Markov Decision Processes
Chapter 17

Mausam

Planning Agent

Static vs. Dynamic

Environment

Fully vs. Partially Observable

Perfect vs. Noisy

What action next?

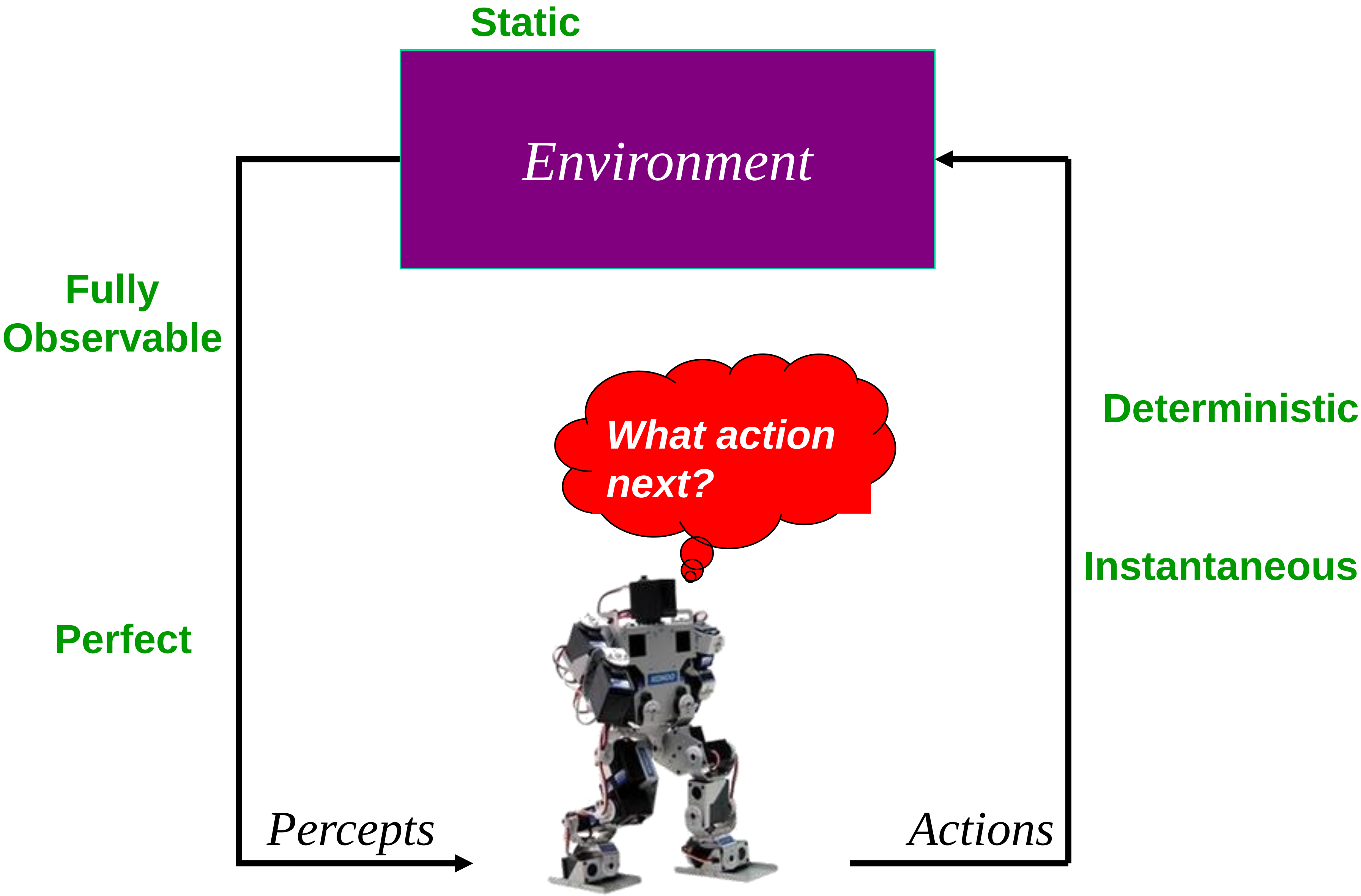
Deterministic vs. Stochastic

Instantaneous vs. Durative

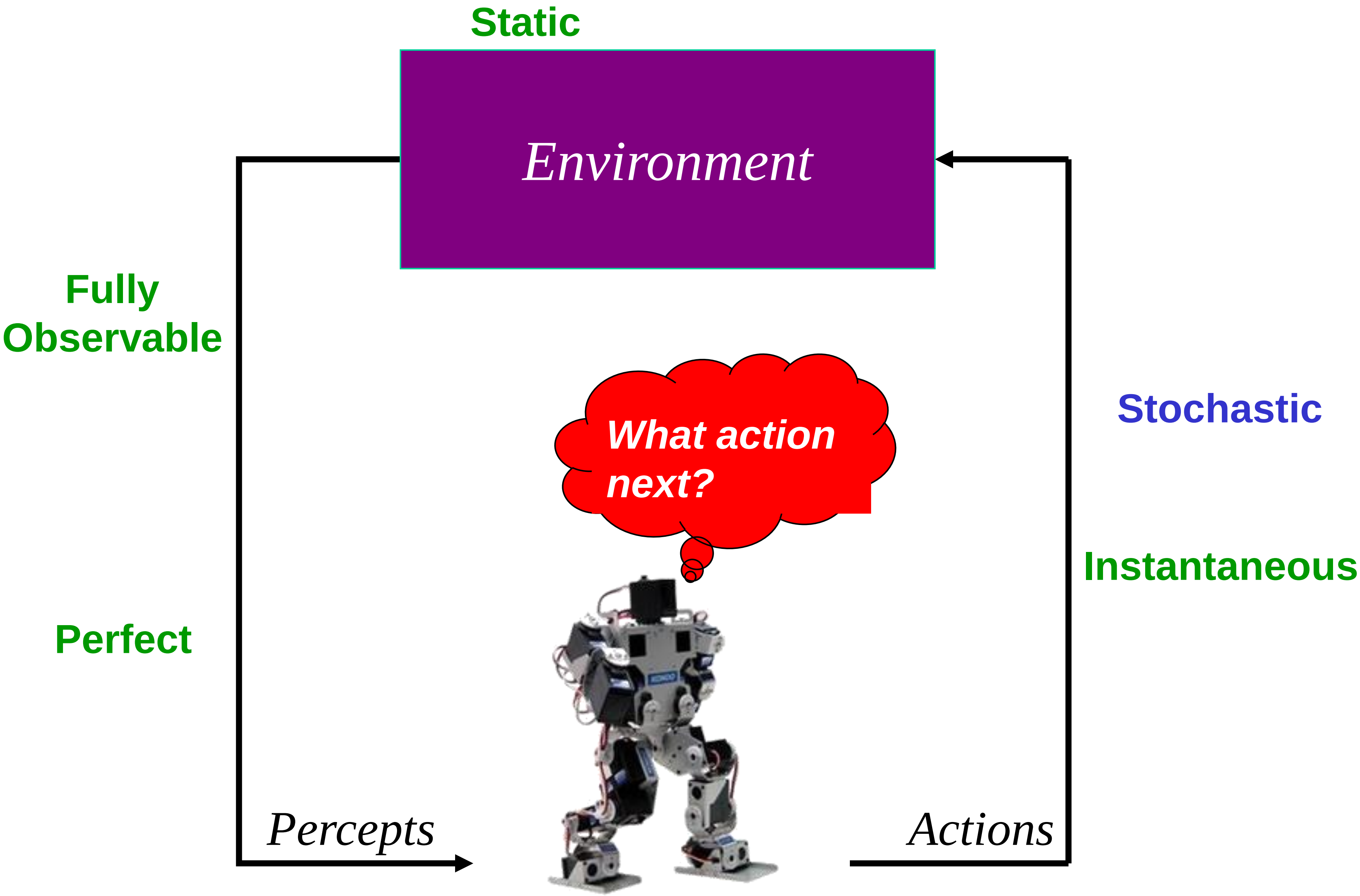
Percepts

Actions

Classical Planning



Stochastic Planning: MDPs



MDP vs. Decision Theory

- Decision theory - episodic
- MDP -- sequential

Markov Decision Process (MDP)

- $\mathcal{S}$: A set of states
- $\mathcal{A}$: A set of actions
- $\mathcal{P}r(s'|s,a)$: transition model
- $\mathcal{C}(s,a,s')$: cost model
- $\mathcal{G}$: set of goals
- s_0: start state
- γ: discount factor
- $\mathcal{R}(s,a,s')$: reward model

factored

Factored MDP

absorbing/
non-absorbing

Objective of an MDP

- Find a policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$

- which optimizes
  - minimizes $\begin{pmatrix} \text{discounted} \\ \text{or} \\ \text{undiscount.} \end{pmatrix}$ expected cost to reach a goal
  - maximizes expected reward
  - maximizes expected (reward-cost)

- given a ____ horizon
  - finite
  - infinite
  - indefinite

- assuming full observability

Role of Discount Factor (γ)

- Keep the total reward/total cost finite
  - useful for infinite horizon problems

- Intuition (economics):
  - Money today is worth more than money tomorrow.

- Total reward: $r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots$
- Total cost: $c_1 + \gamma c_2 + \gamma^2 c_3 + \ldots$

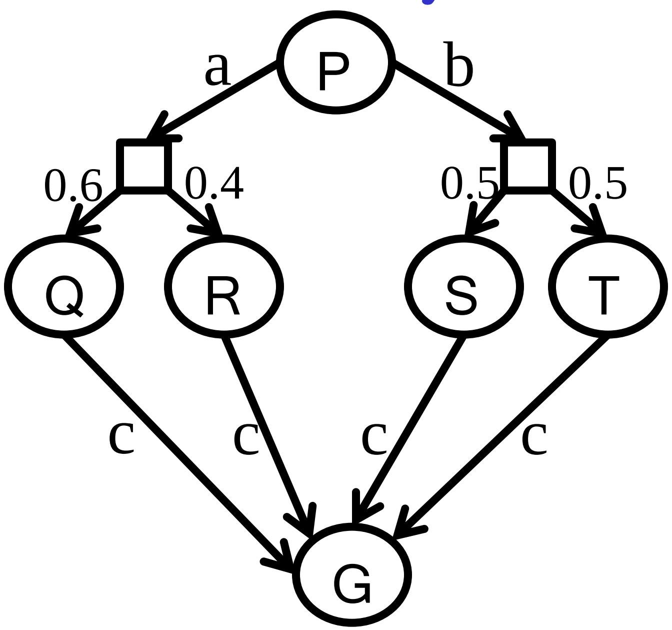
Examples of MDPs

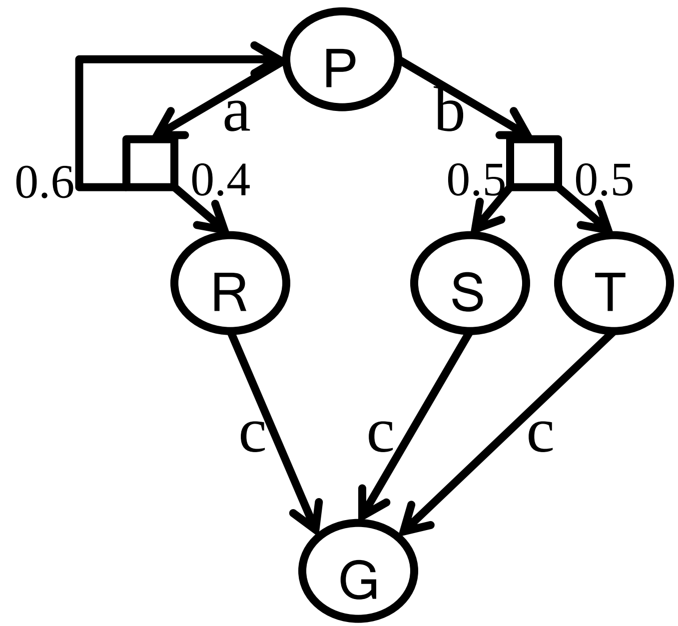
- Goal-directed, Indefinite Horizon, Cost Minimization MDP
  - $<\mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{C}, \mathcal{G}, s_0>$
  - Most often studied in planning, graph theory communities
- Infinite Horizon, Discounted Reward Maximization MDP
  - $<\mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{R}, \gamma>$
  - Most often studied in machine learning, economics, operations research communities

most popular

- Oversubscription Planning: Non absorbing goals, Reward Max. MDP
  - $<\mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{G}, \mathcal{R}, s_0>$
  - Relatively recent model

AND/OR Acyclic Graphs vs. MDPs





C(a) = 5, C(b) = 10, C(c) =1

Expectimin works

- V(Q/R/S/T) = 1
- V(P) = 6 – action a

Expectimin doesn't work

- infinite loop

- V(R/S/T) = 1
- Q(P,b) = 11
- Q(P,a) = ????
- suppose I decide to take a in P
- Q(P,a) = 5+ 0.4*1 + 0.6Q(P,a)
- ➔ = 13.5

Bellman Equations for MDP_1

- $<\mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{C}, \mathcal{G}, s_0>$
- Define J*(s) {optimal cost} as the minimum expected cost to reach a goal from this state.
- J* should satisfy the following equation:

$$J^*(s) = 0 \ if \ s \in \mathcal{G}$$

$$J^*(s) = \qquad \qquad]$$

Bellman Equations for MDP_2

- $<\mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{R}, s_0, \gamma>$
- Define **V*(s)** {optimal **value**} as the **maximum** expected **discounted reward** from this state.
- V* should satisfy the following equation:

$$V^*(s) \quad = \quad \max_{a \in Ap(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a) \left[\mathcal{R}(s,a,s') + \gamma V^*(s') \right]$$

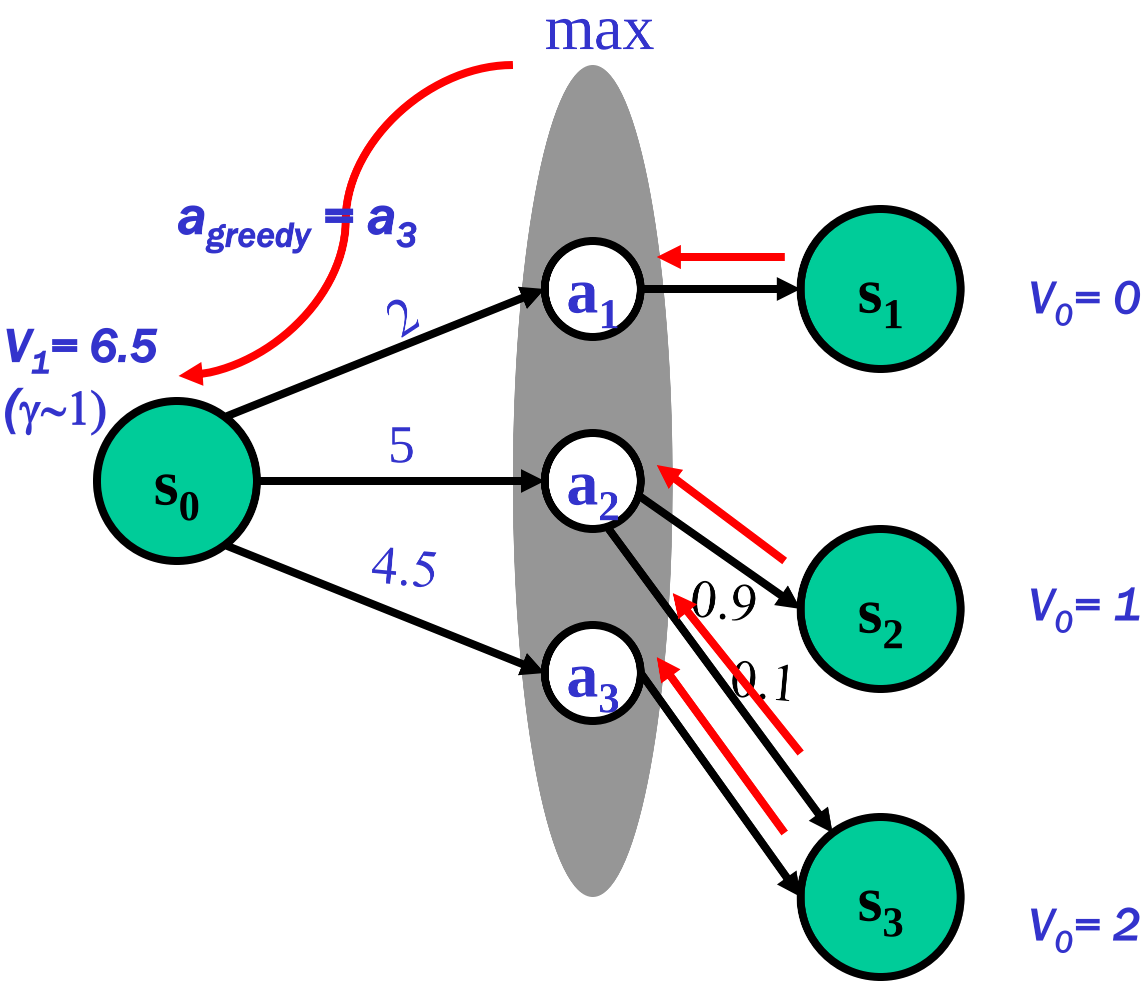
Bellman Backup (MDP_2)

- Given an estimate of V* function (say V_n)
- Backup V_n function at state s
  - calculate a new estimate (V_{n+1}) :

$$\begin{aligned} Q_{n+1}(s,a) &= \sum_{s' \in \mathcal{S}} Pr(s'|s,a) \left[\mathcal{R}(s,a,s') + \gamma V_n(s')\right] \\ V_{n+1}(s) &= \max_{a \in Ap(s)} \left[Q_{n+1}(s,a)\right] \end{aligned}$$

- $Q_{n+1}(s,a)$: value/cost of the strategy:
  - execute action a in s, execute π_n subsequently
  - $\pi_n = \text{argmax}_{a \in Ap(s)} Q_n(s,a)$

Bellman Backup



$$Q_1(s,a_1) = 2 + 0\,\gamma$$
$$Q_1(s,a_2) = 5 + \gamma\, 0.9 \times 1 + \gamma\, 0.1 \times 2$$
$$Q_1(s,a_3) = 4.5 + 2\,\gamma$$

Value iteration [Bellman'57]

- assign an arbitrary assignment of V_0 to each state.
- repeat
  - for all states s
    - compute $V_{n+1}(s)$ by Bellman backup at s.
- until $\max_s |V_{n+1}(s) - V_n(s)| < \varepsilon$

Iteration n+1

Residual(s)

ε-convergence

Comments

- Decision-theoretic Algorithm
- Dynamic Programming
- Fixed Point Computation
- Probabilistic version of Bellman-Ford Algorithm
  - for shortest path computation
  - MDP_1 : Stochastic Shortest Path Problem

- Time Complexity
  - one iteration: $O(|\mathcal{S}|^2|\mathcal{A}|)$
  - number of iterations: poly($|\mathcal{S}|$, $|\mathcal{A}|$, $1/(1-\gamma)$)
- Space Complexity: $O(|\mathcal{S}|)$
- Factored MDPs = Planning under uncertainty
  - exponential space, exponential time

Convergence Properties

- $V_n \rightarrow V^*$ in the limit as $n \rightarrow \infty$
- ε-convergence: V_n function is within ε of V^*
- Optimality: current policy is within $2\varepsilon\gamma/(1-\gamma)$ of optimal

- Monotonicity
  - $V_0 \leq_p V^* \Rightarrow V_n \leq_p V^*$ (V_n monotonic from below)
  - $V_0 \geq_p V^* \Rightarrow V_n \geq_p V^*$ (V_n monotonic from above)
  - otherwise V_n non-monotonic

Policy Computation

$$\pi^*(s) = \operatorname*{argmax}_{a \in Ap(s)} Q^*(s,a)$$
$$= \operatorname*{argmax}_{a \in Ap(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a) \left[\mathcal{R}(s,a,s') + \gamma V^*(s')\right]$$

Policy Evaluation

$$V_\pi(s) = \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,\pi(s)) \left[\mathcal{R}(s,\pi(s),s') + \gamma V_\pi(s')\right]$$

A system of linear equations in $|\mathcal{S}|$ variables.

Changing the Search Space

- Value Iteration
  - Search in value space
  - Compute the resulting policy

- Policy Iteration
  - Search in policy space
  - Compute the resulting value

Policy iteration [Howard'60]

- assign an arbitrary assignment of π_0 to each state.

- repeat
  - Policy Evaluation: compute V_{n+1}: the evaluation of π_n
  - Policy Improvement: for all states s
    - compute $\pi_{n+1}(s)$: $\text{argmax}_{a\in Ap(s)} Q_{n+1}(s,a)$
- until $\pi_{n+1} = \pi_n$

costly: $O(n^3)$

Modified Policy Iteration → **approximate by value iteration using fixed policy**

Advantage

- searching in a finite (policy) space as opposed to uncountably infinite (value) space ⇒ convergence faster.
- all other properties follow!

Modified Policy iteration

- assign an arbitrary assignment of π_0 to each state.
- repeat
  - Policy Evaluation: compute V_{n+1} the *approx.* evaluation of π_n
  - Policy Improvement: for all states s
    - compute $\pi_{n+1}(s)$: $\text{argmax}_{a \in Ap(s)} Q_{n+1}(s,a)$
- until $\pi_{n+1} = \pi_n$

Advantage

- probably the most competitive synchronous dynamic programming algorithm.

Applications

- Stochastic Games
- Robotics: navigation, helicopter manuevers…
- Finance: options, investments
- Communication Networks
- Medicine: Radiation planning for cancer
- Controlling workflows
- Optimize bidding decisions in auctions
- Traffic flow optimization
- Aircraft queueing for landing; airline meal provisioning
- Optimizing software on mobiles
- Forest firefighting
- …

Extensions

- Heuristic Search + Dynamic Programming
  - AO*, LAO*, RTDP, …
- Factored MDPs
  - add planning graph style heuristics
  - use goal regression to generalize better
- Hierarchical MDPs
  - hierarchy of sub-tasks, actions to scale better
- Reinforcement Learning
  - learning the probability and rewards
  - acting while learning – connections to psychology
- Partially Observable Markov Decision Processes
  - noisy sensors; partially observable environment
  - popular in robotics

Asynchronous Value Iteration

- States may be backed up in any order
  - instead of an iteration by iteration
- As long as all states backed up infinitely often
  - Asynchronous Value Iteration converges to optimal