Markov Decision processes

Reinforcement learning – LM Artificial Iintelligence
(2022-23)

Alberto Castellini
University of Verona

Summary

- Introduction
- The Agent-Environment Interface
- Goals and Rewards
- Returns and Episodes
- Policies and Value Functions
- Optimal Policies and Optimal Value Functions
- Optimality and Approximation

Introduction

Introduction

- **Markov Decision Process (MDP)**: formalization of sequential decision making problem

- **Actions influence** not only immediate **rewards** but also **subsequent situations** (delayed reward)

- Trade-off immediate and delayed reward

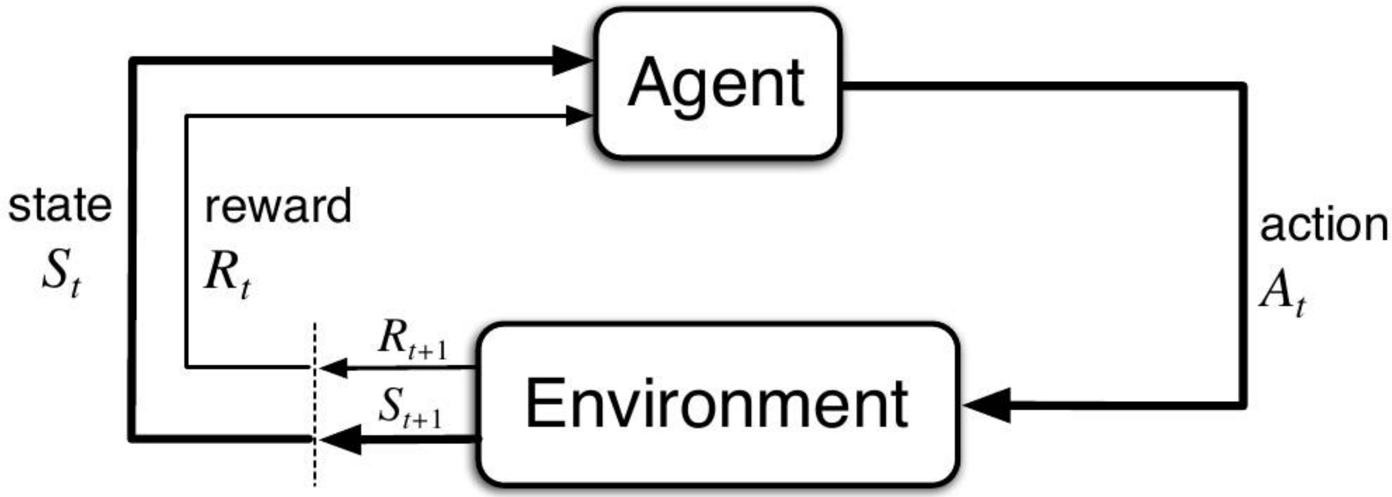
The Agent-Environment Interface

The Agent-Environment Interface

- **MDP**: formal framework representing problems of **learning from interaction** to achieve a goal

- **Agent**: the learner

- **Environment**: everything outside the agent



Markov Decision Process (notation)

The **main elements** of an **MDP** are:

- **States**: $S_t \in S$ (where t = 0, 1, 2, 3 … represent time steps)

- **Actions**: $A_t \in A$

- **Rewards**: $R_{t+1} \in R \subset \mathbb{R}$

- **Dynamics** function:

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

$$s, s' \in S, r \in R, a \in A(s)$$

- *p* specifies a probability distribution

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

Markov Decision Processes (notation)

From *p* we can compute:

- **State-transition probabilities**

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

- **Expected rewards**

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

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

Markov property: The state must include all information about all aspects of the past agent-environment interaction

Trajectory: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$

The "art" of generating MDPs

- The **MDP framework** is **abstract** and **flexible** and can be applied to **different problems** in different **ways** (e.g., low/high level actions)

- **High/Low level decisions:** in a complex robot many agents may be operating at once (e.g., high level decisions can form part of the state for lower-level decisions)

- **Boundary between agent and environment** is typically not the same as the physical boundary (anything that cannot be changed arbitrarily by the agent is considered part of the environment)

- The agent may **know** everything about the **environment** but still face a difficult RL task (e.g., Rubik's cube)

Example: Recicling Robot

States:
S={high, low} (charge levels)

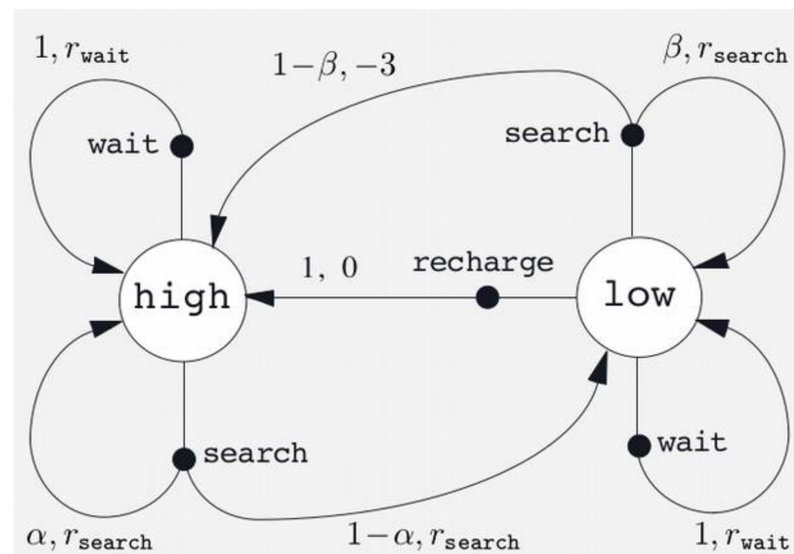
Actions:
A(low)={search, wait, recharge}
A(high)={search, wait}

Dynamics/Rewards:

Transition probabilities | Expected rewards

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	-

Transition graph



Goals and Rewards

Goals and rewards

- In RL the **goal is formalized in terms of reward**
- The agent's goal is to **maximize the total amount of reward**
- Not immediate reward but **cumulative reward**

- **Reward hypothesis:** 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**)

- The use of reward signal to formalize the goal is one of the most **distinctive features of RL**

- Examples: learning to walk, learning to escape from a maze, learning to play checkers

- Reward is a way to say the agent **what** to do, **not how**

Returns and Episodes

Returns and episodes

Episodic tasks: applications in which there is a natural notion of final step (e.g., the plays of a game)

- Each **episode** ends in a state called **terminal state**, followed by a reset to a standard **starting state**
- At time *t* the agent seeks to maximize the **expected return**
- The **return** is the sum of rewards **until the final step T**

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

Continuing tasks: the agent-environment interactions do not break naturally in epiodes but go on continuously without limit ($T = \infty$)

- The agent maximizes the **discounted return**

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

where γ is the discount factor, with $0 \leq \gamma \leq 1$

Returns and episodes

- If $\gamma < 1$ the **infinite sum** of the expected reward has a **finite value** as long as the **reward sequence is bounded**
- If $\gamma = 0$ the agent is **myopic** (i.e., considers only immediate reward). In general this reduces access to future rewards, with reduced return

- The **(dicounted) return** can be written in a **recursive** way:

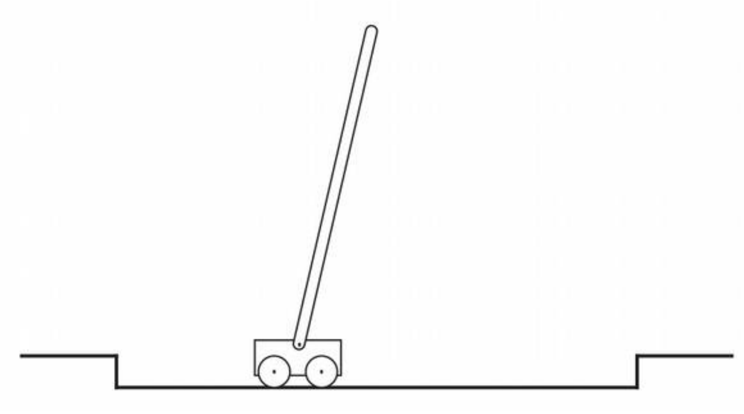
$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \right) \\ &= R_{t+1} + \gamma \boxed{G_{t+1}} \end{aligned}$$

- **Note:** if $\gamma < 1$, although the **expected return** is a **sum of infinite terms**, it is still **finite** if the reward is nonzero and constant.
- E.g., if reward is always 1 and $\gamma < 1$ then

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma} \quad \text{(geometric series)}$$

Example: Pole Balancing

- **Objective:** to apply forces to a cart moving along a track to keep a pole hinged to the cart from falling over



Episodic task:

- **Episodes** are the repeated attempts to balance the pole
- **Reward**: +1 for every time step in which failure did not occur
- **Return**: number of steps until failure
- **Problem: successful balancing forever → infinite reward**

Continuing task (using discounting):

- **Reward**: -1 on each failure, 0 at all other times
- **Return** at each step: $-\gamma^K$ where K is the number of time steps before failure

Policies and Value Functions

Policies and Value Functions

- Almost all RL algorithms involve estimating **value functions**, i.e., functions of state (or state-action pairs) that estimate *how good* it is for the agent to be in a given state (in terms of **expected return**)

- Since the future expected return depends on what actions the agent will take, **value functions** are defined w.r.t. particular ways of acting, called **policies**

- **Policy:** is a **mapping from states to probabilities of selecting each possible action**. Symbol $\pi(a|s)$ indicates the probability that action *a* is selected from state *s*.

- **RL algorithms** specify how the **agent's policy** is **changed** as a result of its **experience**.

Policies and Value Functions

- The **state-value function** of a state *s* under a policy π, denoted by $v_\pi(s)$ is the expected return when starting in *s* and following π thereafter

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

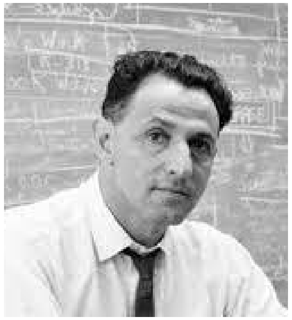
- The **action-value function** of taking action *a* in state *s* under a policy π denoted by $q_\pi(s,a)$ is the expected return starting from state *s*, taking action *a*, and therefore following π

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

- Both **state** and **action value functions** can be **estimated** from **experience**

Bellman Equation

- Fundamental **property** of **value functions**: they satisfy the following recursive relationships (**Bellman Equation**)

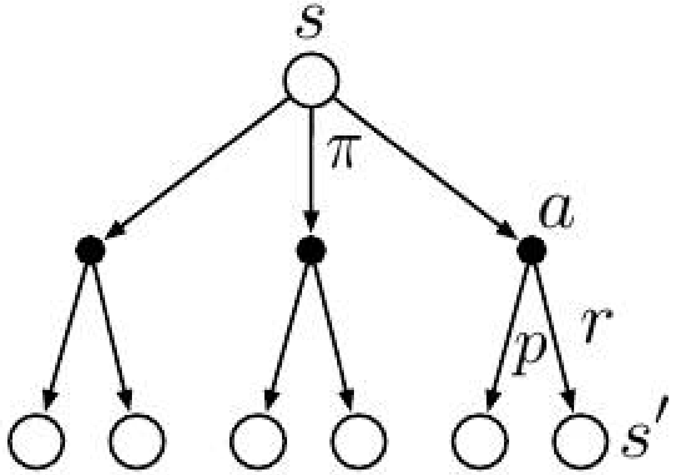


Richard Bellman

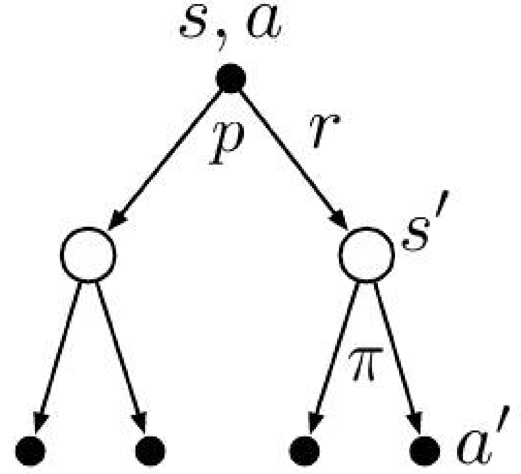
$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \Big] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \Big[r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

- Last expression: sum over all values of *a*, *s'* and *r*.
- For each triple, we
  - compute its **probability** $\pi(a|s)p(s', r | s, a)$,
  - **weight** the quantity in brackets by this probability,
  - **sum** over all possibilities to get an expected value.

Bellman Equation: backup diagrams



Backup diagram for v_π

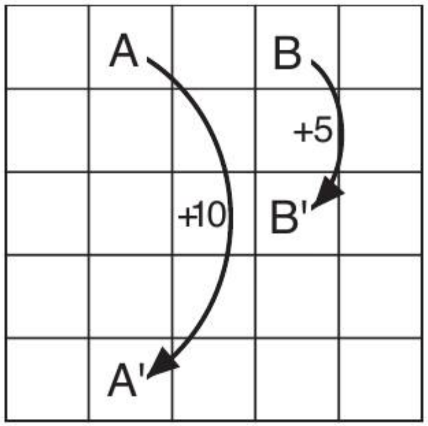


q_π backup diagram

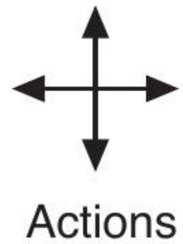
Bellman Equation: observations

- The value function $v_\pi(s)$ is the unique solution to its Bellman equation
- The Bellman equation forms the basis of a number of ways to compute, approximate and learn $v_\pi(s)$ (backup diagrams)
- The Bellman equation is actually a **system of equations (one for each state)** → Method for solving non-linear equations
- **Backup operators** transfer value information back to a state (or state-action pair) from its successor state (or state-action pair).

Example: Gridworld



Exceptional reward dynamics



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

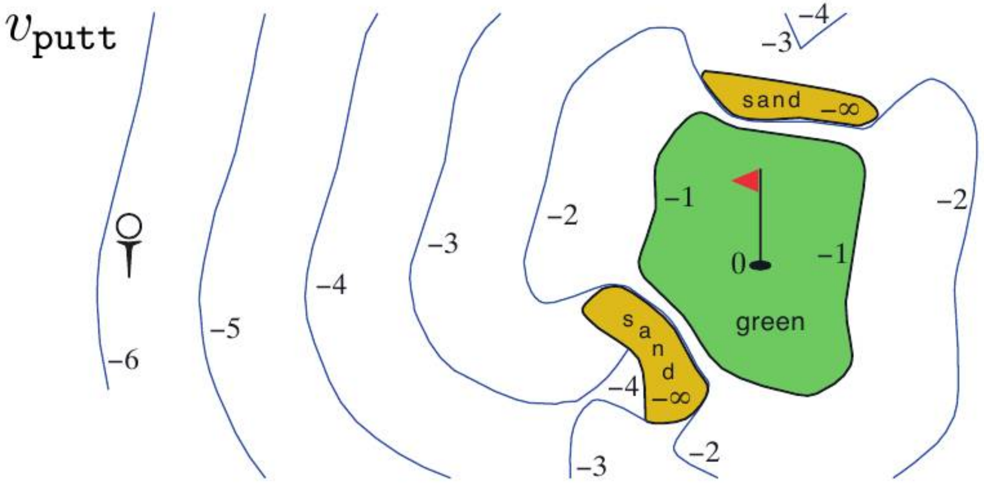
$v_\pi(s)$

Value function computed by solving the Bellman equation

- **Actions**: N, S, E, W (1 cell in the direction, deterministically)
- **Rewards**: exceptional rewards (A→A', all actions from A, +10; B → B', all actions from B, +5); off the grid (-1); other actions (0)
- **Policy**: uniformly random action selection in all states
- **Discount factor**: 0.9

Example: Golf

- **Reward**: -1 for each stroke
- **State**: location of the ball
- **State value**: negative number of strokes to the hole from the location
- **Actions**: which club we select **(putter or driver)**



Value function for a policy that **always selects a putter**

Optimal Policies and Optimal Value Functions

Optimal Policies and Optimal Value Functions

- **Solving RL tasks** means finding a policy that achieves large reward over long runs → **Finding an optimal policy**

- Value functions define a **partial ordering over policies**

- **A policy π is defined to be better than or equal to a policy π' if its expected return (i.e., value) is greater than or equal to that of π for all states,** namely

$$\pi \geq \pi' \Leftrightarrow v_{\pi}(s) \geq v_{\pi'}(s), \forall s \in S$$

- There is always **at least one** policy that is better than or equal to all other policies. This is an **optimal policy** (notation π_*).

- All optimal policies share the same **optimal state-value function**

$$v_*(s) \doteq \max_{\pi} v_{\pi}(s)$$

Optimal Policies and Optimal Value Functions

- **Optimal policies** also share the same **optimal action-value function**

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a)$$

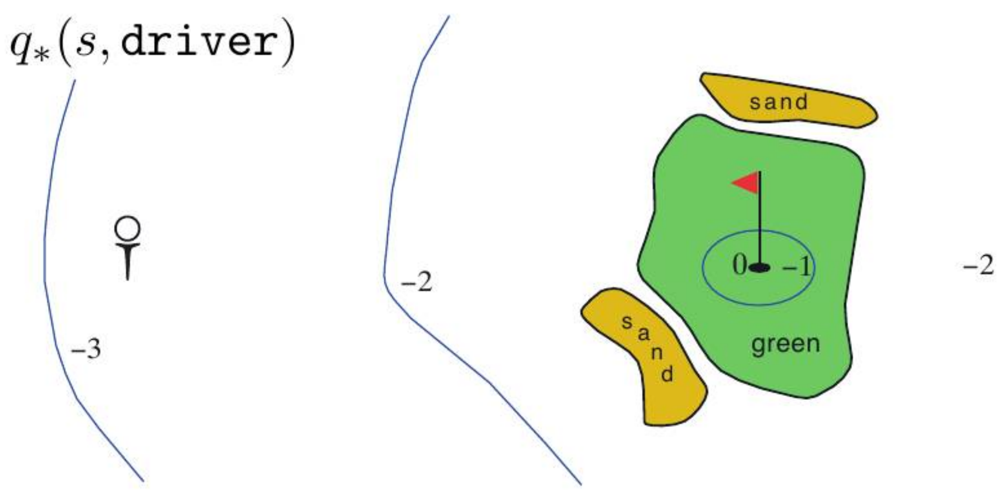
for all $s \in S$ and $a \in A$

- We can write q_* **in terms of** v_* as

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

- **Example: Optimal Value Function for Golf**

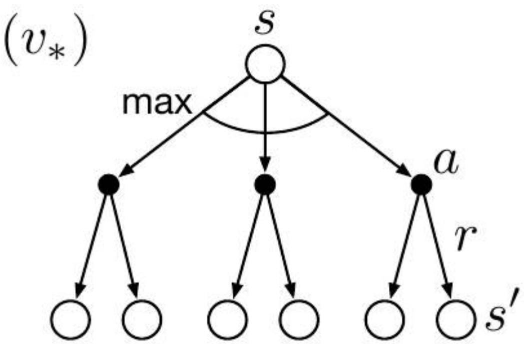
Value of each **state** if we **first** play a stroke with the **driver** and afterward optimally select either a driver or a putter



Optimal Policies and Optimal Value Functions

- **Bellman optimality equation for** v_*
- The **value** of a **state** under an **optimal policy** must equal the expected return for the **best action** from that state:

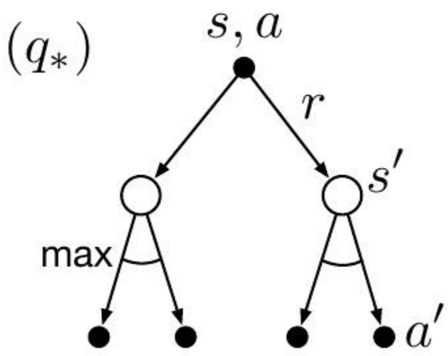
$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \boxed{\max_a}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



Optimal Policies and Optimal Value Functions

- **Bellman optimality equation for q_***

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

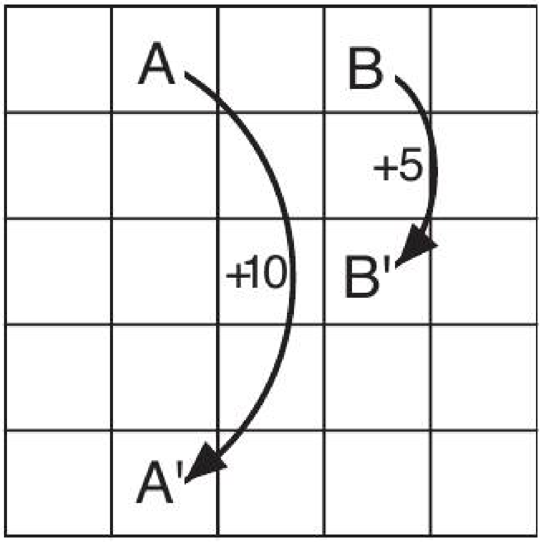


Optimal Policies and Optimal Value Functions

- **Given** v_* **,** the **optimal policy for a state *s*** is achieved selecting an action by which the **maximum** is obtained in the Bellman optimality equation. Any policy that assigns nonzero probability only to these actions is optimal (i.e., **one-step search**, **greedy policy w.r.t.** v_*)

- **Given** q_* , the **optimal policy for a state *s*** is achieved simply selecting an action that maximizes $q_*(s,a)$ (i.e., **zero-step search**, **greedy policy w.r.t.** q_*)

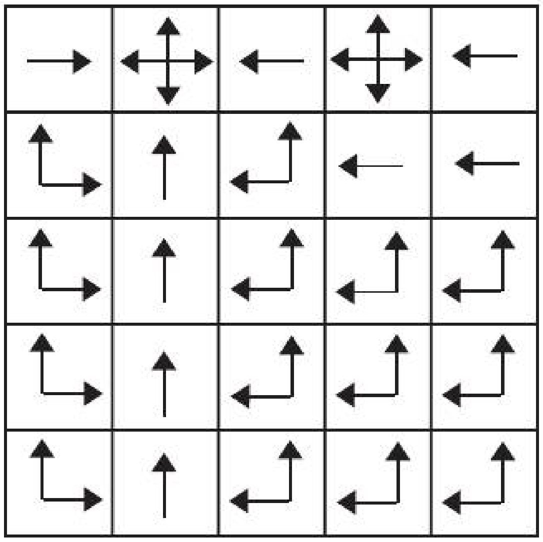
Example: solving the Gridworld



Gridworld

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

v_*



π_*

Figure 3.5: Optimal solutions to the gridworld example.

Homework: check the correctness of the optimal values and policy of this example using the codes developed in the next lab exercise (i.e., value and policy iteration)

Example: Recycling Robot

Bellman optimality equations for the recycling robot

$$\begin{aligned} v_*(\mathtt{h}) &= \max\left\{ \begin{array}{l} p(\mathtt{h}|\mathtt{h},\mathtt{s})[r(\mathtt{h},\mathtt{s},\mathtt{h})+\gamma v_*(\mathtt{h})]+p(\mathtt{l}|\mathtt{h},\mathtt{s})[r(\mathtt{h},\mathtt{s},\mathtt{l})+\gamma v_*(\mathtt{l})], \\ p(\mathtt{h}|\mathtt{h},\mathtt{w})[r(\mathtt{h},\mathtt{w},\mathtt{h})+\gamma v_*(\mathtt{h})]+p(\mathtt{l}|\mathtt{h},\mathtt{w})[r(\mathtt{h},\mathtt{w},\mathtt{l})+\gamma v_*(\mathtt{l})] \end{array} \right. \\ &= \max\left\{ \begin{array}{l} \alpha[r_{\mathtt{s}}+\gamma v_*(\mathtt{h})]+(1-\alpha)[r_{\mathtt{s}}+\gamma v_*(\mathtt{l})], \\ 1[r_{\mathtt{w}}+\gamma v_*(\mathtt{h})]+0[r_{\mathtt{w}}+\gamma v_*(\mathtt{l})] \end{array} \right\} \\ &= \max\left\{ \begin{array}{l} r_{\mathtt{s}}+\gamma[\alpha v_*(\mathtt{h})+(1-\alpha)v_*(\mathtt{l})], \\ r_{\mathtt{w}}+\gamma v_*(\mathtt{h}) \end{array} \right\}. \end{aligned}$$

$$v_*(\mathtt{l}) = \max\left\{ \begin{array}{l} \beta r_{\mathtt{s}} - 3(1-\beta)+\gamma[(1-\beta)v_*(\mathtt{h})+\beta v_*(\mathtt{l})], \\ r_{\mathtt{w}}+\gamma v_*(\mathtt{l}), \\ \gamma v_*(\mathtt{h}) \end{array} \right\}.$$

Homework: check the correctness of the formulas of this example

Problems of the Bellman optimality equation

- Bellman optimality equation is similar to an **exhaustive search**, solving it needs to invert a matrix with dimension equal to the number of states (i.e., **complexity O(S^3)**) → **rarely useful in real-world problems**
  - E.g., Backgammon has 10^{20} **states**. It would take thousands of years in modern computers

- **Assumptions** for using Bellman optimality equation to solve an MDP:
  - We accurately **know the dynamics of the environment**
  - We have enough computational resources
  - Markov property

- **Other decision-making methods** are ways to **approximatively solve** the **Bellman optimality equation**
  - **Heuristic search methods**: expand up to some depth, forming a tree of possibilities and evaluate leaves by heuristics (e.g., A*)
  - **Dynamic programming**

Optimality and Approximation

Optimality and approximation

- **Problem:** Optimal policies can be generated only with extreme computational cost

- **Problem:** Even with accurate models of the environment's dynamics it is not always possible to solve Bellman optimality equation

- In problems with **small state and action spaces** it is possible to represent **policy** and **value function approximations** by **arrays/tables**: **tabular methods (Part I of SutBar)**

- In other cases **parametrized function representations** are used (e.g., neural networks) **(Part II of SutBar)**

Categorization of RL algorithms

Value based

- Value function: **yes**
- Policy: **no (implicit)**

Policy based

- Value function: **no**
- Policy: **yes**

Actor-critic

- Value function: **yes**
- Policy: **yes**

Model based

- Dynamics model (i.e., transition and reward): **yes**

Model free

- Dynamics model (i.e., transition and reward): **no**

References

- R. S. Sutton, A. G. Barto. Reinforcement learning, An Introduction. Second edition. Chapter 3