Univ Gustave Eiffel - Cosys / Grettia

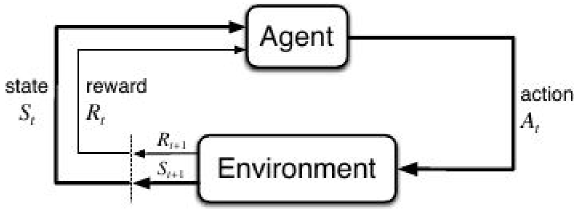
Reinforcement Learning and Optimal Control - Master 2 SIA

Finite Markov decision processes

Nadir Farhi
chargé de recherche, UGE - Cosys/Grettia
nadir.farhi@univ-eiffel.fr

Uni Eiffel - 19 september 2024

Introduction to MDPs



- MDPs are a classical formalization of sequential decision making
- Actions influence immediate rewards, subsequent states and future rewards
- MDPs involve the need to trade off immediate and delayed reward.
- The learner and decision maker is called the agent
- The agent interacts with the environment (everything outside the agent).

Introduction to MDPs

- We have discrete time steps, $t = 0, 1, 2, 3, \dots$
- At each step t, the agent receives a representation of the environment's state, S_t
- On that basis, the agent selects an action, A_t
- One step later, the agent receives a numerical reward, R_{t+1}
- and finds itself in a new state, S_{t+1}
- We the have a sequence or trajectory :

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$$

- $\mathcal{S}$: set of states. (random variables)
- $\mathcal{A}$: set of actions.
- $\mathcal{R}$: set of rewards. (random variables)

In finite DMPs, $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ are finte.

Dynamics of MDPs

- The function p : R $\times$ S $\times$ S $\times$ A $\rightarrow$ [0, 1] defines the dynamics of a MDP :

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\},$$

- p is a conditional probability distribution satisfying :

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s).$$

- Markov property : S_t summarizes all the inforation, past to t.
- State transition probabilities :

$$p(s' \mid s, a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a).$$

- The expected reward for a state-action pair (s, a) :

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a),$$

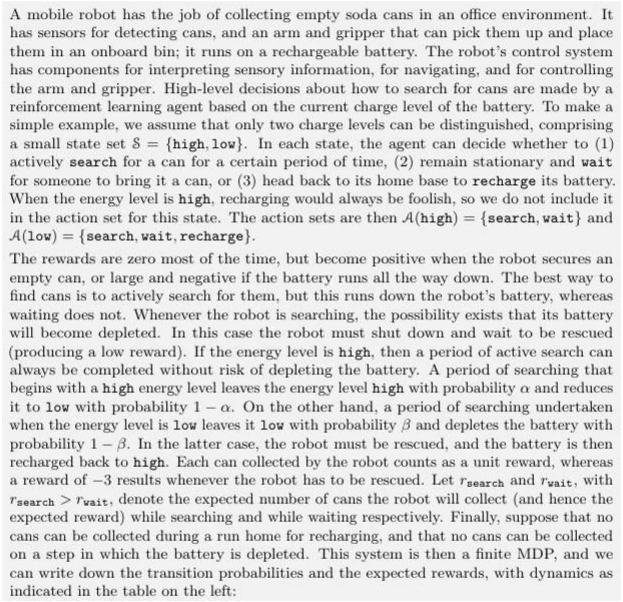
- The expected reward for a state-action-next-state (s, a, s') :

$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}.$$

Agent - environment

- Anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment.

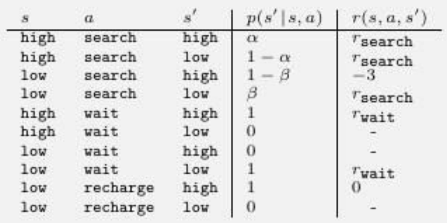
Recycling Robot



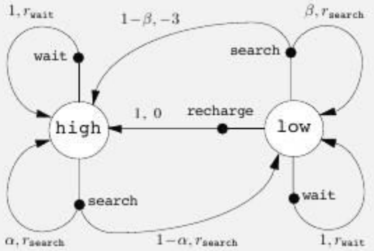
A mobile robot has the job of collecting empty soda cans in an office environment. It has sensors for detecting cans, and an arm and gripper that can pick them up and place them in an onboard bin; it runs on a rechargeable battery. The robot's control system has components for interpreting sensory information, for navigating, and for controlling the arm and gripper. High-level decisions about how to search for cans are made by a reinforcement learning agent based on the current charge level of the battery. To make a simple example, we assume that only two charge levels can be distinguished, comprising a small state set $\mathcal{S} = \{\texttt{high}, \texttt{low}\}$. In each state, the agent can decide whether to (1) actively **search** for a can for a certain period of time, (2) remain stationary and **wait** for someone to bring it a can, or (3) head back to its home base to **recharge** its battery. When the energy level is **high**, recharging would always be foolish, so we do not include it in the action set for this state. The action sets are then $\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$ and $\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$.

The rewards are zero most of the time, but become positive when the robot secures an empty can, or large and negative if the battery runs all the way down. The best way to find cans is to actively search for them, but this runs down the robot's battery, whereas waiting does not. Whenever the robot is searching, the possibility exists that its battery will become depleted. In this case the robot must shut down and wait to be rescued (producing a low reward). If the energy level is **high**, then a period of active search can always be completed without risk of depleting the battery. A period of searching that begins with a **high** energy level leaves the energy level **high** with probability α and reduces it to **low** with probability $1 - \alpha$. On the other hand, a period of searching undertaken when the energy level is **low** leaves it **low** with probability β and depletes the battery with probability $1 - \beta$. In the latter case, the robot must be rescued, and the battery is then recharged back to **high**. Each can collected by the robot counts as a unit reward, whereas a reward of -3 results whenever the robot has to be rescued. Let $r_{\texttt{search}}$ and $r_{\texttt{wait}}$, with $r_{\texttt{search}} > r_{\texttt{wait}}$, denote the expected number of cans the robot will collect (and hence the expected reward) while searching and while waiting respectively. Finally, suppose that no cans can be collected during a run home for recharging, and that no cans can be collected on a step in which the battery is depleted. This system is then a finite MDP, and we can write down the transition probabilities and the expected rewards, with dynamics as indicated in the table on the left:

Recycling Robot



s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-



Rewards

- The agent maximizes cumulative reward in the long run.
- Reward hypothesis :

> That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

Examples :

1. To make a robot learn to walk, one can provid reward on each time step proportional to the robot's forward motion.
2. To make a robot learn how to escape from a maze, the reward is often (-1) for every time step that passes prior to escape ; this encourages the agent to escape as quickly as possible.
3. To make a robot learn to find and collect empty soda cans for recycling, one might give it a reward of zero most of the time, and then a reward of +1 for each can collected.
4. One might also want to give the robot negative rewards when it bumps into things or when somebody yells at it.
5. For an agent to learn to play checkers or chess, the natural rewards are +1 for winning, 1 for losing, and 0 for drawing and for all nonterminal positions.

Returns and Episodes

- In the simplest case, the return is the sum of rewards :

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

- *T* is a final time step.
- For applications when the agent - environment interaction breaks naturally into subsequences called episodes.
- E.g. plays of a game, trips through a maze, etc.
- Each episode ends in a special state called the terminal state.
- The next episode begins independently of how the previous one ended.
- Thus the episodes can all be considered to end in the same terminal state, with different rewards.

Returns and continuing tasks

- There are cases where the agent - environment interaction does not break naturally into identifiable episodes, but goes on continually without limit.
- E.g. control a robot with a long life span.
- We call them contiuing tasks.
- Discounting ($0 \leq \gamma \leq 1$ is the discount rate) :

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

- The discount rate determines the present value of future rewards.

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma\big(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\big) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

- If $\gamma < 1$ and the reward is constant, then the return is finite.
  e.g. if the reward is a constant +1, then the return is

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}.$$

Policies and Value Functions (1/3)

- Value function is a function of states that estimates how good it is for the agent to be in a given state.
- The estimation is in terms of expected return (future rewards that can be expected).
- The return depends also on the actions taken by the agent.
- Accordingly, value functions are defined with respect to particular ways of acting, called policies.

  Value function : $v_\pi(s)$
- A policy is a mapping from states to probabilities of selecting each possible action.
- $\pi(a \mid s)$: the probability that $A_t = a$ if $S_t = s$.
- Reinforcement learning methods specify how the agent's policy is changed as a result of its experience.

Policies and Value Functions (2/3)

- The value function of a state *s* under a policy π is the expected return when
  - starting in *s*
  - and following π thereafter.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S},$$

- The action-value function of a state *s* and action *a*, under policy π, is the expected return when
  - starting from *s*,
  - taking the action *a*,
  - and thereafter following policy π.

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right].$$

Policies and Value Functions (3/3)

- v_π and q_π can be estimated from experience.
- For example, if an agent follows policy π and maintains an average, for each state encountered, of the actual returns that have followed that state, then the average will converge to the state's value, $v_\pi(s)$, as the number of times that state is encountered approaches infinity.
- If separate averages are kept for each action taken in each state, then these averages will similarly converge to the action values, $q_\pi(s, a)$.
- We call estimation methods of this kind *Monte Carlo methods*, because they involve averaging over many random samples of actual returns.

The Bellman equation

- Recall that the discount rate determines the present value of future rewards.

$$
\begin{aligned}
G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\
&= R_{t+1} + \gamma\big(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\big) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

- For any policy π and any state s :

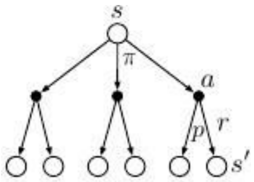
$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\Big[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\Big] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},
\end{aligned}
$$

- Expected value : a sum over all values of the three variables, a, s' , and r.
- For each triple, we compute its probability, $\pi(a \mid s)p(s', r \mid s, a)$
- Which weights the quantity in brackets, then sum over all possibilities to get an expected value.

The Bellman equation

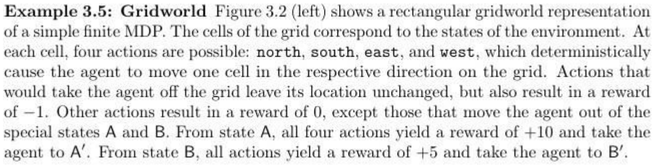
- For any policy π and any state s, we have the linear system :

$$\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \Big] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},
\end{aligned}$$

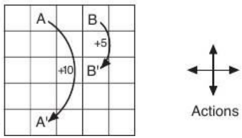


Backup diagram for v_π

Policies and Value Functions



Example 3.5: Gridworld Figure 3.2 (left) 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: `north`, `south`, `east`, and `west`, 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$'$.



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Exercise : Policy : Take actions with equal probability in all states ($\gamma = 0.9$).

1. Compute by soving the system of linear equations (slide 15)
2. Compute by Monte Carlo simulation

Optimality

Optimal Policies and Optimal Value Functions

- Solving a RL problem means finding a policy that achieves a lot of reward over the long run.
- $\pi \geq \pi' \quad \Leftrightarrow \quad v_\pi(s) \geq v_{\pi'}(s), \forall s \in S.$
- π_* : optimal policy : it is better than or equal to all oher policies.
- Optimal policies share the same state-value function :

$$v_*(s) := \max_\pi v_\pi(s), \forall s \in S, \qquad \text{optimal state-value function.}$$

- Optimal policies share also the same action-value function :

$$q_*(s,a) := \max_\pi q_\pi(s,a), \forall s \in S, a \in A, \qquad \text{optimal action-value function.}$$

- $q_*(s,a)$ gives the expected return for taking action a in state s and thereafter following an optimal policy.

$$q_*(s,a) = \mathrm{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\right].$$

The Bellman optimality equation for v_*

- Recall the Bellman equation for v_π :

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \;\; r + \gamma v_\pi(s') \;\; .$$

- v_* satisfies the Bellman equation :

$$\begin{aligned} v_*(s) &= \max_{a\in\mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r \mid s,a)\big[r + \gamma v_*(s')\big]. \end{aligned}$$

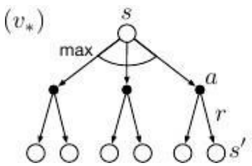
- The value of a state under an optimal policy must equal the expected return for the best action from that state.

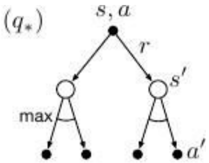
The Bellman optimality equation for q_*

- The Bellman optimality equation for q_* :

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a') \;\Big|\; S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big]. \end{aligned}$$

- the backup diagrams for v_* and q_* :





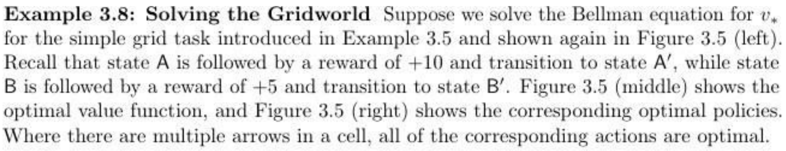
Retrieve the optimal policies from v_*

- For finite MDPs, the Bellman optimality equation for v_* has a unique solution.
- The Bellman optimality equation is a system of non-linear equations, one for each state.
- If the dynamics p of the environment are known, we can solve this system using any of the existing methods.
- Once v_* is obtained, the optimal policies are the ones at which the maximum is obtained in the Bellman optimalityequation.
- 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.
- Therefore, a one-step-ahead search yields the long-term optimal actions.

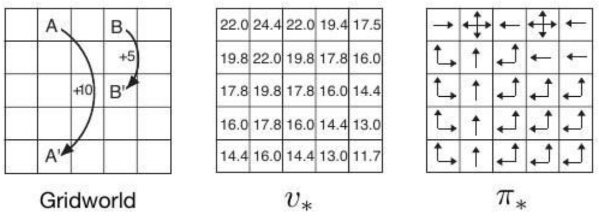
Retrieve the optimal policies from q_*

- Having q_* the agent does not even have to do a one-step-ahead search.
- For any state s, it can simply find any action that maximizes $q_*(s, a)$.
- q_* effectively caches the results of all one-step-ahead searches.
- q_* 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.

Solving the Gridworld



Example 3.8: Solving the Gridworld Suppose we solve the Bellman equation for v_* for the simple grid task introduced in Example 3.5 and shown again in Figure 3.5 (left). Recall that state A is followed by a reward of +10 and transition to state A′, while state B is followed by a reward of +5 and transition to state B′. Figure 3.5 (middle) shows the optimal value function, and Figure 3.5 (right) shows the corresponding optimal policies. Where there are multiple arrows in a cell, all of the corresponding actions are optimal.



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7