

[Mur23] K. P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

[Put14] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.