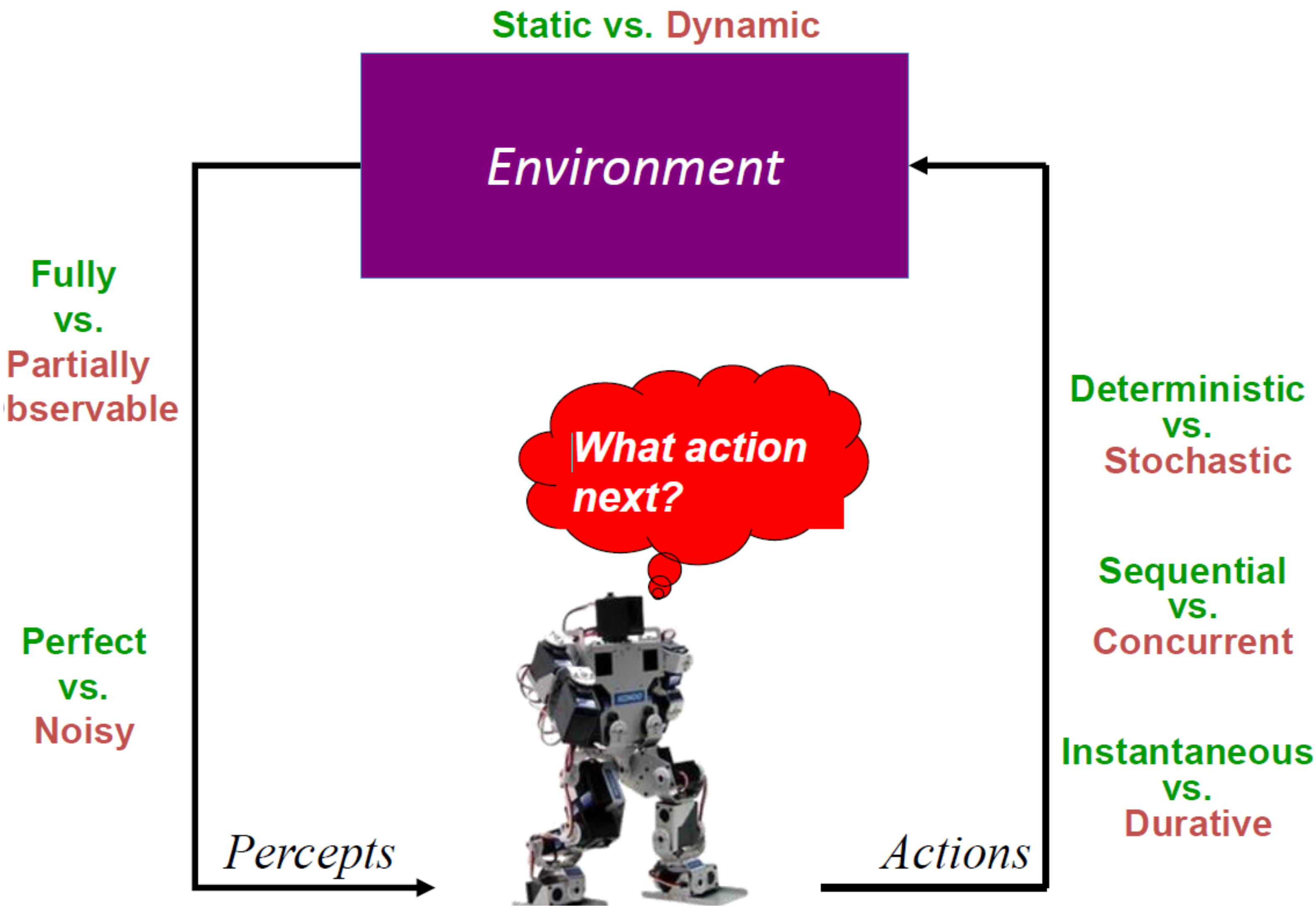
Markov Decision Processes

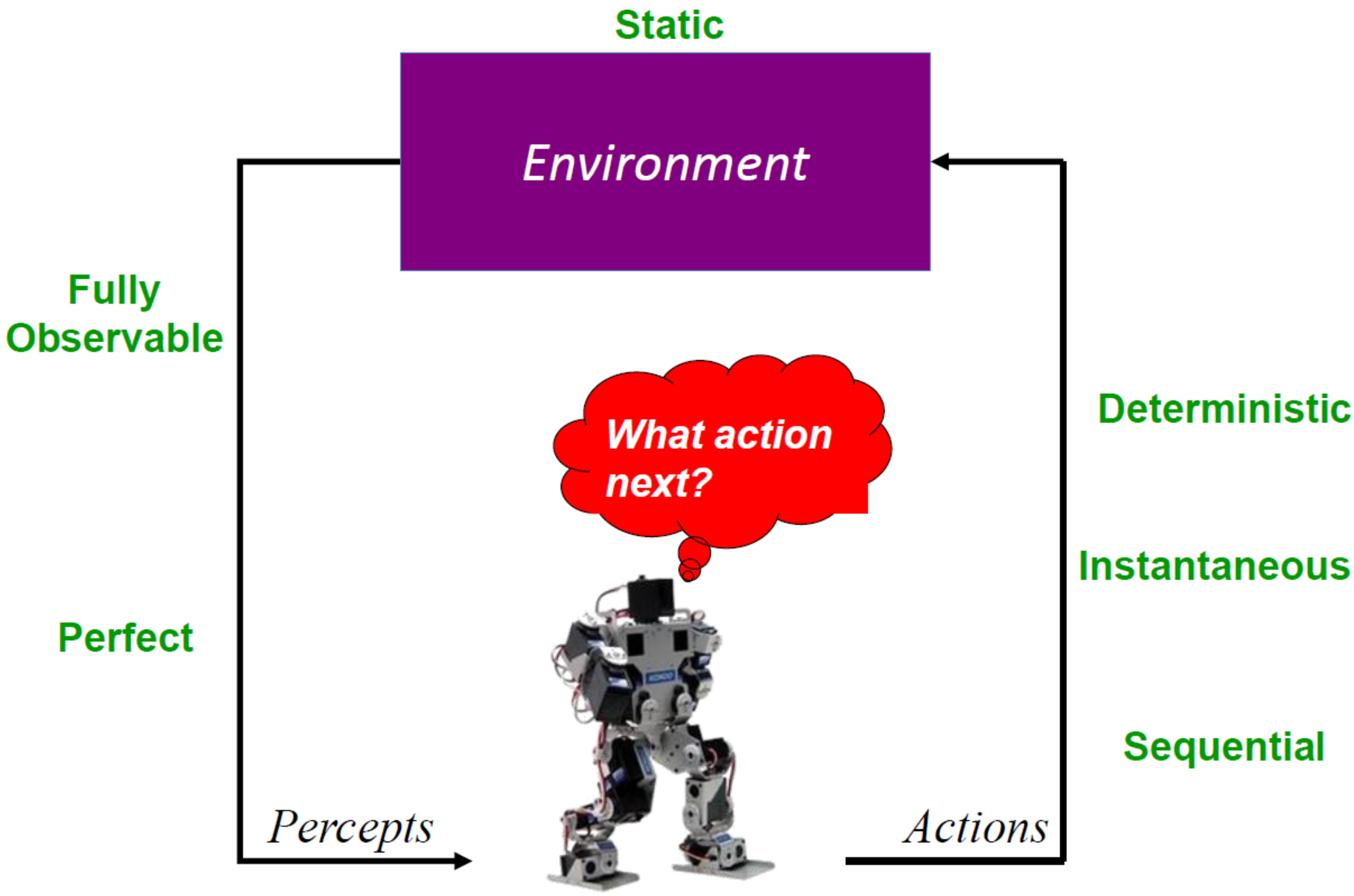
Michal Jakob
Agent Technology Center,
Dept. of Computer Science and Engineering,
FEE, Czech Technical University

AE4M36PAH 2014/2015 - Lecture 12

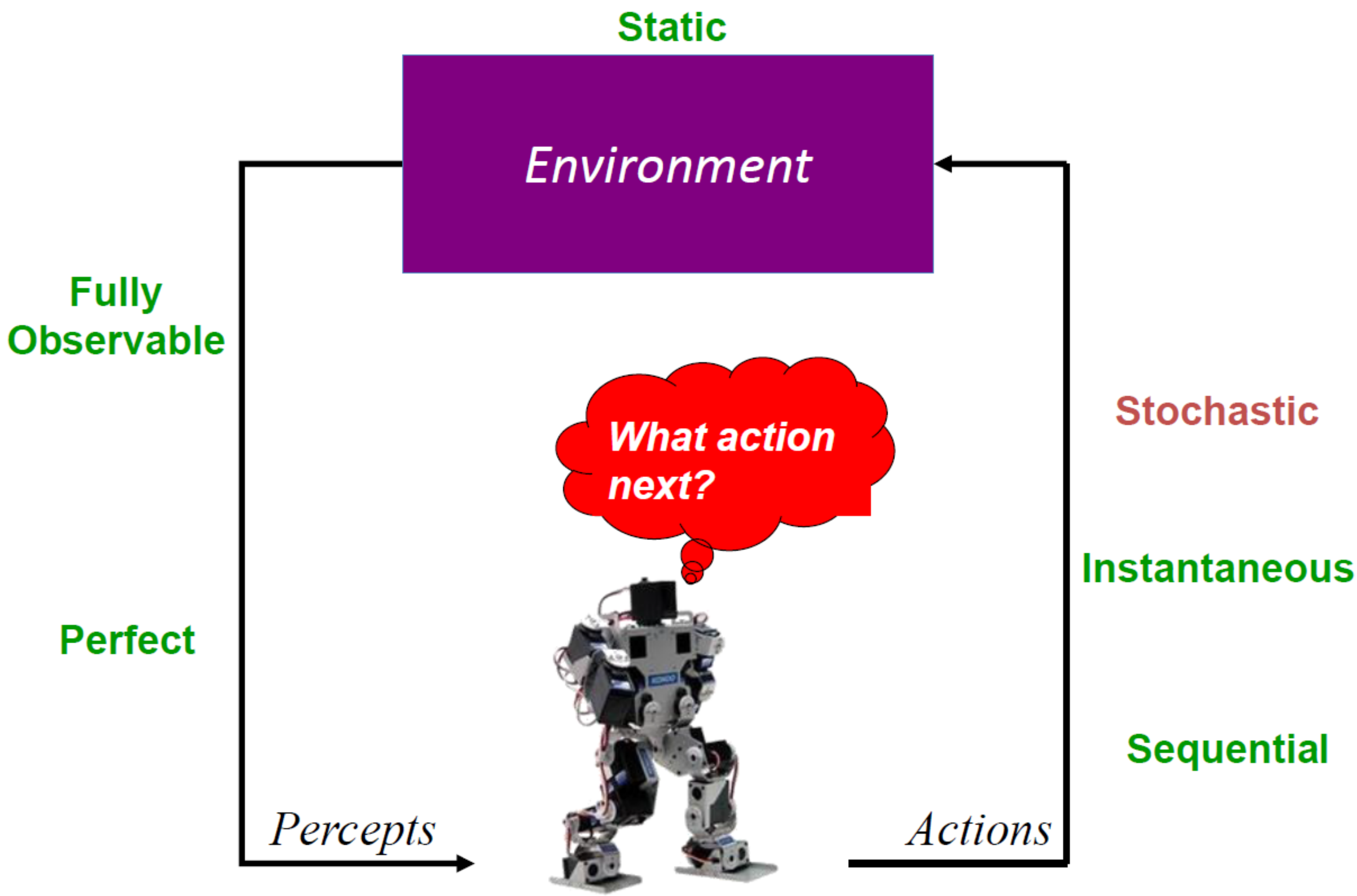
Planning


Static vs. Dynamic
Environment
Fully
vs.
Partially
bservable
Perfect
vs.
Noisy
What action next?
Deterministic
vs.
Stochastic
Sequential
vs.
Concurrent
Instantaneous
vs.
Durative
Percepts
Actions

Classical Planning



Probabilistic Planning



Rational Behaviour in a Stochastic World

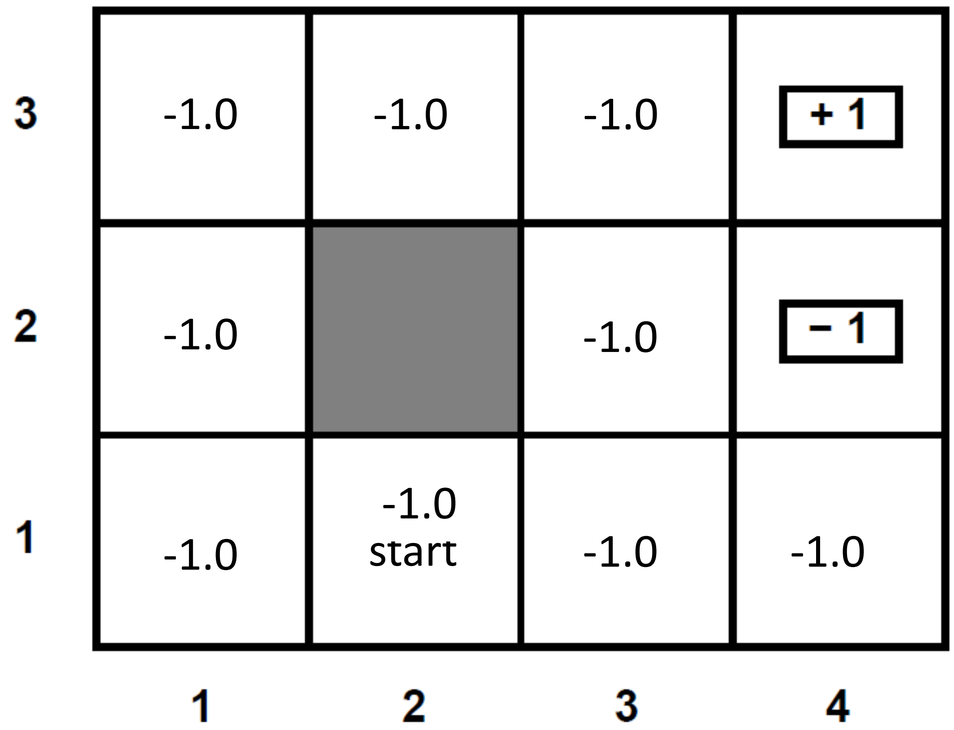
Plan and then execute no longer **rational** in **stochastic** world.

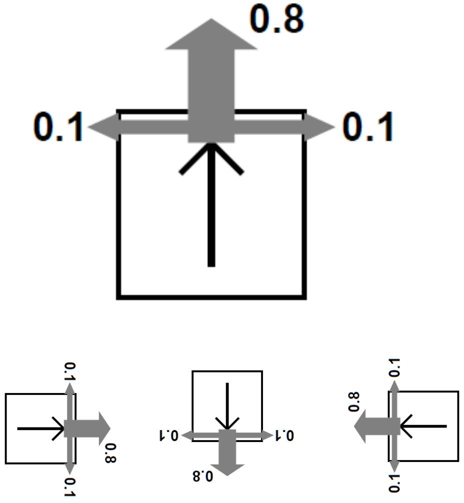
Future **projections not reliable**. Action may result in other states than desired.

Rational agent has to consider the new reality… and **decide/replan** accordingly.

➔ **Sequential decision making**

Grid World Example





Lecture Online

MDP: Formal Model

MDP: Solution Techniques

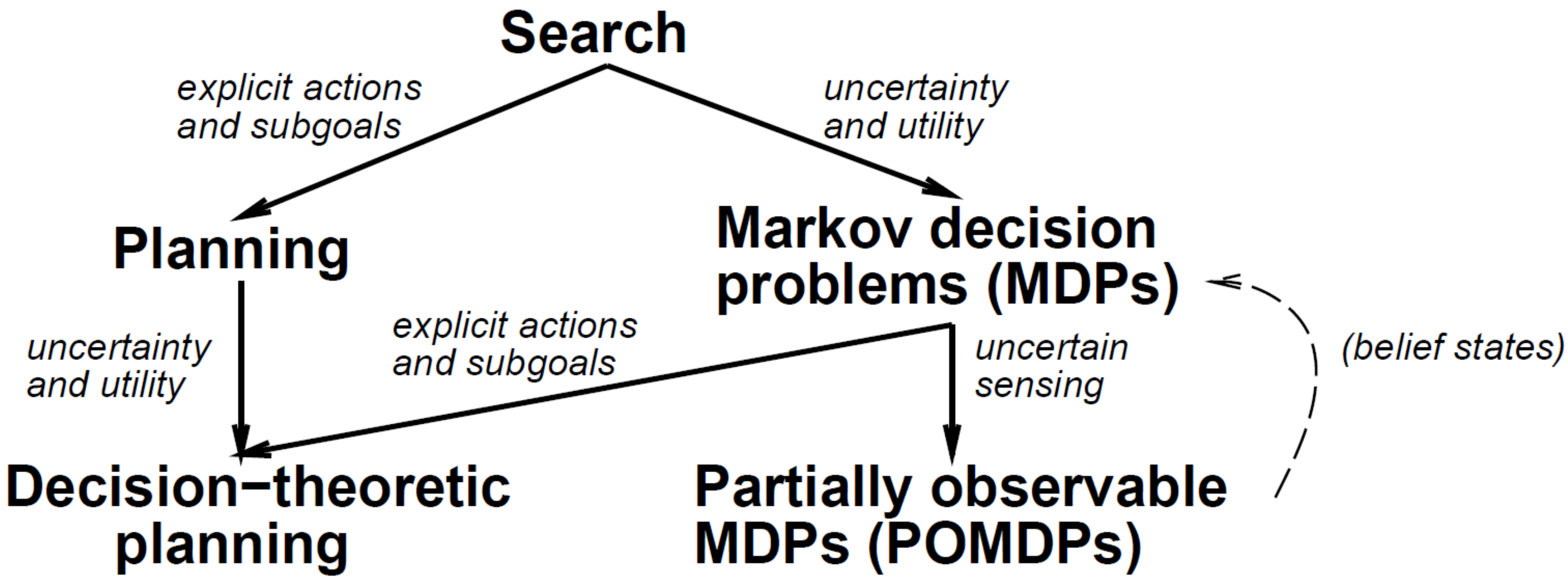
- Value iteration
- Policy iteration
- Modified policy interation

MDP: Advanced Topics

MDP Formalization

Markov Decision Processes

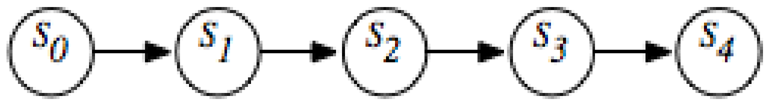
MDPs in Context



(Probabilistic planning)

Markov Chain

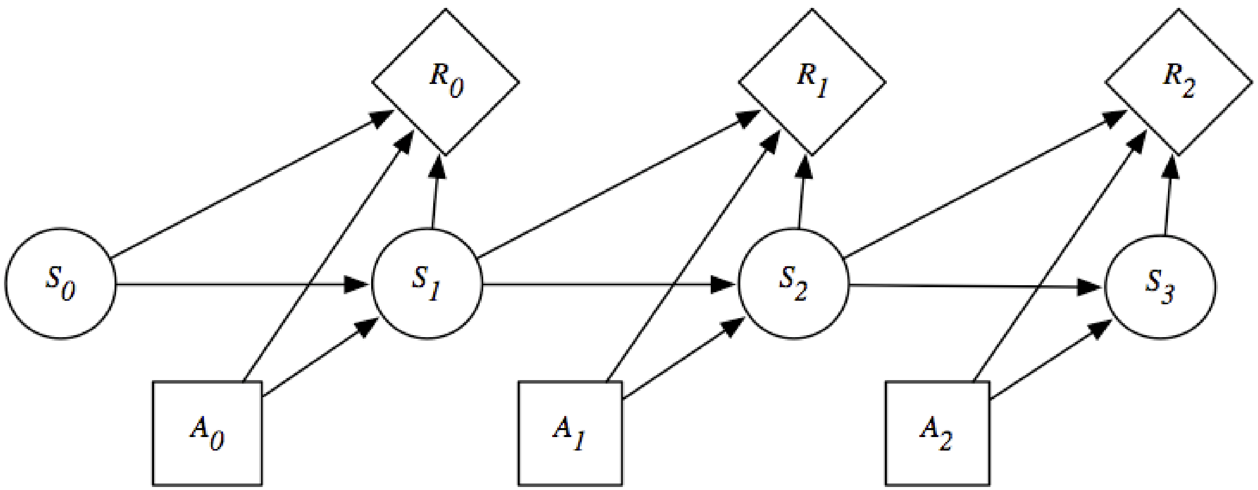
Stationary **Markov Chain**



Defined by **state transition probabilities** $P(S_{t+1}|S_t)$

- **Markovian property**: $P(S_{t+1}|S_t, S_{t-1}, \dots S_0) = P(S_{t+1}|S_t)$

Markov Decision Processes: Augments the stationary Markov chain with **actions** and **rewards**



Markov Decision Process Definition

Markov Decision Process

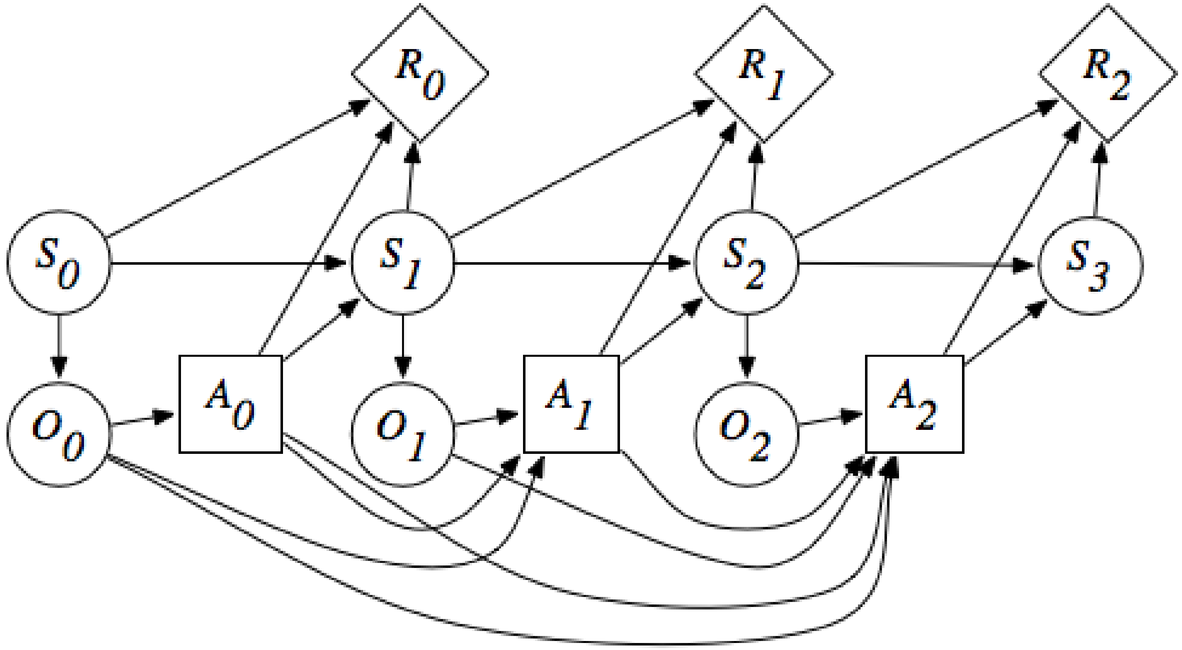
Markov decision process is a **5-tuple** $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}_{.}(.,.), \boldsymbol{R}(.), \boldsymbol{\gamma})$ where

- S is a finite set of **states.**
- A is a finite set of **actions** ($A(s)$ is the finite set of actions available from state s).
- $P(s'|s,a) = \Pr(s_{t+1} = s'|s_t = s, a_t = a)$ is the **(transition) probability** that action a in state s at time t will lead to state s' in time $t+1$.
- $R(s)$ is the **reward** the agent receives after entering state s. Reward can be positive or negative but must be bounded.
- $\gamma \in [0,1]$ is the **discount factor**, which represents the difference in importance between future rewards and present rewards.

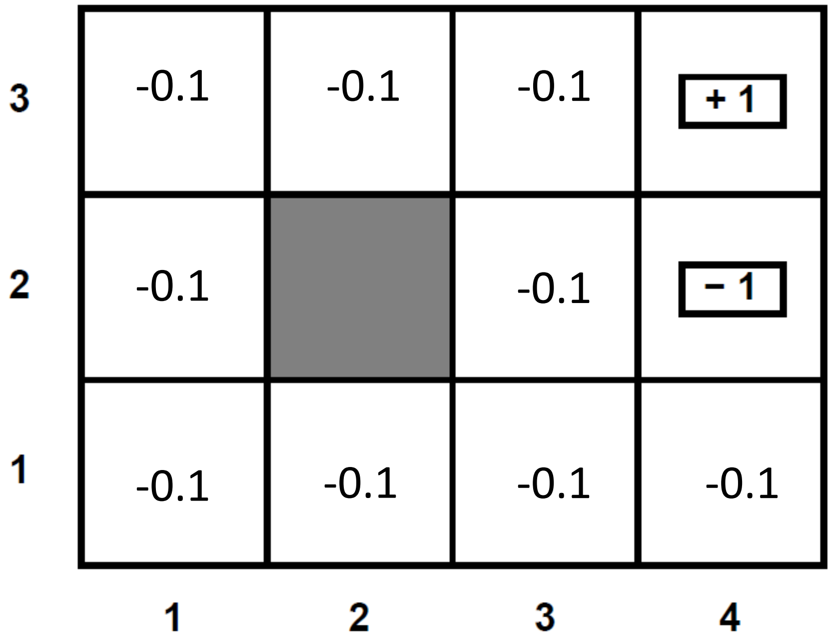
Partially Observable MDP (POMDPs)

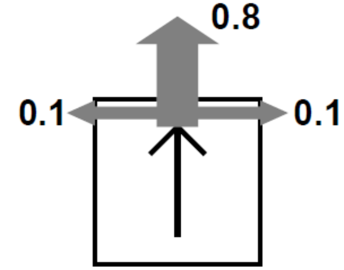
Fully-observable MDP: the agent **knows perfectly** the current state when deciding on the action.

Partially observable MDPs (POMDPs): the agent only has **noisy observation** of the current state when deciding on the action.



Grid World Example





Transition model

Utilities over Time

*How to measure the **performance of the agent**?*

Utility function is a function of environment (state) history

$$U_h([s_0, s_1, \dots, s_n])$$

What is a sensible choice of U_h?

Preference stationarity assumption assumption

If two state sequences $[s_0, s_1, s_2, \dots]$ and $[s'_0, s'_1, s'_2, \dots]$ start with the same state (i.e. $s_0 = s'_0$), then the two sequences should be preference-ordered the same way as sequences $[s_1, s_2, \dots]$ and $[s'_1, s'_2, \dots]$.

Utilities over Time

Under the stationarity assumption, there are only two **coherent ways** to assign **utilities to state sequences**

1. **Additive rewards**

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \cdots$$

2. **Discounted rewards**

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

where the **discount factor** $\gamma \in [0,1]$

*Discount factor captures the preference for **current rewards** over **future rewards**.*

Decision Horizon

Finite horizon: there is a **fixed time *D*** after which decisions does not matter

- $\forall k \geq 1 \quad U_h([s_0, s_1, \ldots, s_{D+k}]) = U_h([s_0, s_1, \ldots, s_D])$
- optimal action in a given state can change over time → optimal policy **non-stationary**

Infinite horizon: no fixed deadline

- no need to behave differently in the same state → optimal policy is **stationary**
- $\gamma < 1$: utility U_h is bounded
- $\gamma = 1$: then there needs to be **absorbing states** and the agent needs to be guaranteed to reach them (→ **proper policy**)

Absorbing/termination states: agent stays forever receiving zero reward

Policy

Stationary Policy

Stationary policy for an MDP (S, A, P, R, γ) is a function

$$\pi: S \mapsto A$$

Value of policy (from state s):

$$U^{\pi}(s) = E_{\Pr([s_0, s_1, \dots] \mid s_0 = s, \pi)}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

i.e. **"long-term" total reward** from s onwards (assuming policy π)

Optimal policy (from state s):

$$\pi_s^* = \arg\max_{\pi} U^{\pi}(s)$$

Optimal Policy

Optimal policy ***independent of the initial state*** *(under discounted rewards and infinite horizon), i.e*

$$\pi_s^* = \pi_{s'}^*, \text{ for any } s'$$

We can thus define the **utility (value) of the state**

$$U(s) = U^{\pi^*}(s)$$

Optimal policy (discounted rewards and infinite horizon)

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} U(s')P(s'|s,a)$$

For MDPs with stationary dynamics and rewards and infinite decision horizon, there ***always exists an optimum stationary policy.***

Example: Optimal Policies in the Grid World

$U(s)$

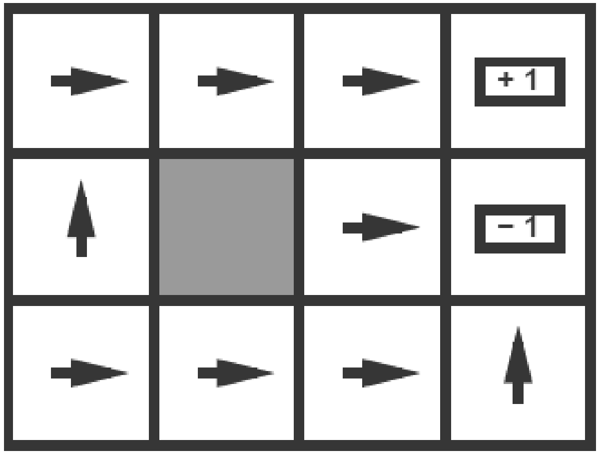
	1	2	3	4
3	0.812	0.868	0.912	+ 1
2	0.762		0.660	– 1
1	0.705	0.655	0.611	0.388

$\pi^*(s)$

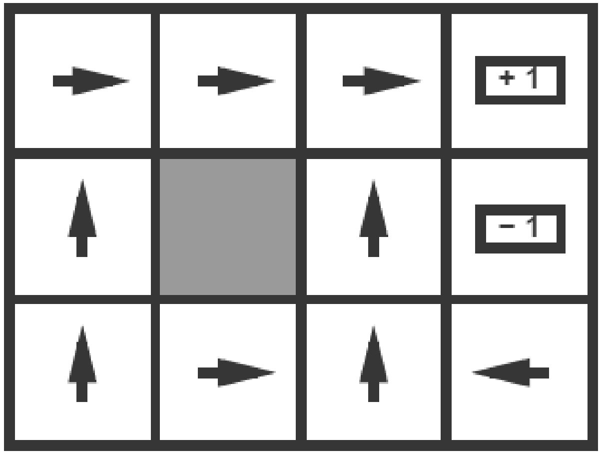
	1	2	3	4
3	→	→	→	+ 1
2	↑		↑	– 1
1	↑	←	←	←

Utilities of states and the optimal policy for $\boldsymbol{\gamma = 1}$ and $\boldsymbol{R(s) = -0.04}$ for non-terminal states

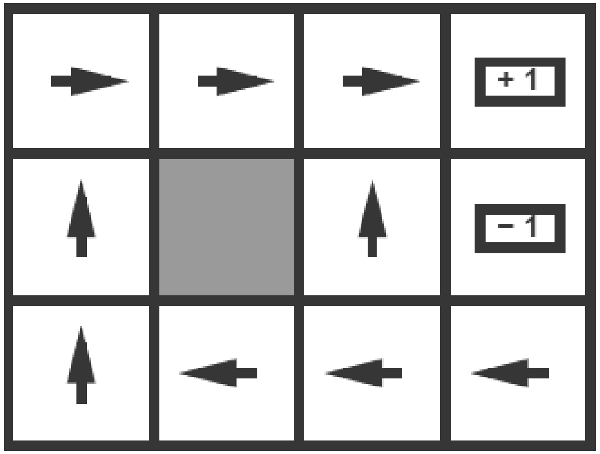
Dependence on Penalty



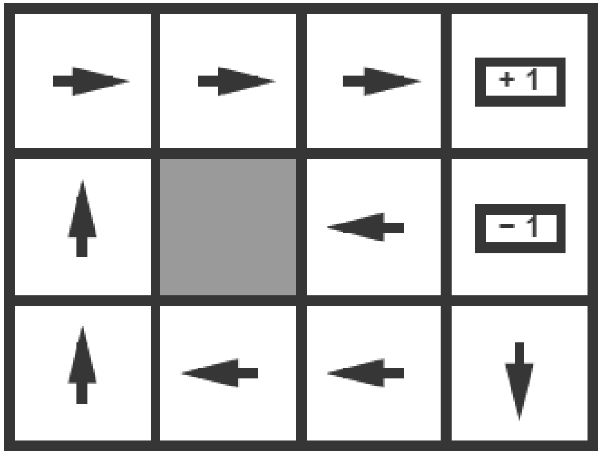
$r = [-\infty : -1.6284]$



$r = [-0.4278 : -0.0850]$



$r = [-0.0480 : -0.0274]$



$r = [-0.0218 : 0.0000]$

Solving MDPs

Markov Decision Processes

Solving MDPs

How do we find the optimum policy?

Basic (dynamic programming-based) techniques:

1. **value iteration** – compute utility $U(s)$ for each state and use is for selecting best action
2. **policy iteration** – represent policy $\pi(s)$ explicitly and update it in parallel to the utility function $U(s)$

Advanced approaches

Value Iteration

Recall the **utility* of a state**

$$U(s) = E_{\Pr([s_0, s_1, \dots] | s_0 = s, \pi^*)} \left[\sum_{t=0}^{\infty} \gamma^t R(S_t) \right]$$

I.e.

Bellman Equation (1957)

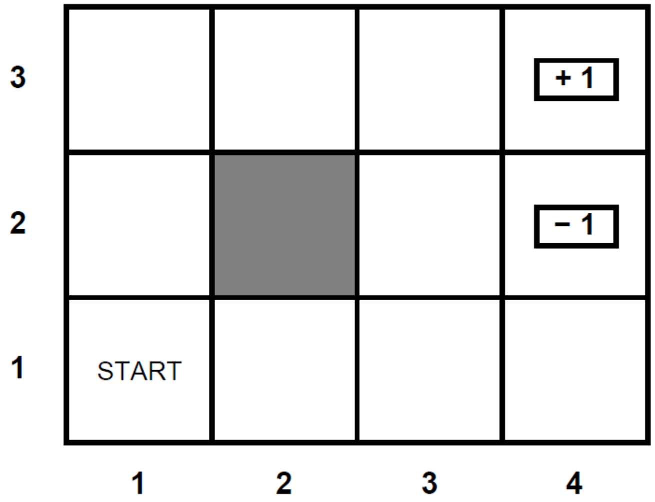
$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) U(s') \quad \forall s \in S$$

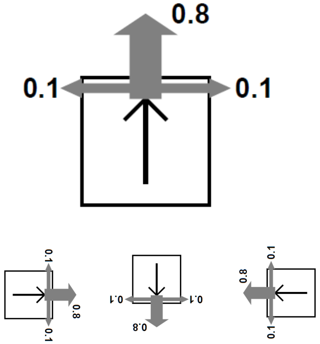
One equation per state → n **non-linear equations** for n **unknowns**

- solution is **unique**

* also termed value of a state $V(s)$

Bellman Equation Example





$\gamma = \mathbf{0.5}$ and $\boldsymbol{R(s)} = \mathbf{-0.04}$ for non-terminal states

$U(1,1) =$

Value Iteration

Analytical solution not feasible → **iterative solution**

Bellman Update (Backup)

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) U_i(s') \quad \forall s \in S$$

If applied infinitely often, **guaranteed to reach an equilibrium** and the final utility values are the **solutions** to the **Bellman equations.**

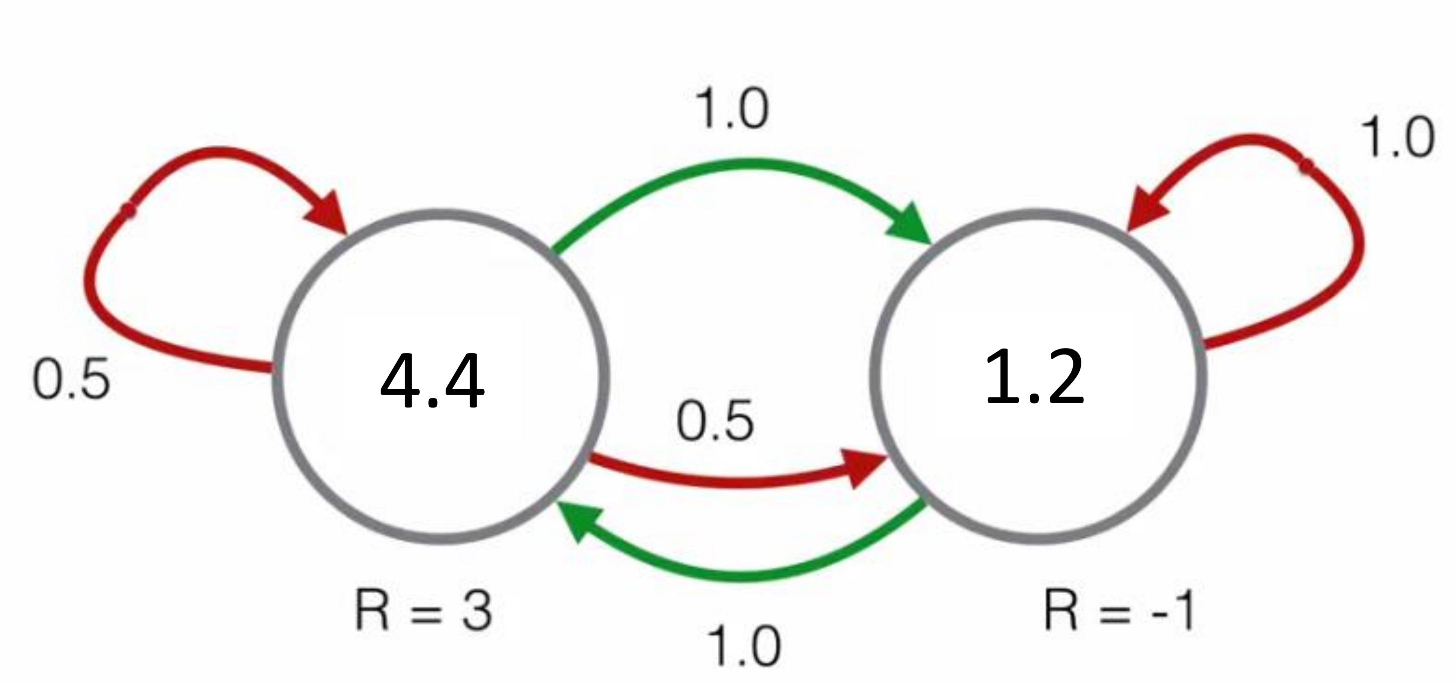
Value Iteration

function VALUE-ITERATION (*mdp*, ε)
returns a utility function
inputs:
mdp, an MDP with states S, transition model P, reward function R, discount γ
ε, the maximum error allowed in the utility of a state
local variables:
U, *U'*, vectors of utilities for states in S, initially zero
δ, the maximum change in the utility of any state in an iteration

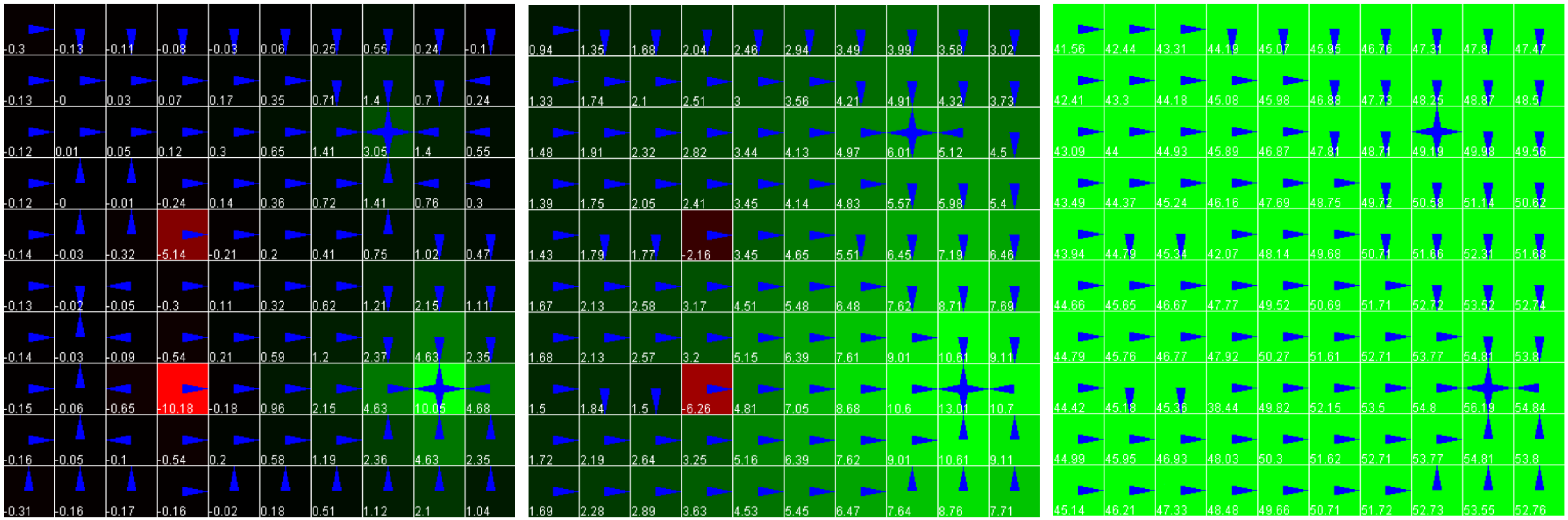
repeat
$U \leftarrow U'$; $\delta \leftarrow 0$
for each state s in S do
$U'[s] \leftarrow R[s] + \gamma max_a \sum_{s'} P(s'|s,a) U[s']$
if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
until $\delta < \varepsilon(1-\gamma)/\gamma$
return U

Value Iteration Example

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) U_i(s') \qquad \gamma = 0.5$$



Dependency on the Discount



(a) $\gamma = 0.6$ (b) $\gamma = 0.9$ (c) $\gamma = 0.99$

Four movement actions; **0.7** probability of moving in the desired direction, **0.1** in the others

R = -1 for bumping into walls; four special **rewarding states** +10 (at position (9,8); 9 across and 8 down), one worth +3 (at position (8,3)), one worth -5 (at position (4,5)) and one -10 (at position (4,8))

Convergence of Value Iteration

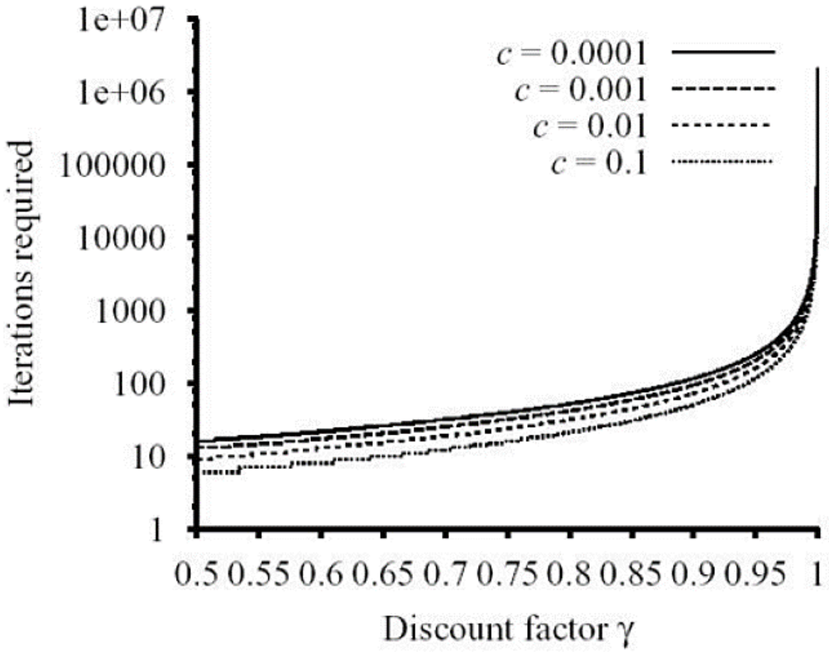
Value iteration eventually **converges** to a **unique** set of **solutions** of the Bellman equations (for $\gamma < 1$)

- Proof based on the fact that the Bellman update is an contraction on the space of utility vectors

Number of iterations for reaching an **error bound** $\|U_i - U\| < \epsilon$

$$N = \left\lceil \log\left(\frac{2R_{max}}{\epsilon(1-\gamma)}\right) / \log(1/\gamma) \right\rceil$$

For finite horizon MDPs: $|D|$ steps.



Policy Convergence

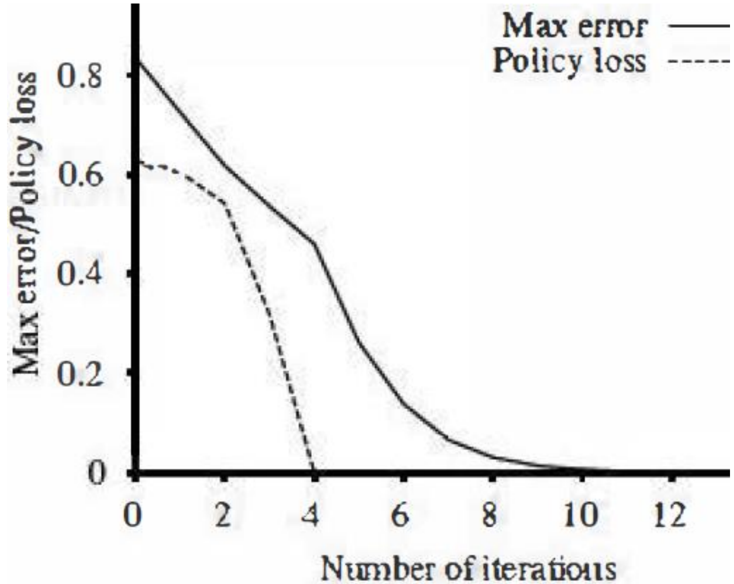
Policy loss resulting from executing a policy based on error-bounded approximate state utility function

$$\|U_i - U\| < \epsilon \Rightarrow \|U^{\pi_i} - U\| < \frac{2\epsilon\gamma}{1-\gamma}$$

→ **Termination condition**: $\|U_{i+1} - U_i\| < \epsilon(1-\gamma)/\gamma$

Policy convergence may occur **long before** utility convergence

Trade-off between **long-term** decision making and computational **cost**



Policy Iteration

Possible to get an optimal policy even when the utility estimate is inaccurate ➔ search for optimal policy and utility values **simultaneously ➔ Policy iteration**

Alternates between two steps:

1. **policy evaluation**: recalculates values of states $U_i = U^{\pi_i}$ given the current policy π_i
2. **policy improvement/iteration:** calculates a new maximum expected utility policy π_{i+1} using one-step look-ahead based on U_i

Terminates when the policy improvement step yields no change in the utilities.

Policy Evaluation

Simplified Bellman Equation

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s))U_i(s') \quad \forall s \in S$$

The equations are now **linear** ➔ can be solved in $O(n^3)$

Policy Iteration Algorithm

function POLICY-ITERATION (*mdp*)
returns a policy
  inputs:
    mdp, an MDP with states S, transition model P
  local variables:
    U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, *mdp*)
    unchanged? $\leftarrow$ true
    for each state s in S do
      if $\max_a \sum_{s'} P(s'|s,a) U[s'] > \sum_{s'} P(s'|s,\pi(s)) U[s']$ **then**
        $\pi(s) \leftarrow argmax_a \sum_{s'} P(s'|s,a) U[s']$
        unchanged? $\leftarrow$ false
  until *unchanged?*
  return π

Modified Policy Iteration

Policy iteration often converges in few iterations but each iteration is **expensive**.

Main idea: use **iterative approximate** policy evaluation.

Simplified Bellman Update

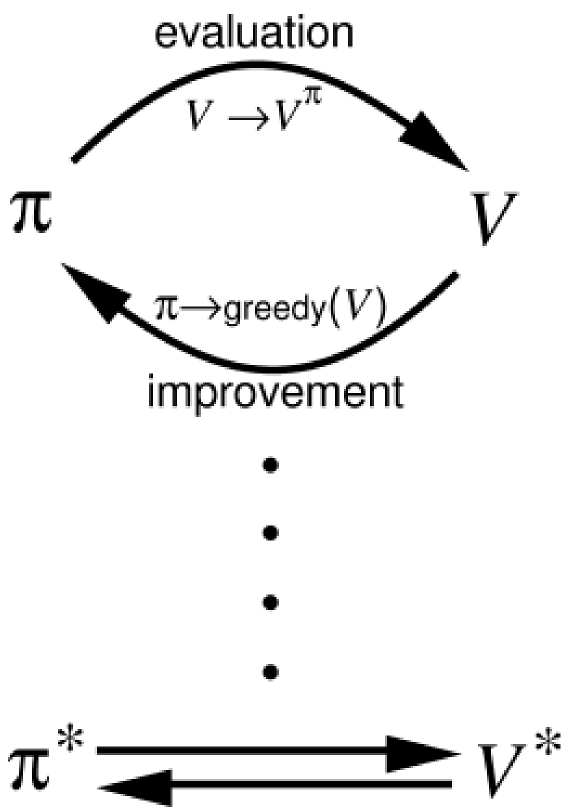
$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s') \quad \forall s \in S$$

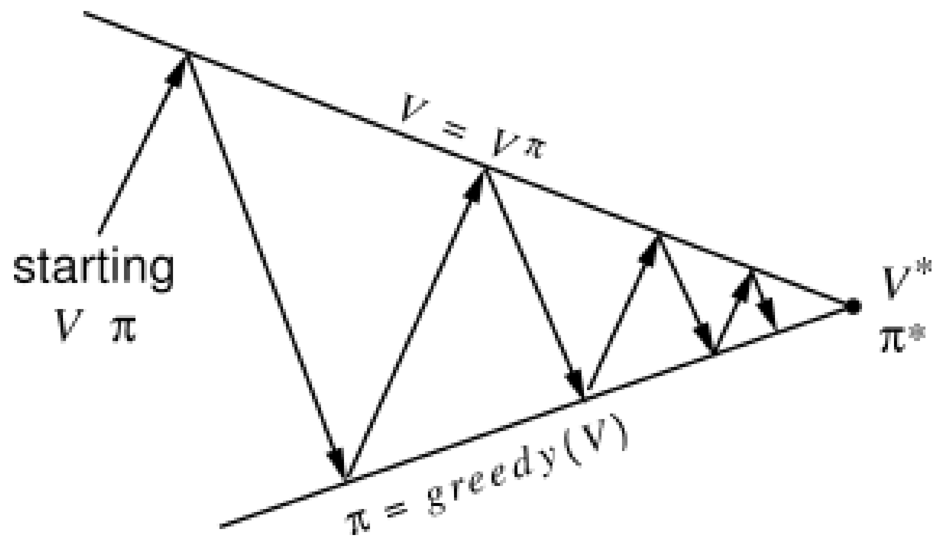
- Use *a few* steps of value iteration (with π fixed)
- Start from the value function produced in the last iteration

Often **converges much faster** than pure value iteration or policy iteration (combines the strength of both approaches).

Generalized Policy Iteration

$V \sim U$





Note: Value iteration is a special case of modified / generalized policy iteration

Asynchronous Policy Iteration

Previous algorithms required updating values of utilities and policies in all states.

This is not necessary ➔ **Asynchronous policy iteration**

- pick *any* subset of states and apply *either* kind of updating (policy improvement or simplified value iteration)
- still **guaranteed to converge** under certain assumptions

Enables much more general **asynchronous heuristic algorithms** (e.g. prioritized sweeping).

Efficiency of DP-based Approaches

DP methods take to find an optimal policy is **polynomial in the number of states and actions**.

- A DP method is guaranteed to find an optimal policy in polynomial time even though the total number of (deterministic) policies is $|S|^{|A|}$.
- DP is **exponentially faster** than any **direct search** in policy space could be.

Both policy iteration and value iteration are used, and it is not clear which, if either, is better in general.

On problems with large state spaces, **asynchronous DP methods** are often preferred.

Advanced Solution Techniques

Markov Decision Processes

Speeding-up MDP Search

1. **Prioritizing updates** based on the estimation of which updates have the largest impact.
2. **Prunning the state space** based on the knowledge of the **initial state**.
3. **Prioritizing updates** based on the additional knowledge in the form of a **heuristic function**.

Prioritized Value Iteration

Many Bellman updates do not change the utility function

Idea: Prioritize Bellman updates – prefer those that have most impact.

When the utility $U(s')$ of no successor s' of a state s has been updated since the last update of s, we don't need to update $U(s)$.

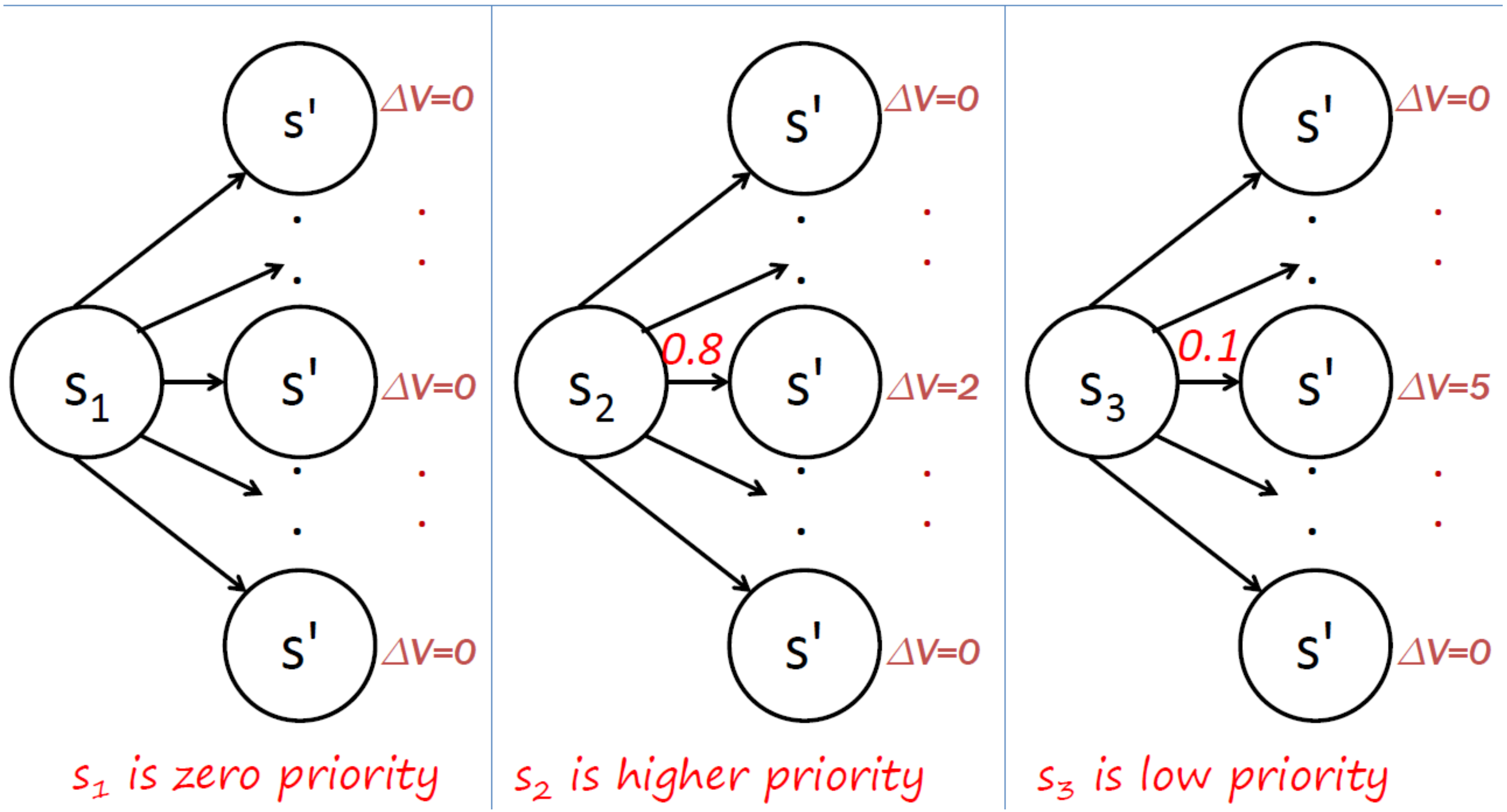
Prioritized Value Iteration Algorithm

Algorithm 3.5: Prioritized Value Iteration

initialize V
initialize priority queue q
repeat
    select state $s' = q.pop()$
    compute $V(s')$ using a Bellman backup at s'
    foreach *predecessor s of* s', *i.e.*, $\{s \mid \exists a[\mathcal{T}(s, a, s') > 0]\}$ **do**
        compute priority(s)
        $q.push(s, \text{priority}(s))$
    end
until *termination*;
return greedy policy π^V

Many ways to set the update priority…

How to Prioritize



Prioritized Sweeping

Idea: Estimate the **expected change** in the **utility** of a state if an update was performed at it now, and treats this as the priority of a state.

Let $\Delta U(s')$ denote the change in the utility of $U(s')$ after its latest Bellman update. Then

$$\text{priority}_{PS}(s) \leftarrow \max\left\{\text{priority}_{PS}(s), \max_{a \in \mathrm{A}}\{P(s'|s,a)\Delta U(s')\}\right\}$$

Priority sweeping **converges** the optimal utility function if the initial priority values are non-zero for all states $s \in S$.

Other variants: **Improved prioritized sweeping** – takes into account proximity to high-reward (goal) states.

Heuristic Algorithms for MDPs

Storing and updating utility and policy values for all state prohibitively expensive.

- polynomial in the number of states (but this can be huge!)

Idea: Do not compute utility and policy function for all states

Two pieces of information can drastically reduce the amount of computation resources needed.

1. **MDP's initial state**: a policy closed w.r.t. initial state often excludes large parts of the state space.
2. **Heuristic function**: a prior knowledge that helps us assess the quality of different states in MDPs

FIND-and-REVISE Algorithm

Greedy graph $G_{s_0}^{U'}$: all states that can be reach from s_0 by any policy that is greedy w.r.t to U' and closed w.r.t. s_0.

Residual $Res^{U'}(s) = \left|U'(s) - R(s) - \max_a \sum_{s'} P(s'|s,a)U'(s')\right|$

Algorithm: FIND-and-REVISE

Start with a heuristic value function $U \leftarrow h$

while U's greedy graph $G_{s_0}^{U'}$ contains a state s with $Res^{U'}(s) > \epsilon$ **do**

  FIND a state s in $G_{s_0}^{U'}$ with $Res^{U'}(s) > \epsilon$

  REVISE $U'(s)$ with a Bellman update

end

return $\pi^{U'}$

Other Topics

More **expressive representations**

- Factored/Relational MDDPs
- PPDDL and RDDL (Relational Domain Definition Language)
- …..

Other solution techniques:

- Real-time dynamic programming
- Monte Carlo-based techniques
- ….

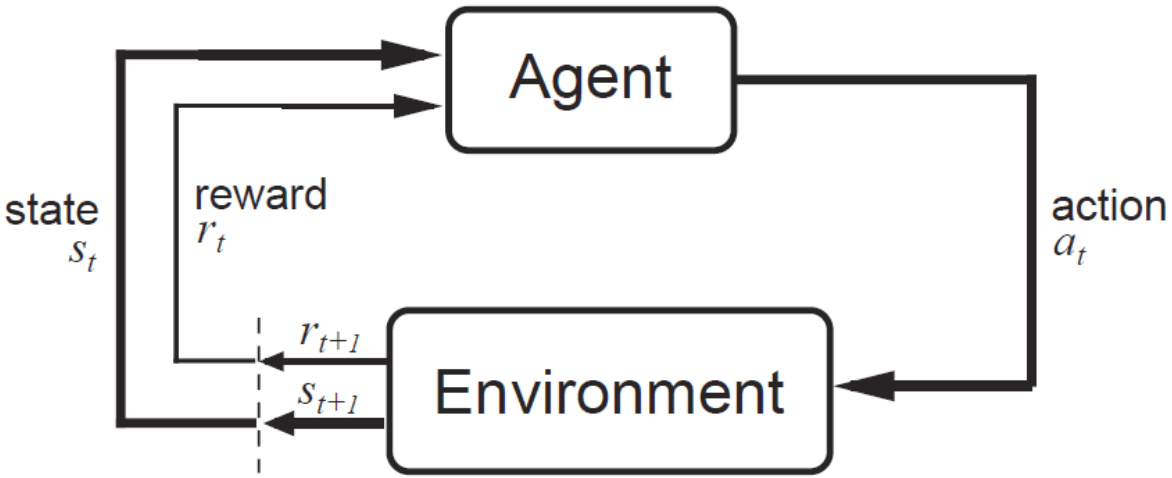
Concluding Remarks

MDPs are a very powerful model for sequential decision making…

…but with strong assumptions

1. **States fully observable:** generalization ➔ **partially observable MDPs** (next lecture)
2. **State transition and reward model known:** generalization ➔ **reinforcement learning**

Reinforcement Learning



Reinforcement learning (RL) is based on MDPs but **transition and reward models not known**.

MDP computes an optimal policy. **RL learns** an optimum policy.

Key ingredient: **exploitation vs. exploration** control**.**

Model-based vs. **model-free** approaches.

Summary

MDPs **generalize** deterministic **state space search** to **stochastic** environments.

MDPs are a foundation for probabilistic planning.

An **optimum policy** associates an action with (every) state.

Basic dynamic programming-based solution techniques: **value iteration** and **policy iteration**.

Advanced techniques based **intelligent prioritization** of **asynchronous update**.

Very active area of research (and progress).

Reading:

- Russel and Norvig: Artificial Intelligence: Modern Approach, 2010, Sections 17.1-17.3.
- Mausam and Kolobov: Planning with Markov Decision Processes: An AI Perspective, 2012 (advanced)