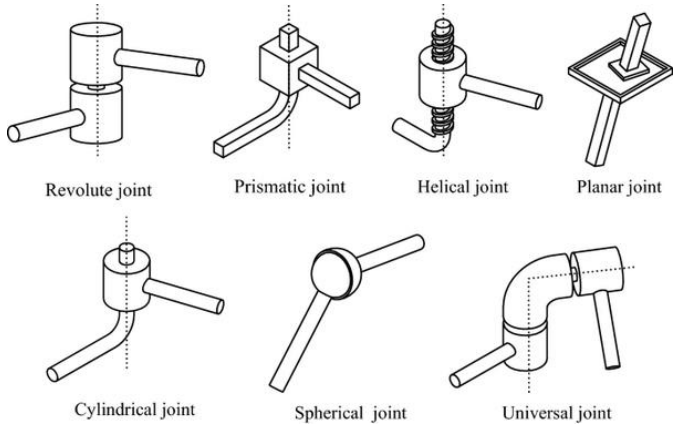
Robot Learning

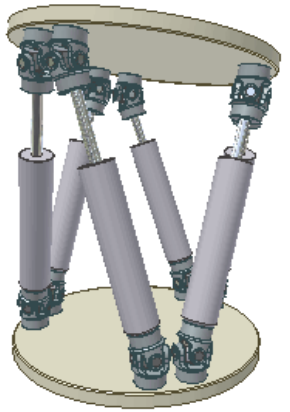
Reinforcement learning

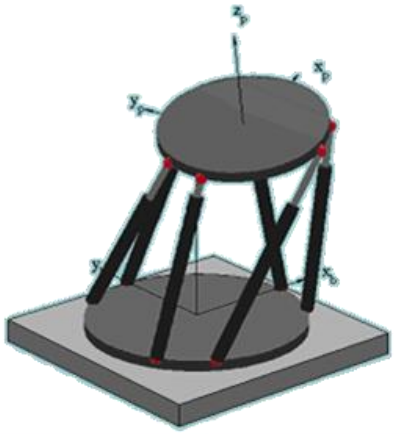
USC University of Southern California

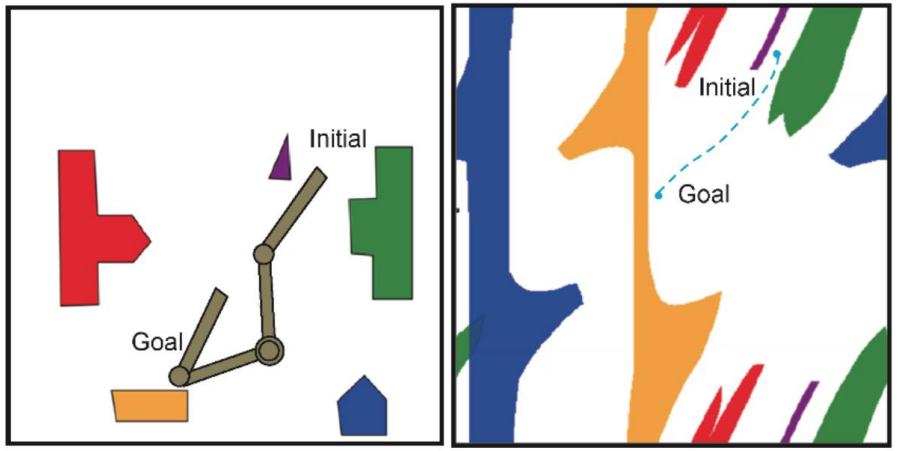
LiraLab

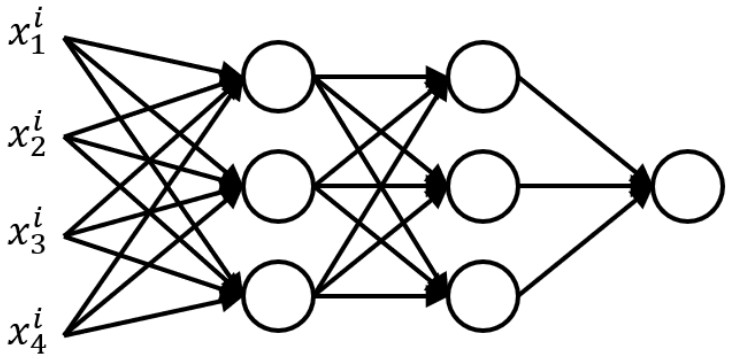
So far…


Revolute joint
Prismatic joint
Helical joint
Planar joint
Cylindrical joint
Spherical joint
Universal joint

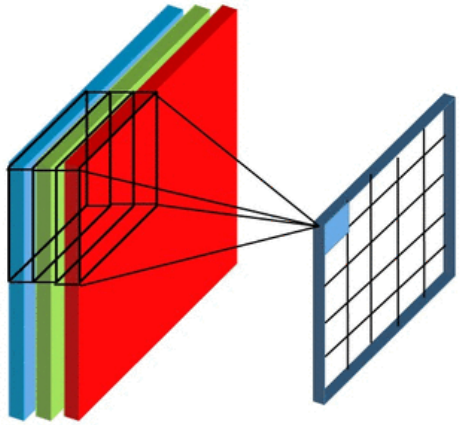


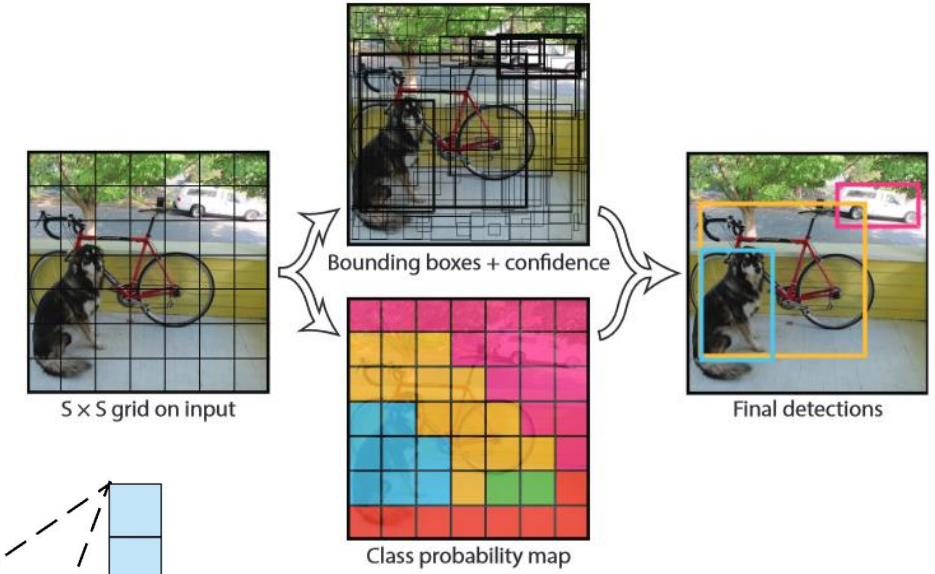


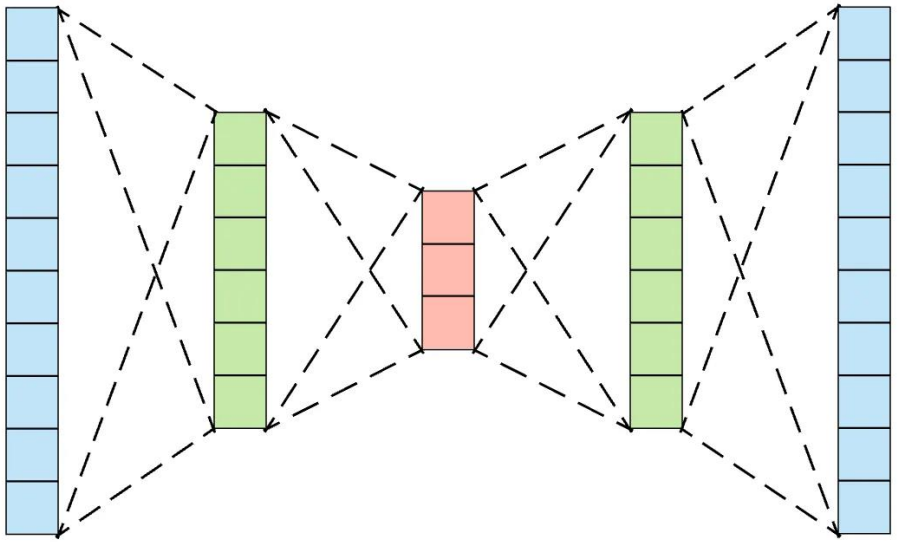

Initial
Goal
Initial
Goal


x_1^i
x_2^i
x_3^i
x_4^i

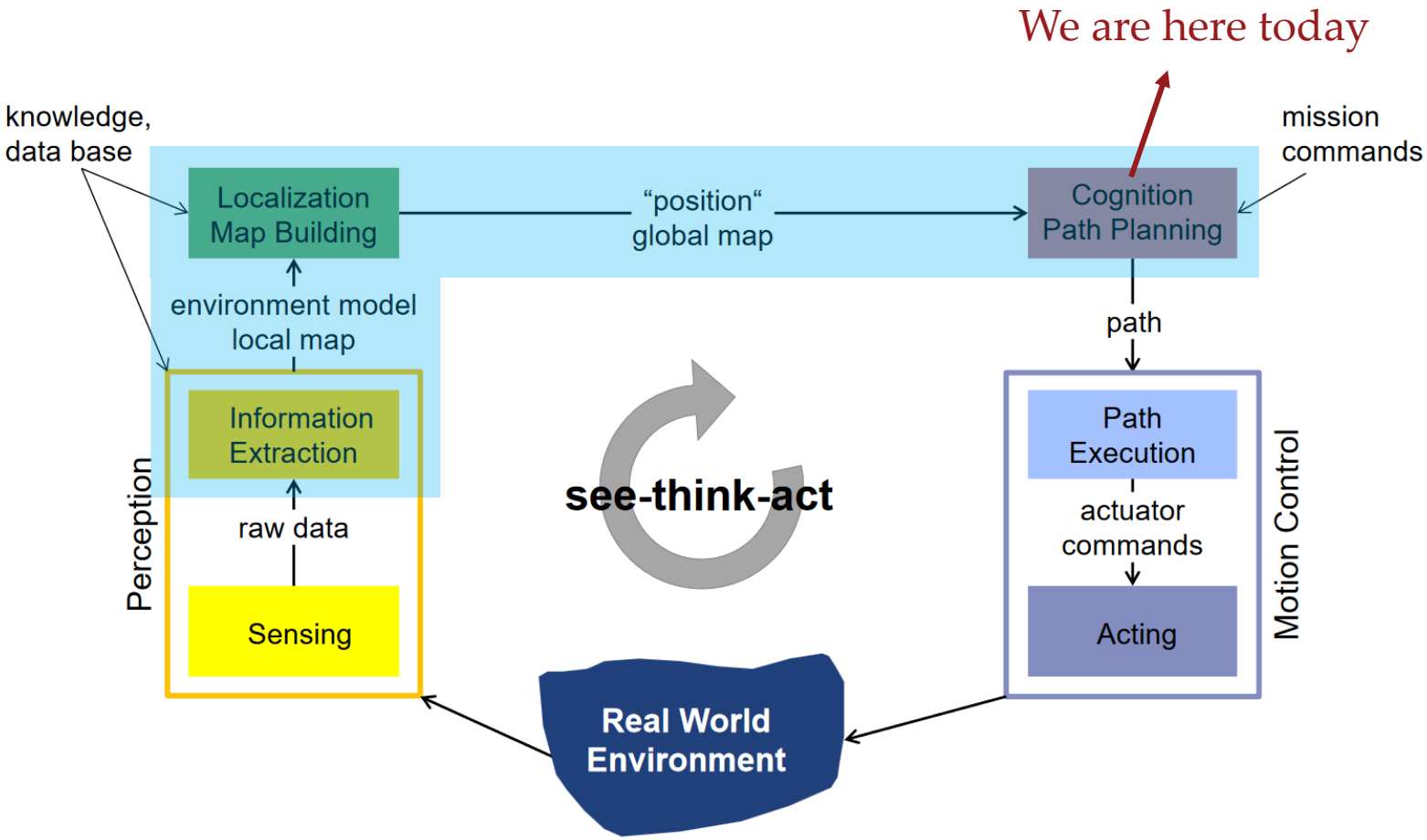
So far…




S × S grid on input
Bounding boxes + confidence
Class probability map
Final detections



Robot learning



Autonomous Mobile Robots
Siegwart et al.

Today…

- Dynamic programming in deterministic systems
- Dynamic programming in stochastic systems
- Markov decision processes
- Value/policy iteration

Decision making in deterministic systems

State: $s_t \in \boldsymbol{S}$

Action: $a_t \in \boldsymbol{\mathcal{A}}(s_t)$

Transition: $s_{t+1} = f_t(s_t, a_t)$

Decision making in deterministic systems

State: $s_t \in \boldsymbol{S}$

Action: $a_t \in \boldsymbol{\mathcal{A}}(s_t)$

Transition: $s_{t+1} = f_t(s_t, a_t)$

Total reward:

$$J(s_0; a_0, \dots, a_{T-1}) = r_T(s_T) + \sum_{t=0}^{T-1} r_t(s_t, a_t)$$

Traditionally, it is cost, not reward. In general, different communities use different notation and conventions.

Finite horizon

Decision making in deterministic systems

State: $s_t \in \boldsymbol{S}$

Action: $a_t \in \boldsymbol{\mathcal{A}}(s_t)$

Transition: $s_{t+1} = f_t(s_t, a_t)$

Total reward:

$$J(s_0; a_0, \dots, a_{T-1}) = r_T(s_T) + \sum_{t=0}^{T-1} r_t(s_t, a_t)$$

Decision making problem:

$$J^*(s_0) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t), t=0,1,\dots,T-1} J(s_0; a_0, \dots, a_{T-1})$$

Decision making in deterministic systems

State: $s_t \in \boldsymbol{S}$

Action: $a_t \in \boldsymbol{\mathcal{A}}(s_t)$

Transition: $s_{t+1} = f_t(s_t, a_t)$

Discrete-time assumption

Total reward:

Additive rewards assumption

$$J(s_0; a_0, \dots, a_{T-1}) = r_T(s_T) + \sum_{t=0}^{T-1} r_t(s_t, a_t)$$

Decision making problem:

$$J^*(s_0) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t), t=0,1,\dots,T-1} J(s_0; a_0, \dots, a_{T-1})$$

Principle of optimality

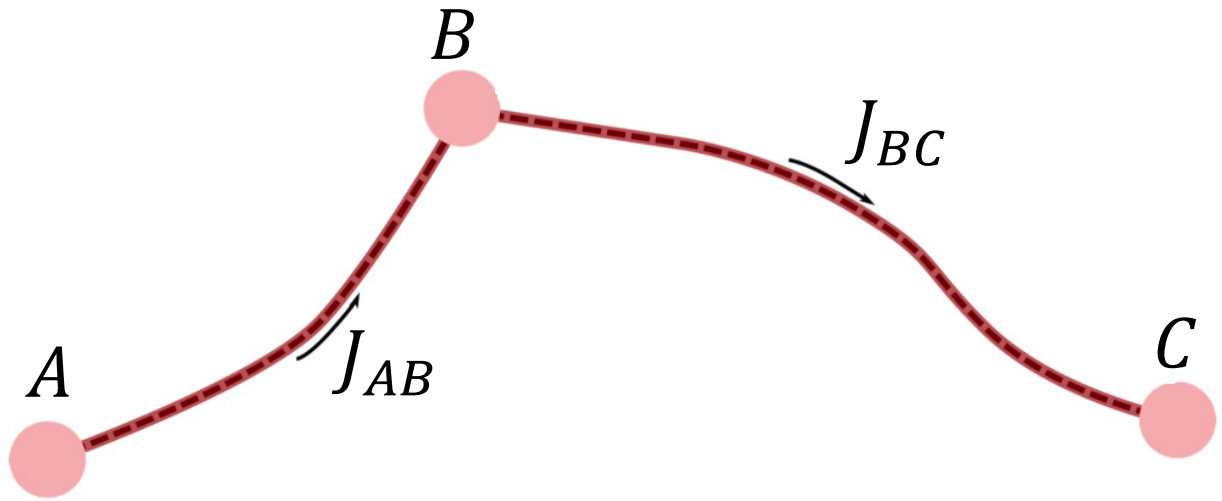
It's the key concept behind the dynamic programming approach.

Suppose $A - B - C$ is the optimal path from A to C.

First segment reward: J_{AB}

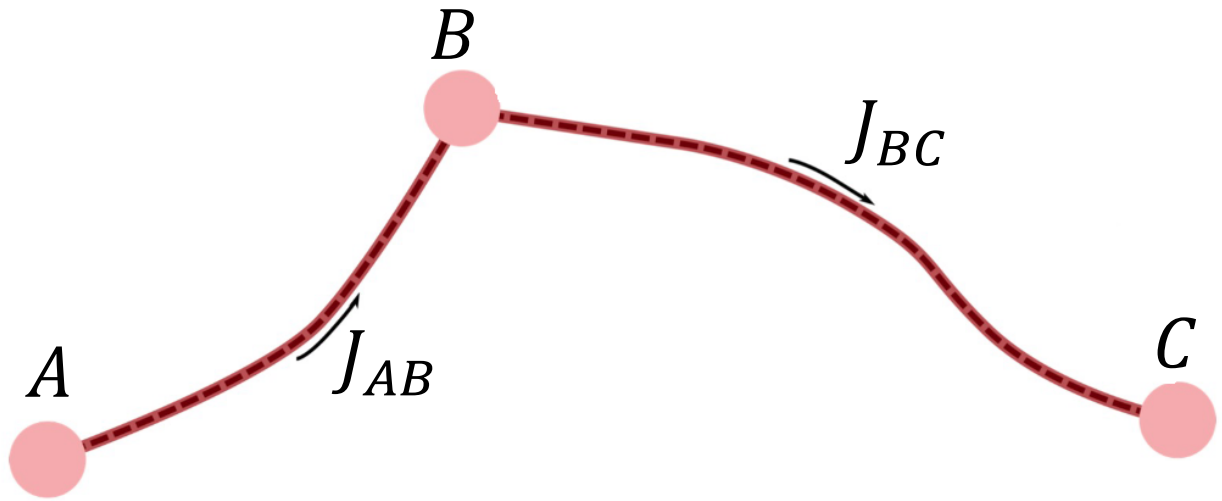
Second segment reward: J_{BC}

Optimal reward $J^*_{AC} = J_{AB} + J_{BC}$



Principle of optimality

If $A - B - C$ is the optimal path from A to C, then $B - C$ is the optimal path from B to C.



Principle of optimality

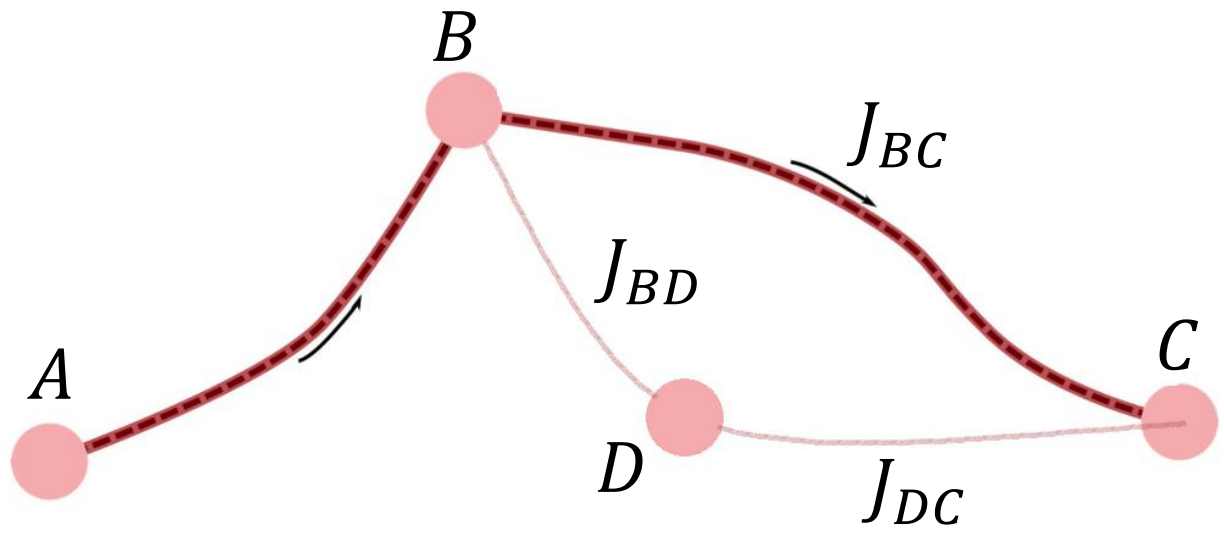
If $A - B - C$ is the optimal path from A to C, then $B - C$ is the optimal path from B to C.

Proof: Suppose $B - D - C$ is the optimal path from B to C. Then,

$$J_{BD} + J_{DC} > J_{BC}$$

and

$$J_{AB} + J_{BD} + J_{DC} > J_{AB} + J_{BC} = J_{AC}^{*}$$



This is a contradiction.

Principle of optimality (deterministic)

Suppose $(a_0^*, a_1^*, \ldots, a_{T-1}^*)$ is an optimal solution to the decision making problem for an initial state s_0^*, and the systems evolves as $(s_0^*, s_1^*, \ldots, s_T^*)$ for this initial state and action sequence.

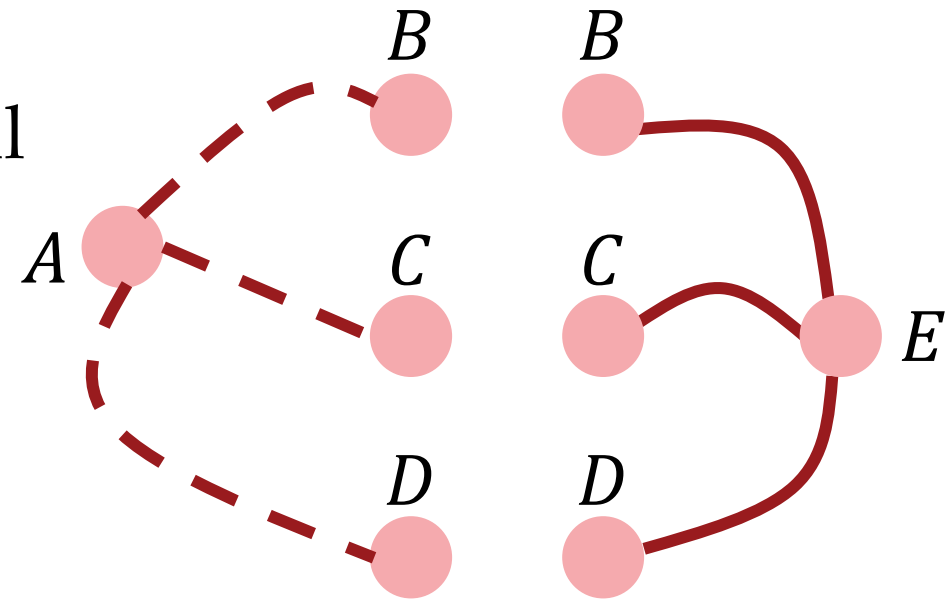
Then, an optimal solution to the subproblem for moving from state s_t^* at time t until time T is $(a_t^*, a_{t+1}^*, \ldots, a_{T-1}^*)$.

Tail of an optimal solution = Optimal for the tail subproblem

How to apply principal of optimality

Goal: Go from A to E.

Given: $B-E$, $C-E$ and $D-E$ are the optimal paths to E from B, C and D, respectively.



Principle of optimality:

If $A-B$ is the initial segment of the optimal path from A to E, then $B-E$ is the final segment of this path.

Then, we find the optimal path by comparing:

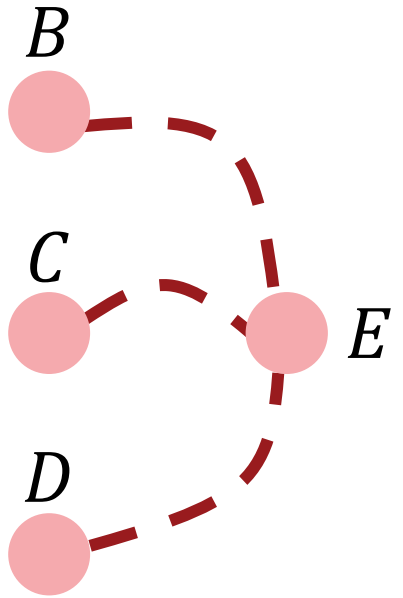
$$J_{ABE} = J_{AB} + J^*_{BE}$$
$$J_{ACE} = J_{AC} + J^*_{CE}$$
$$J_{ADE} = J_{AD} + J^*_{DE}$$

How to apply principal of optimality

Start from the terminal state and go backward in time.

First, compute the optimal paths to E from all possible previous states. The rewards of these paths are reward-to-go (sometimes called return) from those states.

Repeat this procedure backward in time until $t = 0$.



"Life can only be understood backwards; but it must be lived forwards."
- Søren Kierkegaard

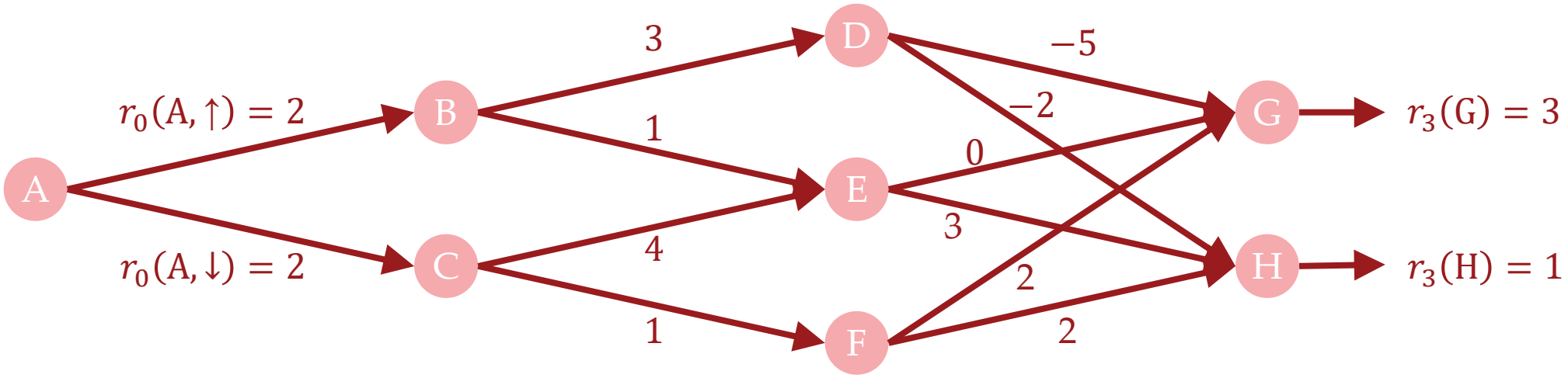
Dynamic programming (deterministic)

$J_T^*(s_T) = r_T(s_T)$, for all $s_T \in \boldsymbol{S}$

for $t = T - 1$ **to** 0 **do**

$$J_t^*(s_t) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t)} r_t(s_t, a_t) + J_{t+1}^*(f_t(s_t, a_t)), \text{ for all } s_t \in \boldsymbol{S}$$

return $J_0^*(\cdot), J_1^*(\cdot), \ldots, J_T^*(\cdot)$



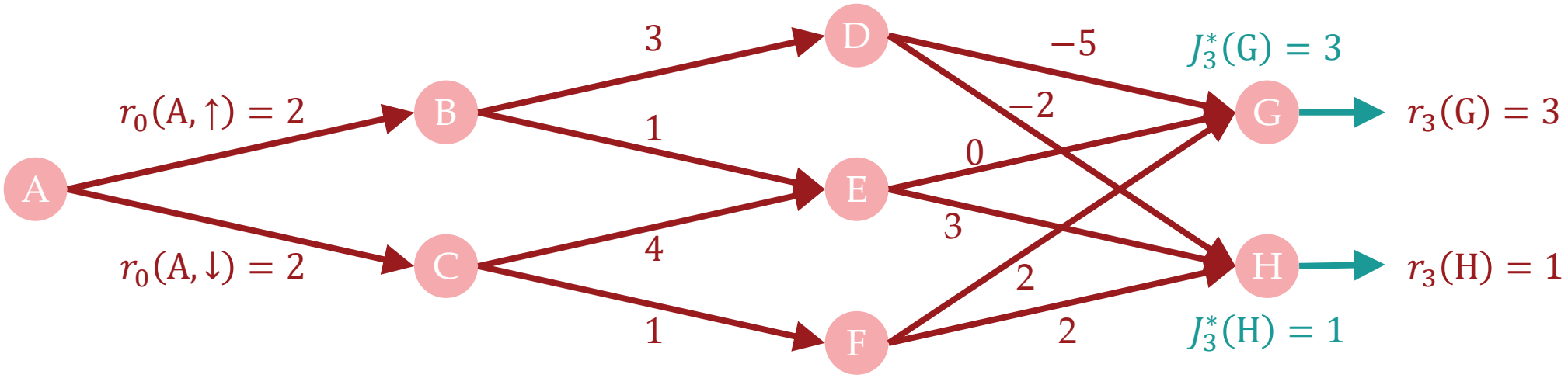
Dynamic programming (deterministic)

$J_T^*(s_T) = r_T(s_T)$, for all $s_T \in \boldsymbol{S}$

for $t = T - 1$ **to** 0 **do**

$$J_t^*(s_t) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t)} r_t(s_t, a_t) + J_{t+1}^*(f_t(s_t, a_t)), \text{ for all } s_t \in \boldsymbol{S}$$

return $J_0^*(\cdot), J_1^*(\cdot), \ldots, J_T^*(\cdot)$



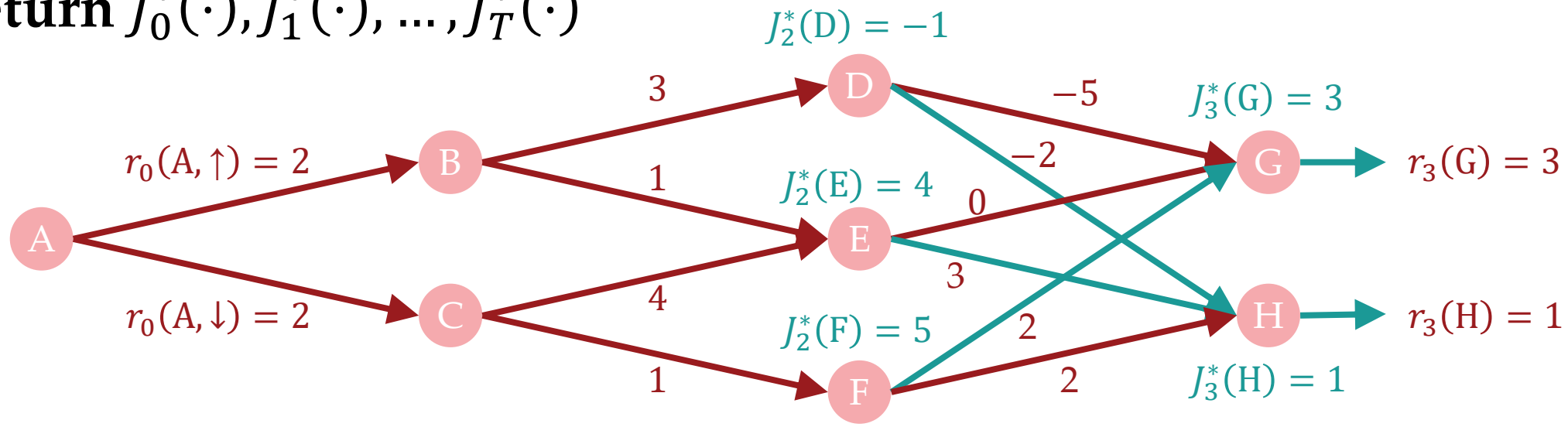
Dynamic programming (deterministic)

$J_T^*(s_T) = r_T(s_T)$, for all $s_T \in \boldsymbol{S}$

for $t = T - 1$ **to** 0 **do**

$$J_t^*(s_t) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t)} r_t(s_t, a_t) + J_{t+1}^*(f_t(s_t, a_t)), \text{ for all } s_t \in \boldsymbol{S}$$

return $J_0^*(\cdot), J_1^*(\cdot), \ldots, J_T^*(\cdot)$



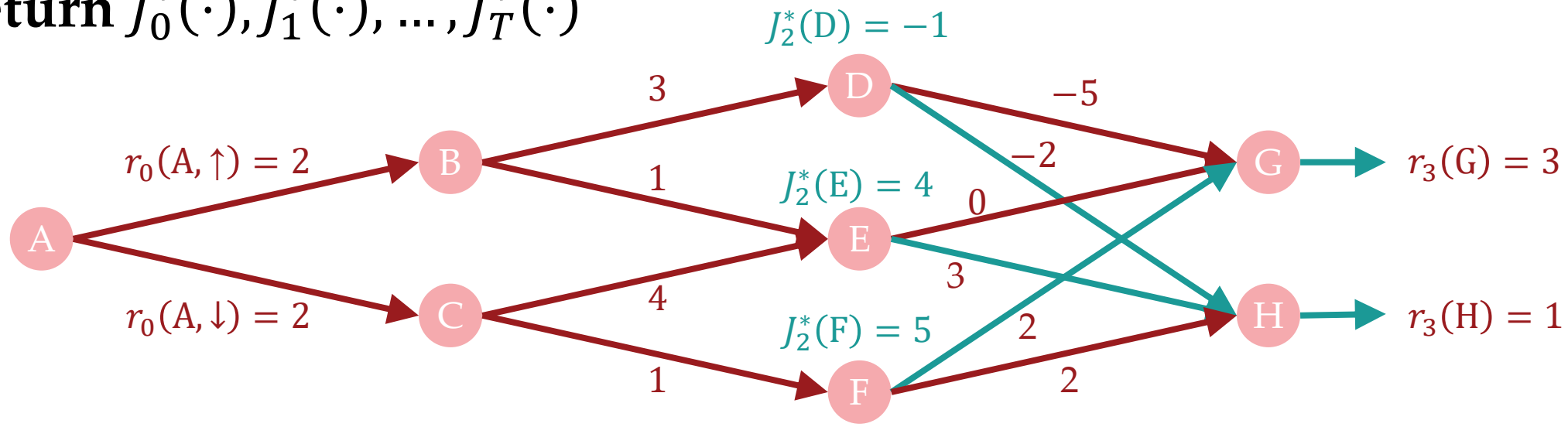
Dynamic programming (deterministic)

$J_T^*(s_T) = r_T(s_T)$, for all $s_T \in \boldsymbol{S}$

for $t = T - 1$ **to** 0 **do**

$$J_t^*(s_t) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t)} r_t(s_t, a_t) + J_{t+1}^*(f_t(s_t, a_t)), \text{ for all } s_t \in \boldsymbol{S}$$

return $J_0^*(\cdot), J_1^*(\cdot), \ldots, J_T^*(\cdot)$



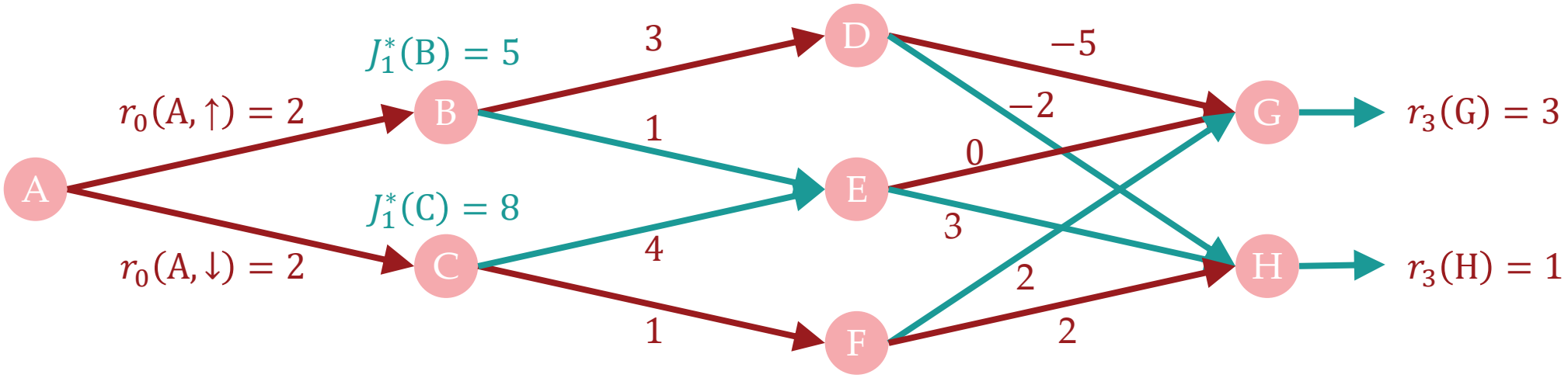
Dynamic programming (deterministic)

$J_T^*(s_T) = r_T(s_T)$, for all $s_T \in \boldsymbol{S}$

for $t = T - 1$ **to** 0 **do**

$$J_t^*(s_t) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t)} r_t(s_t, a_t) + J_{t+1}^*(f_t(s_t, a_t)), \text{ for all } s_t \in \boldsymbol{S}$$

return $J_0^*(\cdot), J_1^*(\cdot), \ldots, J_T^*(\cdot)$



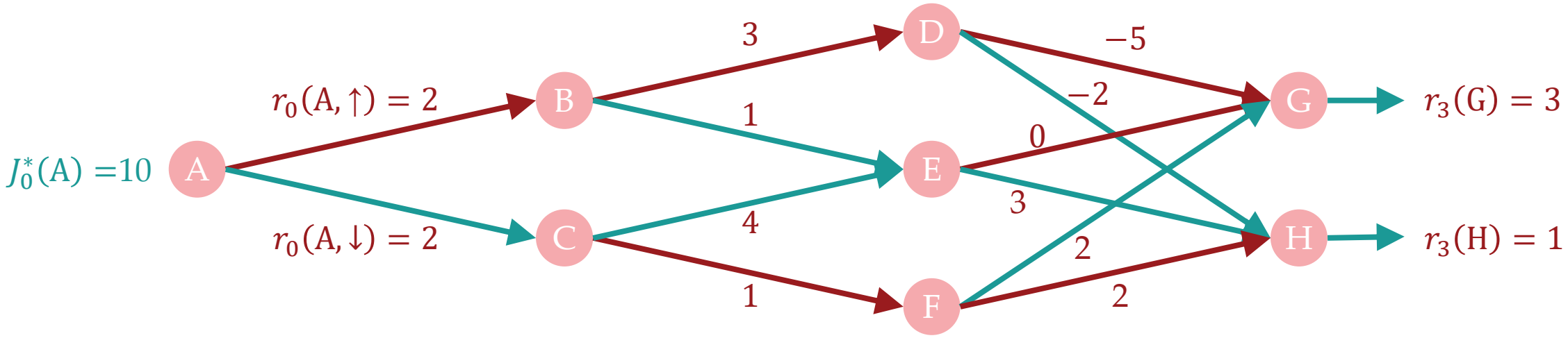
Dynamic programming (deterministic)

$J_T^*(s_T) = r_T(s_T)$, for all $s_T \in \boldsymbol{S}$

for $t = T - 1$ **to** 0 **do**

$$J_t^*(s_t) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t)} r_t(s_t, a_t) + J_{t+1}^*(f_t(s_t, a_t)), \text{ for all } s_t \in \boldsymbol{S}$$

return $J_0^*(\cdot), J_1^*(\cdot), \ldots, J_T^*(\cdot)$



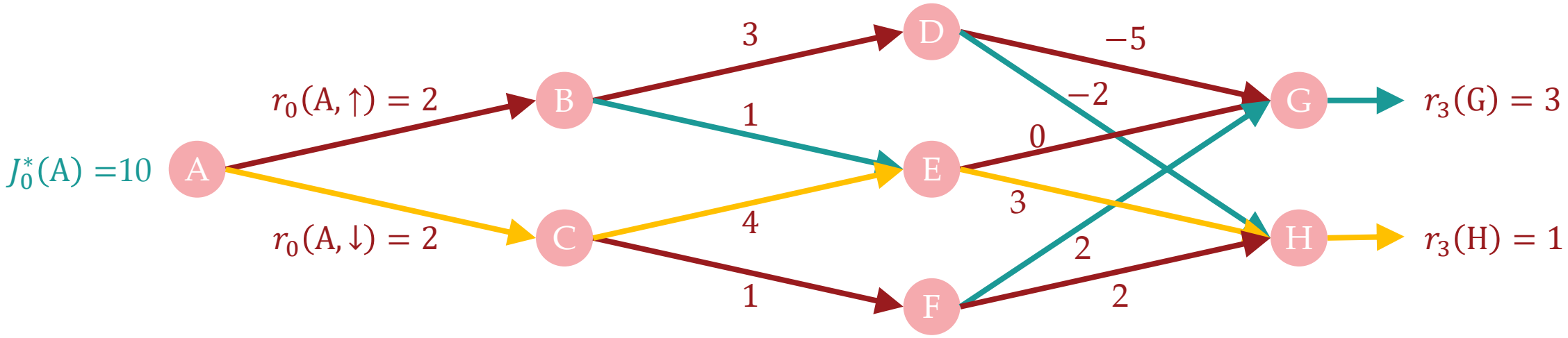
Dynamic programming (deterministic)

$J_T^*(s_T) = r_T(s_T)$, for all $s_T \in \boldsymbol{S}$

for $t = T - 1$ **to** 0 **do**

$$J_t^*(s_t) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t)} r_t(s_t, a_t) + J_{t+1}^*(f_t(s_t, a_t)), \text{ for all } s_t \in \boldsymbol{S}$$

return $J_0^*(\cdot), J_1^*(\cdot), \ldots, J_T^*(\cdot)$



Comments

1- We discretized the time.

Otherwise, we would have to deal with differential equations:

$$\dot{x}(t) = f\big(x(t), u(t)\big)$$

These are usually more difficult to solve. We will mostly focus on discrete-time problems in this course. Exception:

Safe and robust learning:

- Sui et al., Safe Exploration for Optimization with Gaussian Processes (2015).
- Robey et al., Learning Control Barrier Functions from Expert Demonstrations (2020).
- Bansal and Tomlin, Deepreach: A Deep Learning Approach to High-dimensional Reachability (2021).

Comments

2- We quantized the state and action spaces.

This allows us to loop over all states and actions.

Reinforcement learning can be used for continuous spaces, too!

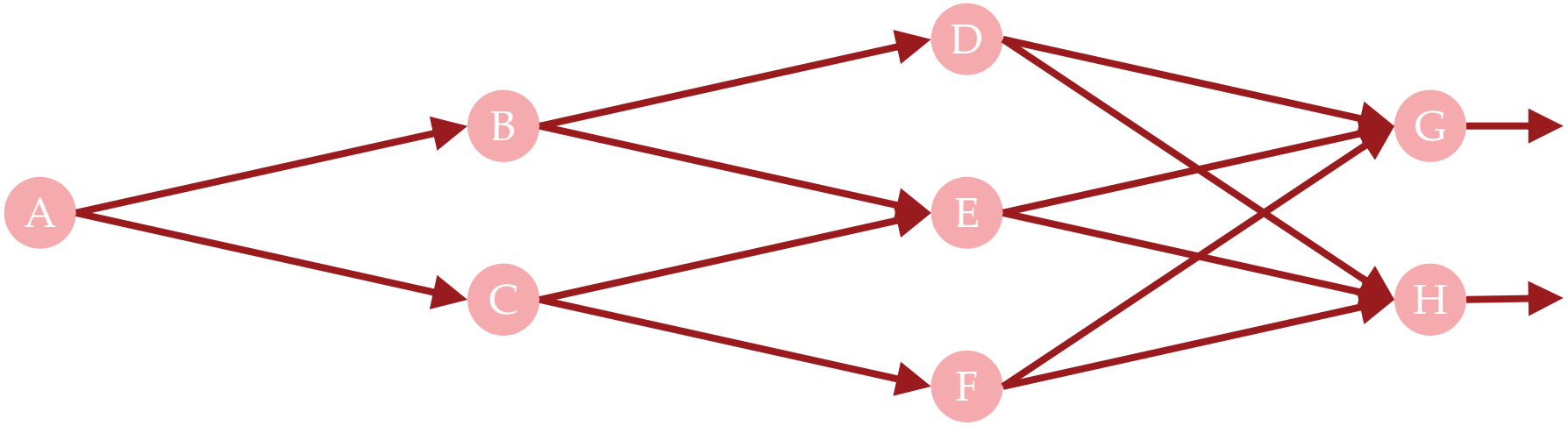
Comments

3- Dynamic programming gives the globally optimal solution!

Comments

4- Constraints help decrease computational costs.

In our example, we did not loop over all states, and optimized over only the feasible actions.



Comments

5- Curse of dimensionality: Computational complexity and memory complexity of dynamic programming increases with the size of the state space.

Size of the state space often increases exponentially with its dimensionality.

Today…

- Dynamic programming in deterministic systems
- Dynamic programming in stochastic systems
- Markov decision processes
- Value/policy iteration

Decision making in stochastic systems

State: $s_t \in \boldsymbol{S}$

Action: $a_t \in \boldsymbol{\mathcal{A}}(s_t)$

This is a random variable.

Transition: $s_{t+1} = f_t(s_t, a_t, w_t)$, or $s_{t+1} \sim P(\cdot|\, s_t, a_t)$

Policies: $\pi = (\pi_0, \pi_1, \ldots, \pi_{T-1})$ where $a_t = \pi_t(s_t)$

We introduce policies since we will find an optimal *closed-loop* policy.

Expected total reward:

$$J_\pi(s_0) = \mathbb{E}_{w_0, w_1, \ldots w_{T-1}}\left[r_T(s_T) + \sum_{t=0}^{T-1} r_t(s_t, \pi_t(s_t), w_t) \right]$$

Decision making problem: $\mathrm{J}^*(\mathrm{s}_0) = \max_\pi J_\pi(s_0)$

Principle of optimality (stochastic)

Suppose $(\pi_0^*, \pi_1^*, \ldots, \pi_{T-1}^*)$ is an optimal solution to the decision making problem and assume state s_t is reachable.

Then, an optimal solution to the subproblem for moving from state s_t at time t until time T is $(\pi_t^*, \pi_{t+1}^*, \ldots, \pi_{T-1}^*)$.

Tail of optimal policies = Optimal for the tail subproblem

Dynamic programming (stochastic)

$J_T(s_T) = r_T(s_T)$, for all $s_T \in \boldsymbol{\mathcal{S}}$

for $t = T - 1$ **to** 0 **do**

$$J_t(s_t) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t)} \mathbb{E}_{w_t}\left[r_t(s_t, a_t, w_t) + J_{t+1}(f_t(s_t, a_t, w_t))\right], \text{ for all } s_t \in \boldsymbol{\mathcal{S}}$$

return $J_0(\cdot), J_1(\cdot), \ldots, J_T(\cdot)$

Comments

DP in stochastic systems suffers from the same problems as DP in deterministic systems.

Also, modeling transitions perfectly is not always possible.

Today…

- Dynamic programming in deterministic systems
- Dynamic programming in stochastic systems
- Markov decision processes
- Value/policy iteration

Markov decision processes (MDP)

MDPs are useful tools to model an agent's interaction with its environment.

Reinforcement learning algorithms try to solve MDPs.

They have advantages over DP, as they only need a reward function.

Infinite horizon MDPs

We are looking at the infinite horizon case, since it makes stationary policies optimal.

State: $s \in \boldsymbol{\mathcal{S}}$

Action: $a \in \boldsymbol{\mathcal{A}}$

We removed the dependence on the state, although that's also creates interesting research questions.

Transition: $s_{t+1} \sim P(\cdot | s_t, a_t)$

Reward: $r_t = R(s_t, a_t)$

Discount: $\gamma \in [0,1)$

Policy: $\pi: \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{A}}$ or $\pi: \boldsymbol{\mathcal{S}} \to \Delta\boldsymbol{\mathcal{A}}$

Infinite horizon MDPs

State: $s \in \boldsymbol{\mathcal{S}}$

Action: $a \in \boldsymbol{\mathcal{A}}$

Transition: $s_{t+1} \sim P(\cdot|\, s_t, a_t)$

Reward: $r_t = R(s_t, a_t)$

Discount: $\gamma \in [0,1)$

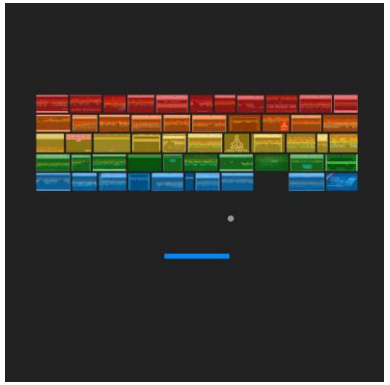
Policy: $\pi: \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{A}}$ or $\pi: \boldsymbol{\mathcal{S}} \to \Delta\boldsymbol{\mathcal{A}}$

Goal:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, \pi(s_t)\big)\right]$$

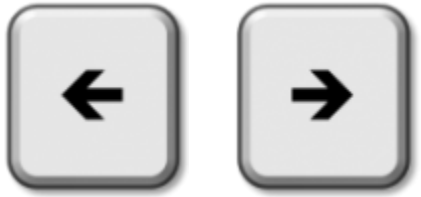
An example MDP

Goal: Achieve a high score in the Atari game "Breakout"



States: Image of the current screen (?)

Actions: Left and right actions



Reward: Change in the score of the game

Playing atari with deep reinforcement learning
Mnih et al., NeurIPS Deep Learning Workshop 2013

Another example MDP

Goal: Make an RC helicopter fly and perform some maneuvers

States: Sensory input of the helicopter

Actions: Control inputs 

Reward: Positive for the maneuvers, negative for crashing

(This is usually what we need to hand-design)

Autonomous helicopter aerobatics through apprenticeship learning
Abbeel et al., IJRR 2010

Not that easy!

Goal: Make an RC helicopter fly and perform some maneuvers

States: Sensory input of the helicopter

Actions: Control inputs 

Reward: Positive for the maneuvers, negative for crashing

This is a very naïve reward function. They instead learned the reward from expert demonstrations. We will cover this topic in a few weeks.

Infinite horizon MDPs

State: $s \in \boldsymbol{\mathcal{S}}$

Action: $a \in \boldsymbol{\mathcal{A}}$

Transition: $s_{t+1} \sim P(\cdot | s_t, a_t)$

Reward: $r_t = R(s_t, a_t)$

Discount: $\gamma \in [0,1)$

Policy: $\pi: \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{A}}$ or $\pi: \boldsymbol{\mathcal{S}} \to \Delta\boldsymbol{\mathcal{A}}$

Goal:

Reinforcement learning tries to solve this problem.

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, \pi(s_t)\big)\right]$$

Partially observable MDPs

State: $s \in \boldsymbol{\mathcal{S}}$

Action: $a \in \boldsymbol{\mathcal{A}}$

Transition: $s_{t+1} \sim P(\cdot | \, s_t, a_t)$

Reward: $r_t = R(s_t, a_t)$

Discount: $\gamma \in [0,1)$

Observation: $o \in \boldsymbol{\mathcal{O}}$

Observation Model: $o_t \sim \Omega(\cdot | \, s_t)$

Policy: $\pi: \boldsymbol{\mathcal{O}} \to \boldsymbol{\mathcal{A}}$ or $\pi: \boldsymbol{\mathcal{O}} \to \Delta\boldsymbol{\mathcal{A}}$

Goal:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, \pi(o_t)\big)\right]$$

Today…

- Dynamic programming in deterministic systems
- Dynamic programming in stochastic systems
- Markov decision processes
- Value/policy iteration

Value functions

State value function: $V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s]$

State-action value function: $Q^{\pi}(s,a) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a]$

$$V^{\pi}(s) = R\big(s,\pi(s)\big) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,\pi(s))}[V^{\pi}(s')]$$

$$Q^{\pi}(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a), a' \sim \pi(s')}[Q^{\pi}(s',a')]$$

For any stationary policy, these have unique solutions.
Hint: Think of it as a system of linear equations.

Bellman equations

State value function: $V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s]$

State-action value function: $Q^{\pi}(s,a) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a]$

$$V^*(s) = \max_a \underbrace{\left(R(s,a) + \gamma \sum_{s' \in \boldsymbol{S}} P(s' \mid s,a) V^*(s') \right)}_{\text{This is just } Q^*(s,a)!}$$

Bellman equations

State value function: $V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s]$

State-action value function: $Q^{\pi}(s,a) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a]$

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s' \in \boldsymbol{S}} P(s' \mid s,a) V^*(s') \right)$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \boldsymbol{S}} P(s' \mid s,a) \max_{a'} Q^*(s',a')$$

Value iteration

Idea: Take Bellman equation and iterate until it converges. It does converge because it is a contractive mapping.

$V_0(s) = 0$ for all $s \in \boldsymbol{\mathcal{S}}$

for $k = 0,1, \ldots$ **until** convergence:

  for all $s \in \boldsymbol{\mathcal{S}}$:

$$V_{k+1}(s) = \max_a \left(R(s,a) + \gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} P(s' \mid s,a) V_k(s') \right)$$

Each iteration is $O(|\boldsymbol{\mathcal{S}}|^2 |\boldsymbol{\mathcal{A}}|)$.

Value iteration

Idea: Take Bellman equation and iterate until it converges. It does converge because it is a contractive mapping.

$Q_0(s, a) = 0$ for all $s \in \boldsymbol{\mathcal{S}}$, $a \in \boldsymbol{\mathcal{A}}$

for $k = 0,1, \ldots$ **until** convergence:

  for all $s \in \boldsymbol{\mathcal{S}}$, $a \in \boldsymbol{\mathcal{A}}$:

$$Q_{k+1}(s, a) = R(s, a) + \gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} P(s' \mid s, a) \max_{a' \in \boldsymbol{\mathcal{A}}} Q_k(s', a')$$

Each iteration is $O(|\boldsymbol{\mathcal{S}}|^2 |\boldsymbol{\mathcal{A}}|^2)$.

Policy iteration

Initialize a random policy π_0.

for $k = 0,1, \ldots$ **until** convergence:

This is called policy evaluation.
It is $O(|\mathcal{S}|^3)$.

Solve the following system for V^{π_k}:

$$V^{\pi_k}(s) = \mathbb{E}_{a \sim \pi_k(s)}\left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^{\pi_k}(s')\right]$$

This is called policy improvement.
It is $O(|\mathcal{S}|^2|\mathcal{A}|)$.

for all $s \in \mathcal{S}$:

$$\pi_k(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^{\pi_k}(s')\right)$$

Value iteration vs policy iteration

- Both converge.
- Policy iteration requires more complex implementation.
- In practice, policy iteration usually converges faster.

Today…

- Dynamic programming in deterministic systems
- Dynamic programming in stochastic systems
- Markov decision processes
- Value/policy iteration

We are still assuming we know the transition function.

Next time…

- Model-based reinforcement learning
- Model-free reinforcement learning