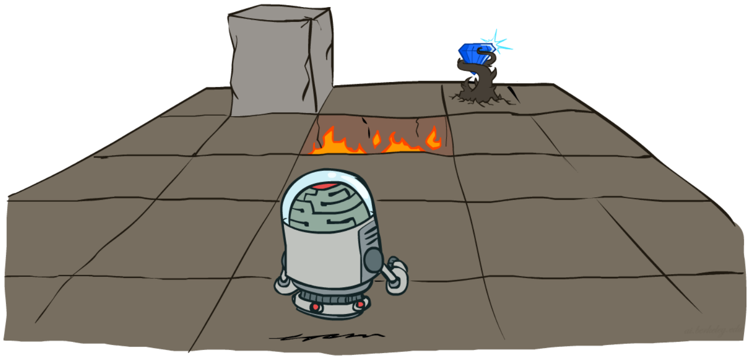
Introduction to Artificial Intelligence

Lecture 8: Making decisions

Prof. Gilles Louppe
g.louppe@uliege.be

LIÈGE université

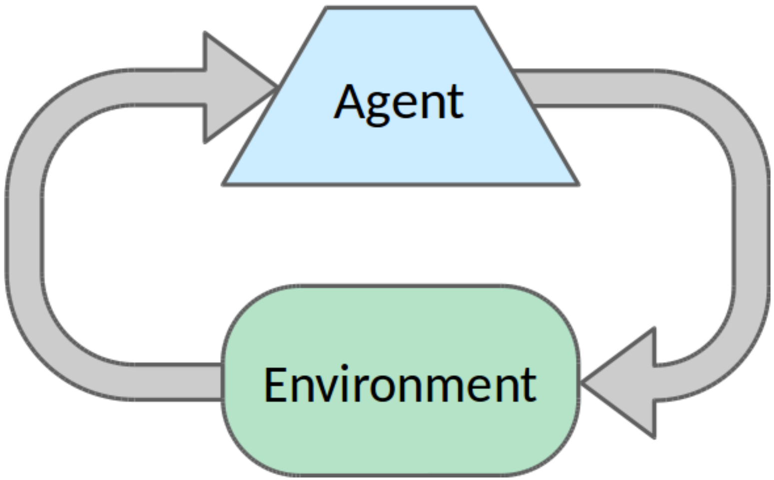
Today



Reasoning under uncertainty and taking decisions:

- Markov decision processes
  - MDPs
  - Bellman equation
  - Value iteration
  - Policy iteration
- Partially observable Markov decision processes

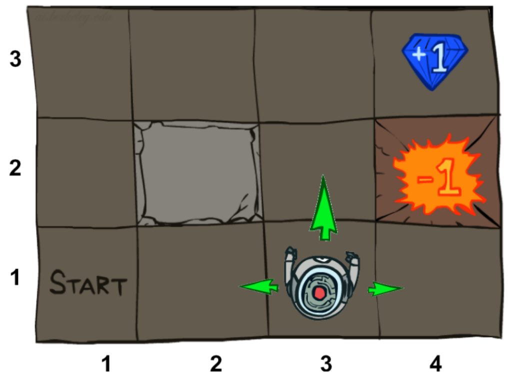
Image credits: CS188, UC Berkeley.


Agent
Environment

Grid world

Assume our agent lives in a 4×3 grid environment.

- Noisy movements: actions do not always go as planned.
    - Each action achieves the intended effect with probability 0.8.
    - The rest of the time, with probability 0.2, the action moves the agent at right angles to the intented direction.
    - If there is a wall in the direction the agent would have been taken, the agent stays put.
- The agent receives rewards at each time step.
    - Small 'living' reward each step (can be negative).
    - Big rewards come at the end (good or bad).

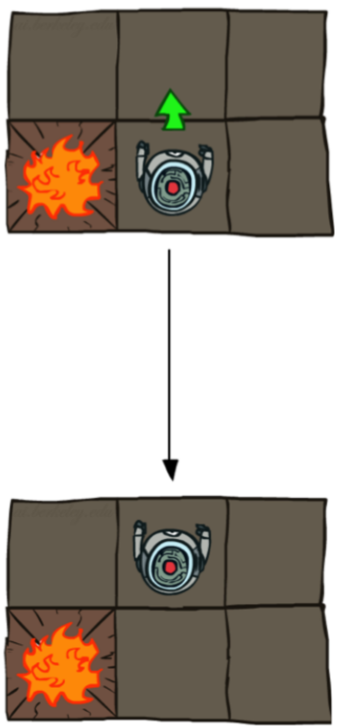


Goal: maximize sum of rewards.

Image credits: CS188, UC Berkeley.

Deterministic actions

Stochastic actions



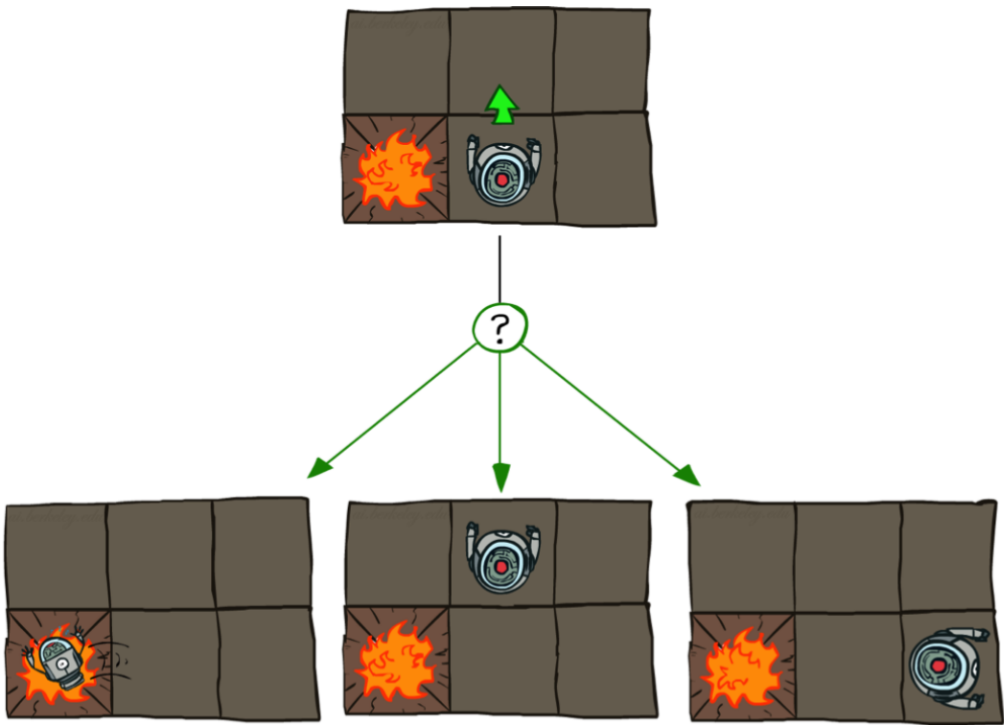


Image credits: CS188, UC Berkeley.

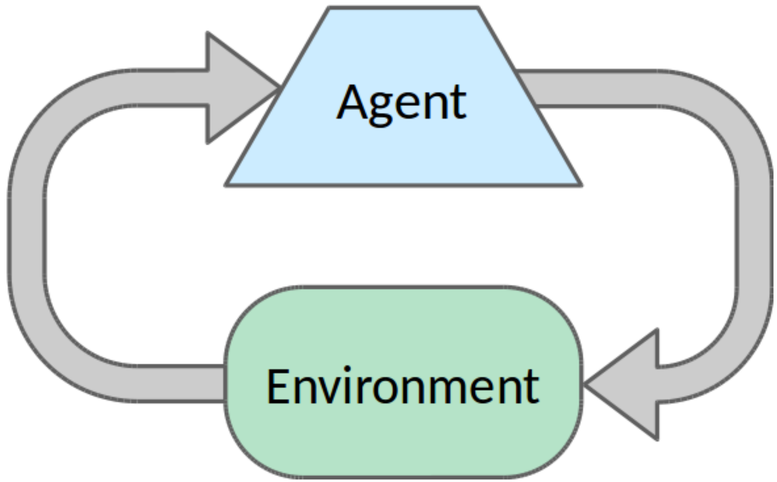
Markov decision processes

Markov decision processes

A Markov decision process (MDP) is a 5-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ such that:

- $\mathcal{S}$ is a set of states s;
- $\mathcal{A}$ is a set of actions a;
- P is a transition model such that $P(s'|s, a)$ denotes the probability of reaching state s' if action a is done in state s;
- R is reward function that maps immediate (finite) reward values $R(s)$ obtained in states s;
- $0 < \gamma \leq 1$ is a discount factor, which represents the difference in importance between future and present rewards.

s_t

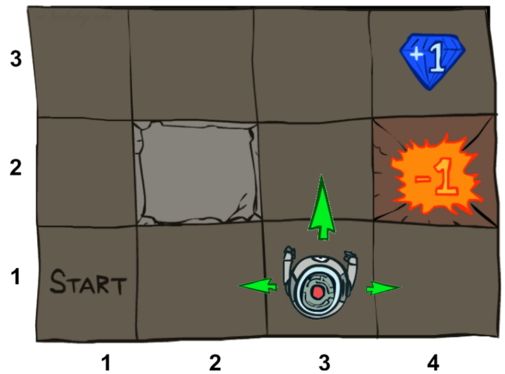


s_{t+1}

$r_{t+1} = R(s_{t+1})$

a_t

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



Example

- $s \in \mathcal{S}$: locations (i, j) on the grid.
- $a \in \mathcal{A}$: $[\text{Up}, \text{Down}, \text{Right}, \text{Left}]$.
- Transition model: $P(s'|s, a)$
- Reward:

$$R(s) = \begin{cases} -0.03 & \text{for non-terminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

- $\gamma = 0.9$.

Image credits: CS188, UC Berkeley.

What is Markovian about MDPs?

Given the present state, the future and the past are independent:

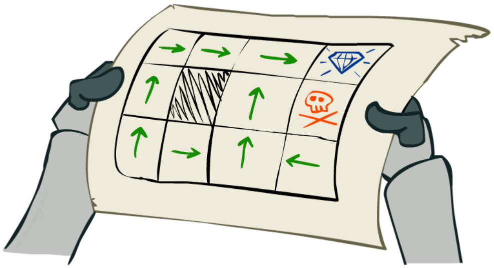
$$P(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, ..., S_0) = P(S_{t+1}|S_t, A_t)$$

This is similar to search problems, where the successor function could only depend on the current state.

Andrey Markov

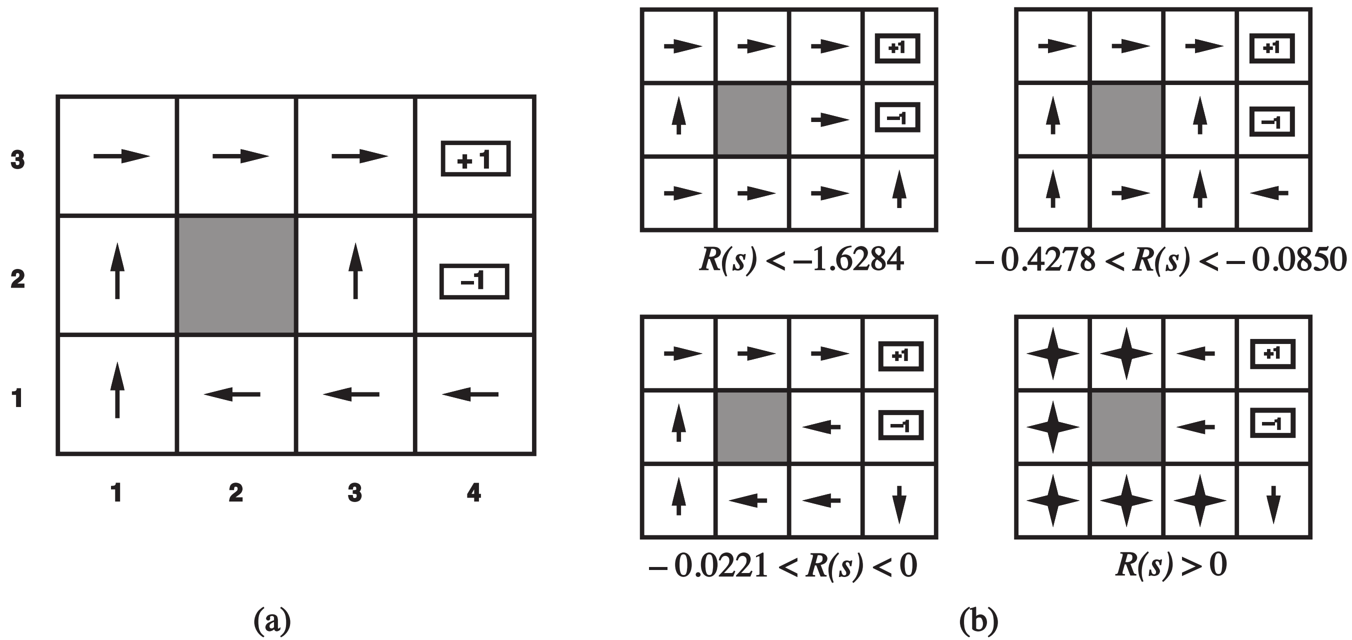
Policies

- In deterministic single-agent search problems, our goal was to find an optimal plan, or sequence of actions, from start to goal.
- For MDPs, we want to find an optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$.
  - A policy π maps actions to states.
  - An optimal policy is one that maximizes the expected utility, e.g. the expected sum of rewards.
  - An explicit policy defines a reflex agent.
- Expectiminimax did not compute entire policies, but only some action for a single state.



Optimal policy when $R(s) = -0.03$ for all non-terminal states s.

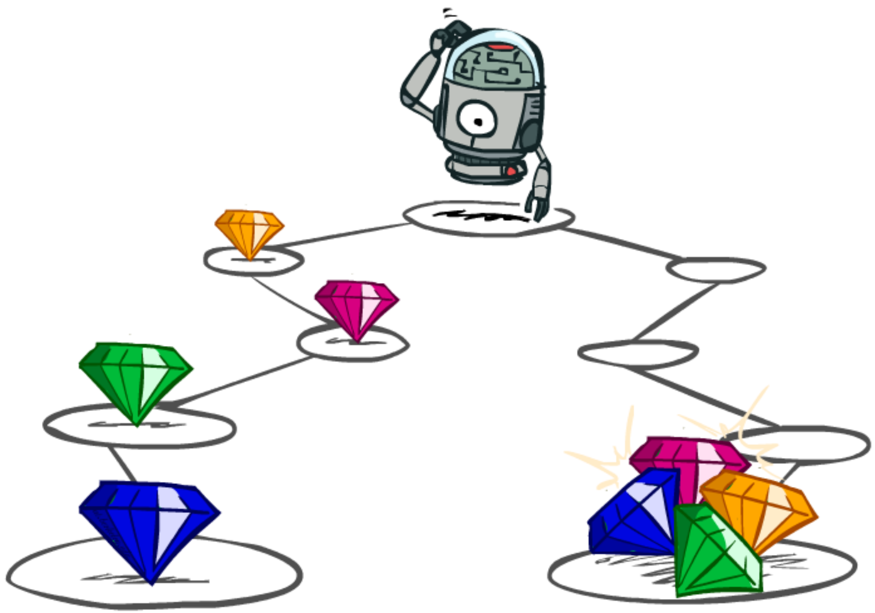
Image credits: CS188, UC Berkeley.



(a) Optimal policy when $R(s) = -0.04$ for all non-terminal states s. (b) Optimal policies for four different ranges of $R(s)$.

Depending on $R(s)$, the balance between risk and reward changes from risk-taking to very conservative.

Utilities over time



What preferences should an agent have over state or reward sequences?

- More or less? $[2, 3, 4]$ or $[1, 2, 2]$?
- Now or later? $[1, 0, 0]$ or $[0, 0, 1]$?

Image credits: CS188, UC Berkeley.

Theorem

If we assume stationary preferences over reward sequences, i.e. such that

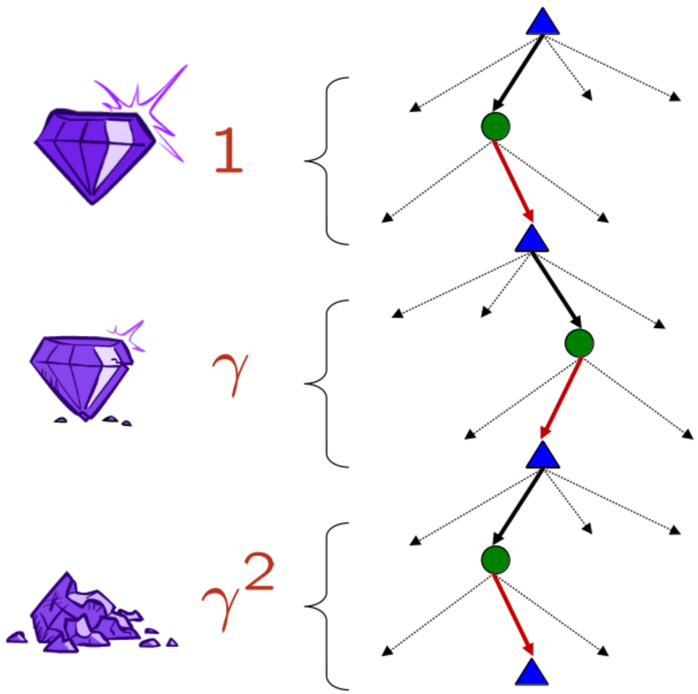
$$[r_0, r_1, r_2, \ldots] \succ [r_0, r'_1, r'_2, \ldots] \Leftrightarrow [r_1, r_2, \ldots] \succ [r'_1, r'_2, \ldots],$$

then there are only two coherent ways to assign utilities to sequences:

Additive utility:	$V([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \ldots$
Discounted utility: ($0 < \gamma < 1$)	$V([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 r + \ldots$

Discounting

- Each we time we transition to the next state, we multiply in the discount once.
- Why discount?
    - Sooner rewards probably do have higher utility than later rewards.
    - Will help our algorithms converge.



Example: discount $\gamma = 0.5$

- $V([1,2,3]) = 1 + 0.5 \times 2 + 0.25 \times 3$
- $V([1,2,3]) < V([3,2,1])$

Image credits: CS188, UC Berkeley.

Infinite sequences

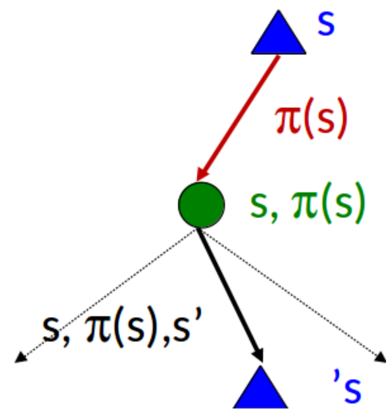
What if the agent lives forever? Do we get infinite rewards? Comparing reward sequences with $+\infty$ utility is problematic.

Solutions:

- Finite horizon: (similar to depth-limited search)
  - Terminate episodes after a fixed number of steps T.
  - Results in non-stationary policies (π depends on time left).
- Discounting (with $0 < \gamma < 1$):

$$V([r_0, r_1, ..., r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{max}}{1-\gamma}$$

  Smaller γ results in a shorter horizon.
- Absorbing state: guarantee that for every policy, a terminal state will eventually be reached.



Policy evaluation

The expected utility obtained by executing π starting in s is given by

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right],$$

where the expectation is with respect to the probability distribution over state sequences determined by s and π.

Image credits: CS188, UC Berkeley.

Optimal policies

Among all policies the agent could execute, the optimal policy is the policy π_s^* that maximizes the expected utility:

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

Because of discounted utilities, the optimal policy is independent of the starting state s. Therefore we simply write π^*.

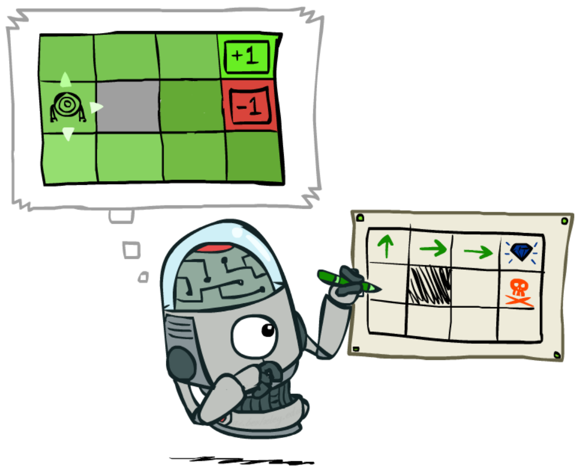
Utilities of states

The utility (or value) $V(s)$ of a state is now simply defined as $V^{\pi^*}(s)$.

- That is, the expected (discounted) reward if the agent executes an optimal policy starting from s.
- Notice that $R(s)$ and $V(s)$ are quite different quantities:
    - $R(s)$ is the short term reward for having reached s.
    - $V(s)$ is the long term total reward from s onward.

3	0.812	0.868	0.918	+1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Utilities of the states in Grid World, calculated with $\gamma = 1$ and $R(s) = -0.04$ for non-terminal states.



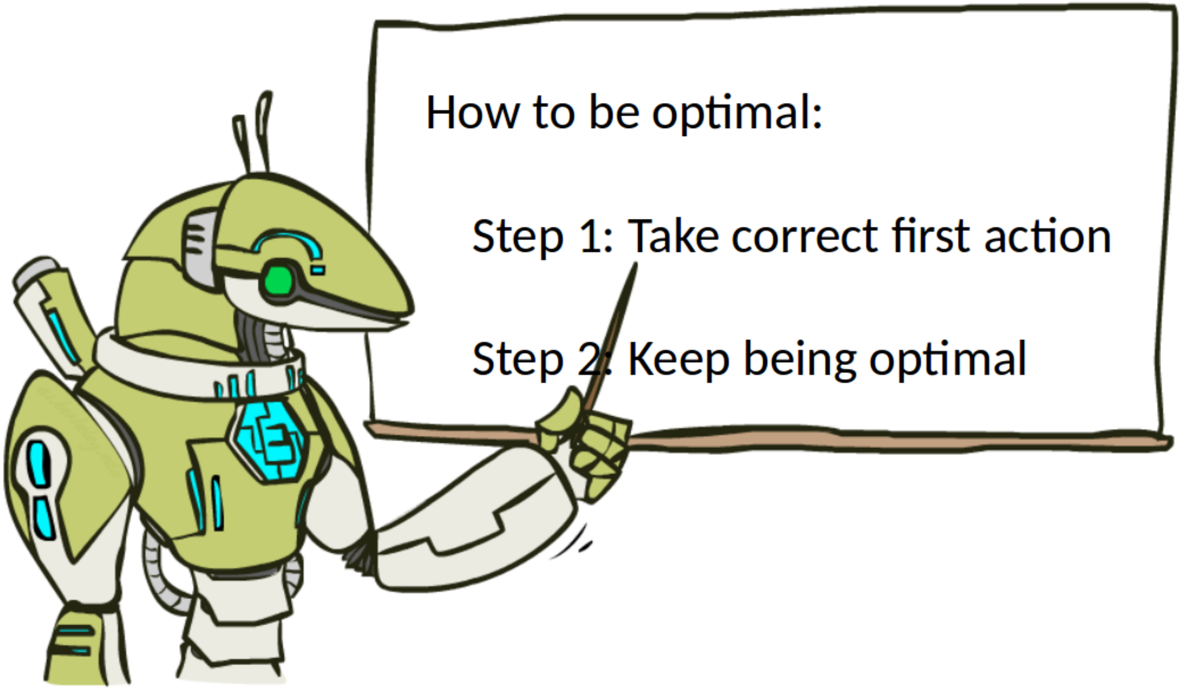
Policy extraction

Using the principle of maximum expected utility (MEU), the optimal action maximizes the expected utility of the subsequent state. That is,

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)V(s').$$

Therefore, we can extract the optimal policy provided we can estimate the utilities of states.

Image credits: CS188, UC Berkeley.


How to be optimal:
Step 1: Take correct first action
Step 2: Keep being optimal

Bellman equation

There is a direct relationship between the utility of a state and the utility of its neighbors:

The utility of a state is the immediate reward for that state, plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action.

That is,

$$V(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V(s').$$

- These equations are called the Bellman equations. They form a system of $n = |\mathcal{S}|$ non-linear equations with as many unknowns.
- The utilities of states, defined as the expected utility of subsequent state sequences, are solutions of the set of Bellman equations.

Example

$$\begin{aligned} V(1,1) = -0.04 + \gamma \max[&0.8V(1,2) + 0.1V(2,1) + 0.1V(1,1), \\ &0.9V(1,1) + 0.1V(1,2), \\ &0.9V(1,1) + 0.1V(2,1), \\ &0.8V(2,1) + 0.1V(1,2) + 0.1V(1,1)] \end{aligned}$$

Value iteration

Because of the $\max$ operator, the Bellman equations are non-linear and solving the system is problematic.

The value iteration algorithm provides a fixed-point iteration procedure for computing the state utilities $V(s)$:

- Let $V_i(s)$ be the estimated utility value for s at the i-th iteration step.
- The Bellman update consists in updating simultaneously all the estimates to make them locally consistent with the Bellman equation:

$$V_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a) V_i(s')$$

- Repeat until convergence.

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
rewards $R(s)$, discount γ
ϵ, the maximum error allowed in the utility of any 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**
$$U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$$
if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
until $\delta < \epsilon(1-\gamma)/\gamma$
return U

Convergence

Let $||V|| = \max_s |V(s)|$ be the max-norm of the vector of utilities, such that $||V - V'||$ is the maximum difference between any two corresponding elements of V and V'.

Let V_i and V_{i+1} be successive approximations to the true utility V.

Theorem. For any two approximations V_i and V_i',

$$||V_{i+1} - V_{i+1}'|| \leq \gamma ||V_i - V_i'||.$$

- That is, the Bellman update is a contraction by a factor γ on the space of utility vector.
- Therefore, any two approximations must get closer to each other, and in particular any approximation must get closer to the true V.

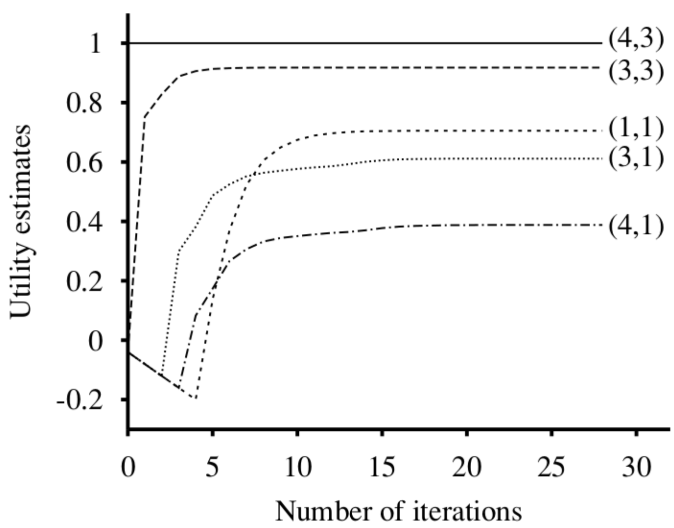
$\Rightarrow$ Value iteration always converges to a unique solution of the Bellman equations whenever $\gamma < 1$.

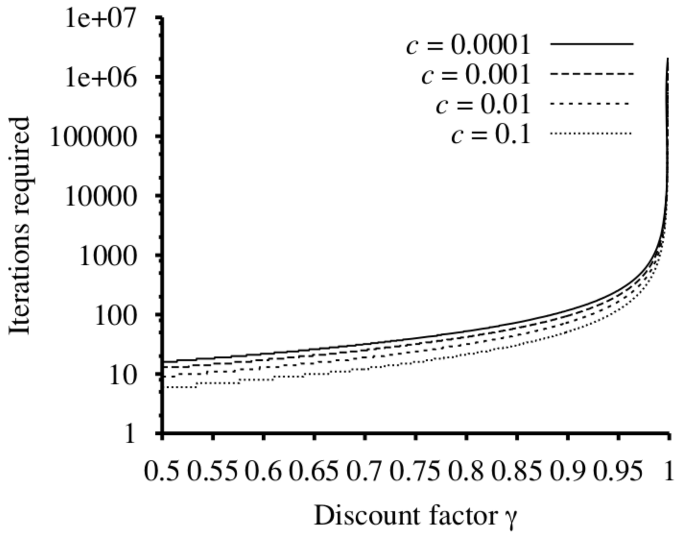
Performance

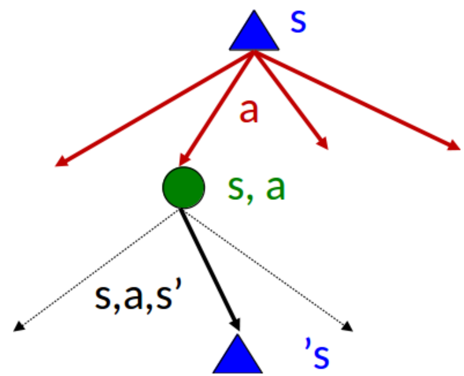
Since $||V_{i+1} - V|| \leq \gamma ||V_i - V||$, the error is reduced by a factor of at least γ at each iteration.

Therefore, value iteration converges exponentially fast:

- The maximum initial error is $||V_0 - V|| \leq 2R_{\max}/(1-\gamma)$.
- To reach an error of at most ϵ after N iterations, we require $\gamma^N 2R_{\max}/(1-\gamma) \leq \epsilon$.







Problems with value iteration

Value iteration repeats the Bellman updates:

$$V_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V_i(s')$$

- Problem 1: it is slow – $O(|\mathcal{S}|^2|\mathcal{A}|)$ per iteration.
- Problem 2: the $\max$ at each state rarely changes.
- Problem 3: the policy π_i extracted from the estimate V_i might be optimal even if V_i is inaccurate!

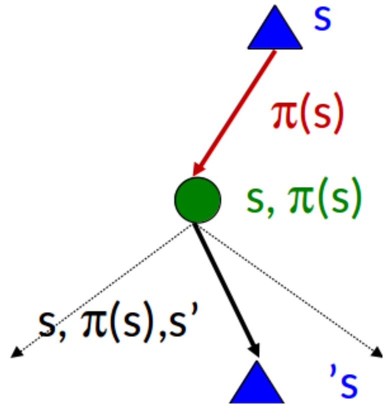
Policy iteration

The policy iteration algorithm instead directly computes the policy (instead of state values). It alternates the following two steps:

- Policy evaluation: given π_i, calculate $V_i = V^{\pi_i}$, i.e. the utility of each state if π_i is executed.
- Policy improvement: calculate a new policy π_{i+1} using one-step look-ahead based on V_i:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} P(s'|s,a)V_i(s')$$

This algorithm is still optimal, and might converge (much) faster under some conditions.



Policy evaluation

At the i-th iteration we have a simplified version of the Bellman equations that relate the utility of s to the utilities of its neighbors:

$$V_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) V_i(s')$$

These equations are now linear because the $\max$ operator has been removed.

- for n states, we have n equations with n unknowns;
- this can be solved exactly in $O(n^3)$ by standard linear algebra methods.

In some cases $O(n^3)$ is too prohibitive. Fortunately, it is not necessary to perform exact policy evaluation. An approximate solution is sufficient.

One way is to run k iterations of simplified Bell updates:

$$V_{i+1}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) V_i(s)$$

This hybrid algorithm is called modified policy iteration.

function POLICY-ITERATION(mdp) **returns** a policy
inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
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 $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
$\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
$unchanged? \leftarrow$ false
until $unchanged?$
return π

Partially observable Markov decision processes

POMPDs

What if the environment is partially observable?

- The agent does not know in which state s it is in.
  - Similarly, it cannot evaluate the reward $R(s)$ associated to the unknown state.
  - Therefore, it makes no sense to talk about a policy $\pi(s)$.
- Instead, the agent maintains a belief state $b(s)$ (a probability distribution over states) which it can update with the evidence e it collects.
  - This is filtering!

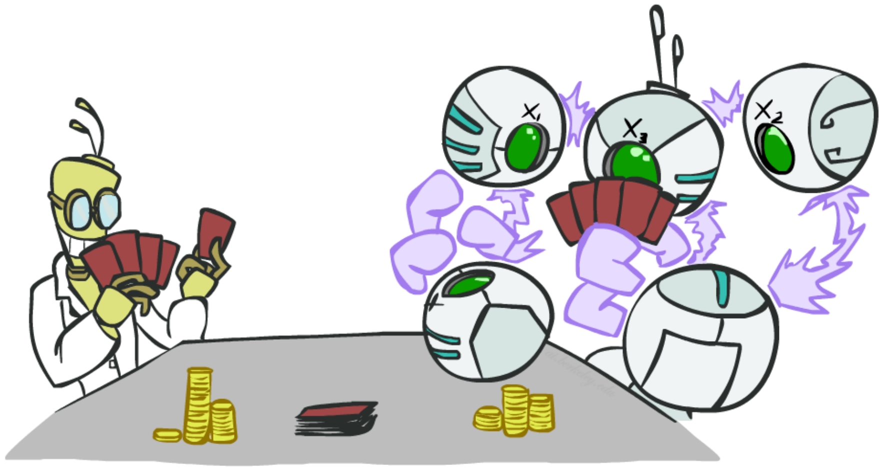


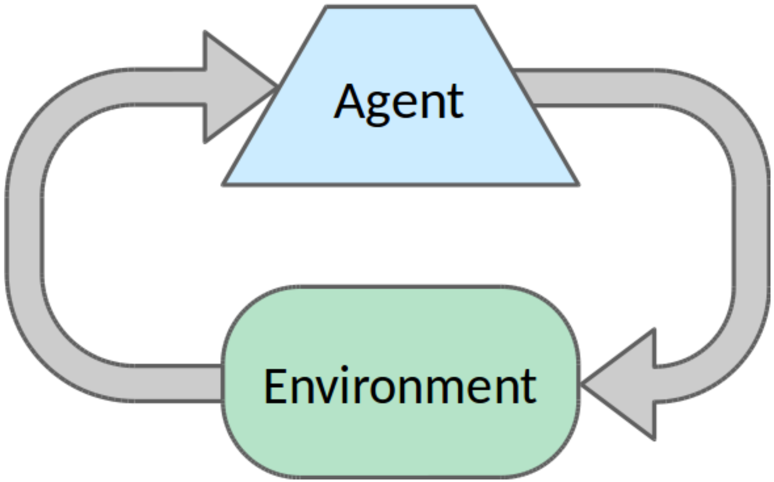
Image credits: CS188, UC Berkeley.

A partially observable Markov decision process (POMDP)

- has the same elements as an MDP,
- but adds a sensor model $P(e|s)$.

$$b_t(s) = P(s_t | a_{1:t-1}, e_{1:t})$$

e_{t+1}



a_t

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

$$e_{t+1} \sim P(e_{t+1} | s_{t+1})$$

Belief MDP

Theorem (Astrom, 1965). The optimal action depends only on the agent's current belief state.

- The optimal policy can be described by a mapping $\pi^*(b)$ from beliefs to actions.
- It does not depend on the actual state the agent is in.

In other words, POMDPs can be reduced to an MDP in belief-state space, provided we can define a transition model $P(b'|b, a)$ and a reward function ρ over belief states.

If $b(s)$ was the previous belief state and the agent does action a and perceives e, then the new belief state is given by

$$b'(s') = \alpha P(e|s') \sum_s P(s'|s,a) b(s) = \alpha \, \text{forward}(b,a,e).$$

Therefore,

$$\begin{aligned} P(b'|b,a) &= \sum_e P(b'|b,a,e) P(e|b,a) \\ &= \sum_e P(b'|b,a,e) \sum_{s'} P(e|b,a,s') P(s'|b,a) \\ &= \sum_e P(b'|b,a,e) \sum_{s'} P(e|s') \sum_s P(s'|s,a) b(s) \end{aligned}$$

where $P(b'|b,a,e) = 1$ if $b' = \text{forward}(b,a,e)$ and 0 otherwise.

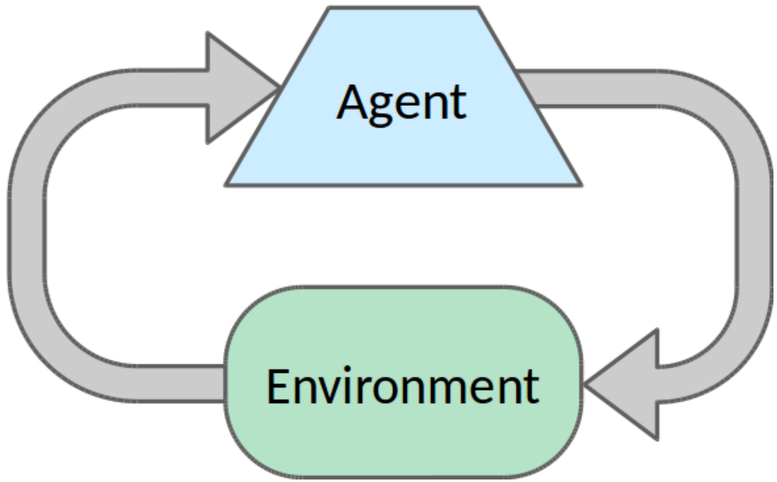
We can also define a reward function for belief states as the expected reward for the actual state the agent might be in:

$$\rho(b) = \sum_s b(s)R(s)$$

b_t

b_{t+1}

ρ_{t+1}



a_t

$$b_{t+1} \sim P(b_{t+1} | b_t, a_t)$$

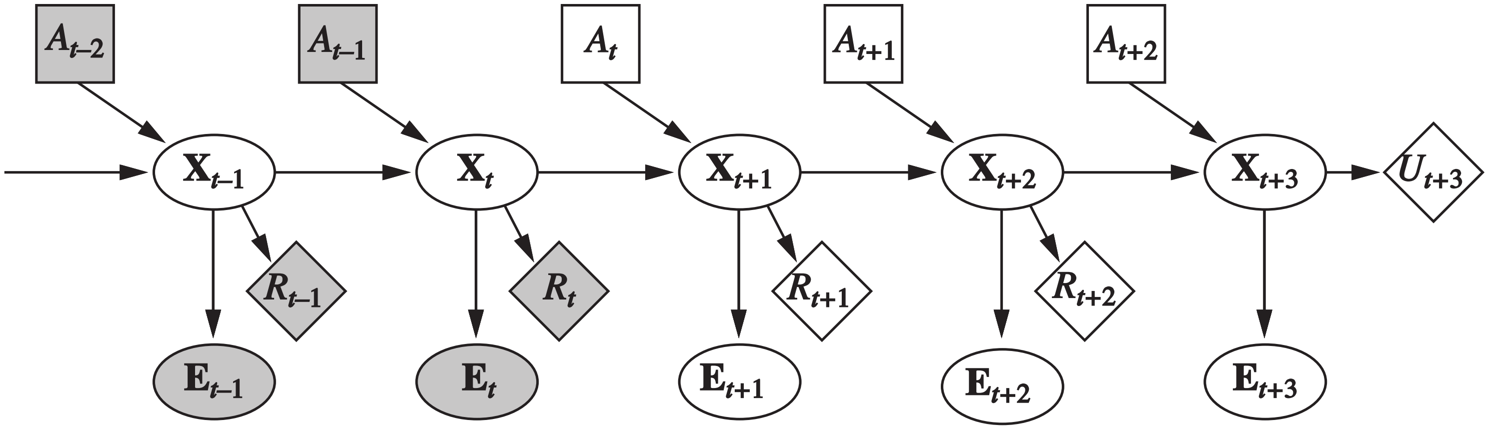
Although we have reduced POMDPs to MPDs, the Belief MDP we obtain has a continuous (and usually high-dimensional) state space.

- None of the algorithms described earlier directly apply.
- In fact, solving POMDPs is very (actually, PSPACE-)hard!

Online agents for POMDPs

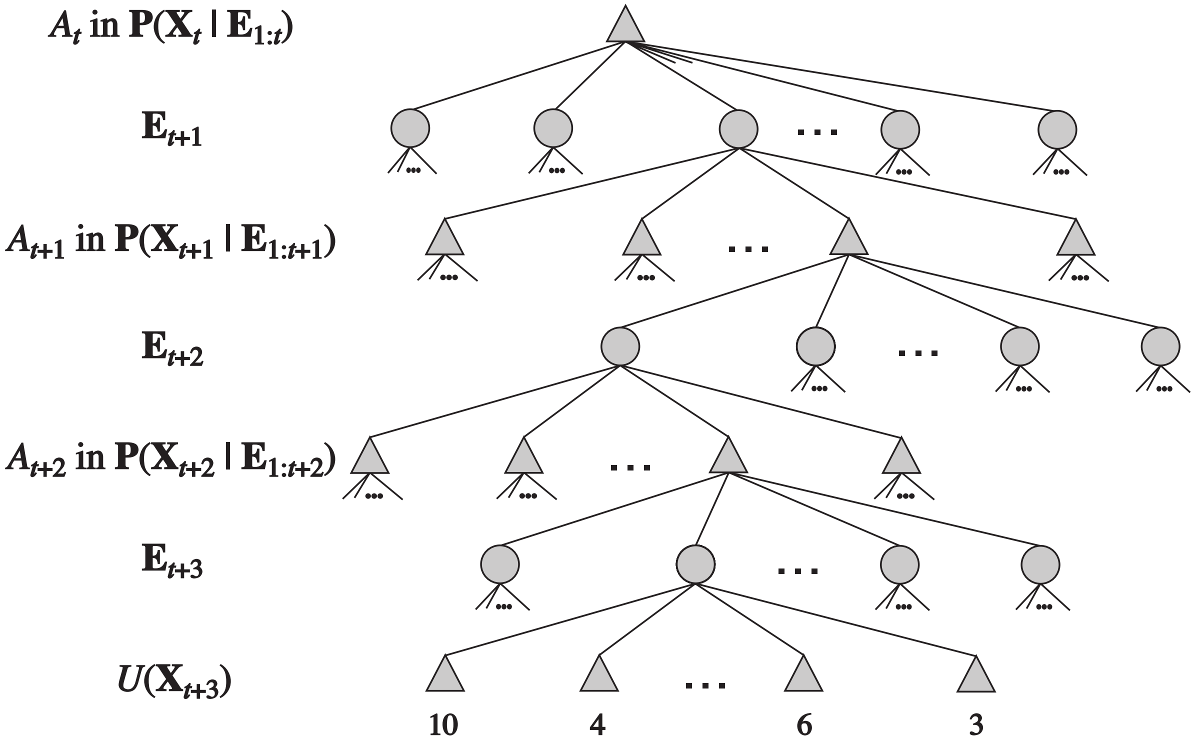
While it is difficult to directly derive π^*, a decision-theoretic agent can be constructed for POMDPs:

- The transition and sensor models are represented by a dynamic Bayesian network;
- The dynamic Bayesian network is extended with decision (A) and utility (R and U) nodes to form a dynamic decision network;
- A filtering algorithm is used to incorporate each new percept and action and to update the belief state representation;
- Decisions are made by projecting forward possible action sequences and choosing (approximately) the best one, in a manner similar to a truncated Expectiminimax.



At time t, the agent must decide what to do.

- Shaded nodes represent variables with known values.
- The network is unrolled for a finite horizon.
- It includes nodes for the reward of $\mathbf{X}_{t+1}$ and $\mathbf{X}_{t+2}$, but the (estimated) utility of $\mathbf{X}_{t+3}$.



Part of the look-ahead solution of the previous decision network.

A decision can be extracted from the search tree by backing up the (estimated) utility values from the leaves, taking the average at the chance nodes and taking the maximum at the decision nodes.

Reinforcement learning

The MDP formulation assumes the knowledge of

- a transition model $P(s'|s,a)$
- a reward function $R : \mathcal{S} \to \mathbb{R}$

What if these elements are unknown?

$\Rightarrow$ We must observe or interact with the environment in order to learn the dynamics. This is reinforcement learning.

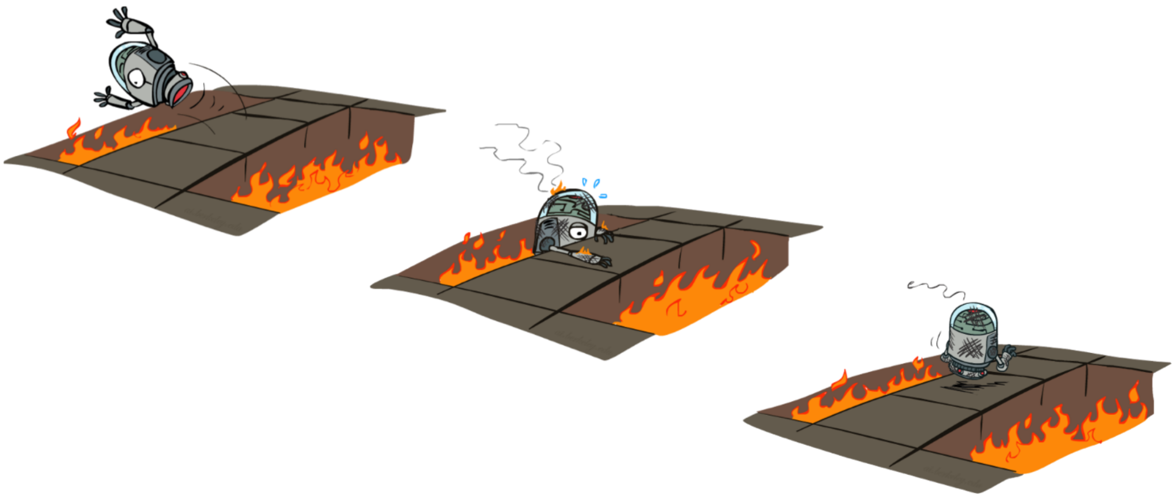


Image credits: CS188, UC Berkeley.

Summary

- Sequential decision problems in uncertain environments, called MDPs, are defined by transition model and a reward function.
- The utility of a state sequence is the sum of all the rewards over the sequence, possibly discounted over time.
    - The solution of an MDP is a policy that associates a decision with every state that the agent might reach.
    - An optimal policy maximizes the utility of the state sequence encountered when it is executed.
- Value iteration and policy can both be used for solving MDPs.
- POMDPs are much more difficult than MDPs. However, a decision-theoretic agent can be constructed for those environments.

The end.

References

xxx