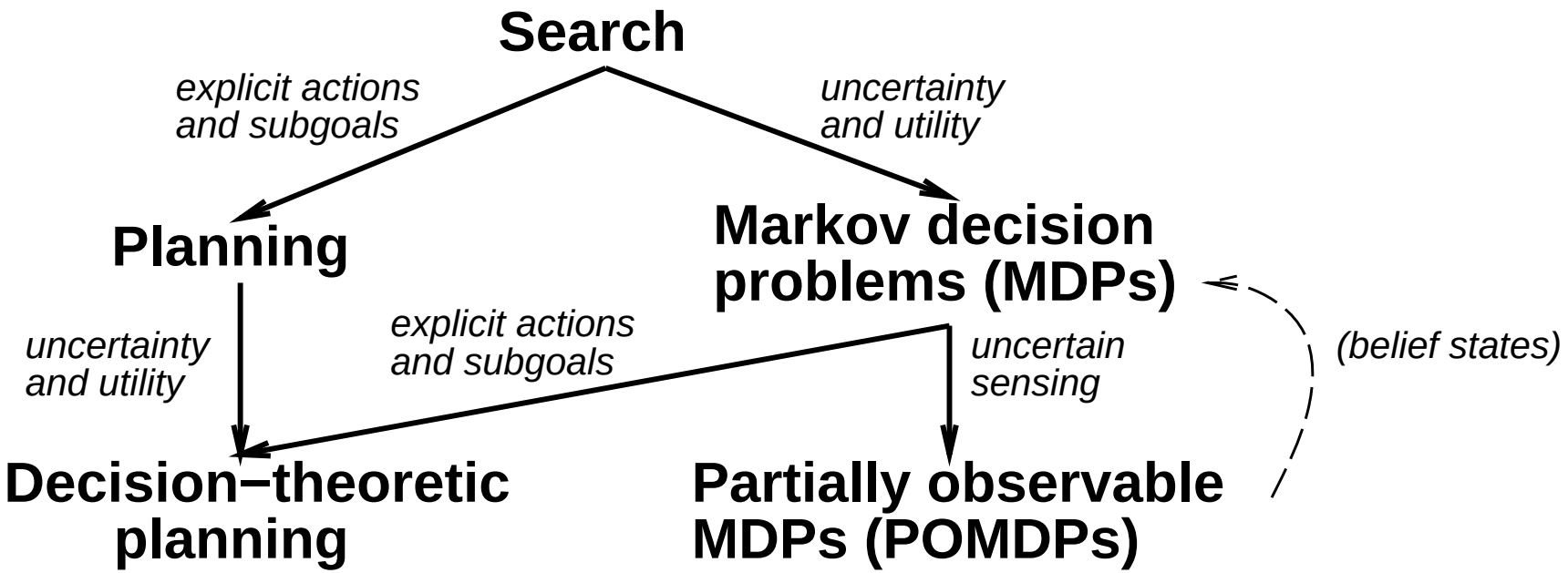
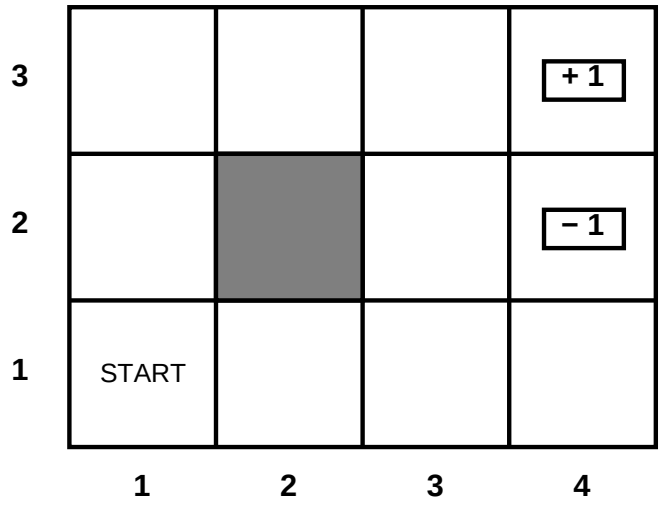
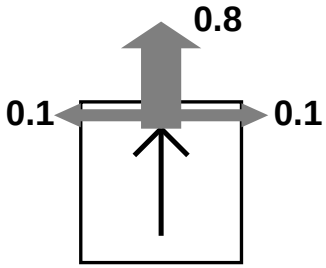
Sequential decision problems

Sequential decision problems



Example MDP





States $s \in S$, actions $a \in A$

Model $T(s, a, s') \equiv P(s'|s, a) =$ probability that a in s leads to s'

Reward function $R(s)$ (or $R(s, a)$, $R(s, a, s')$)

$$= \begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

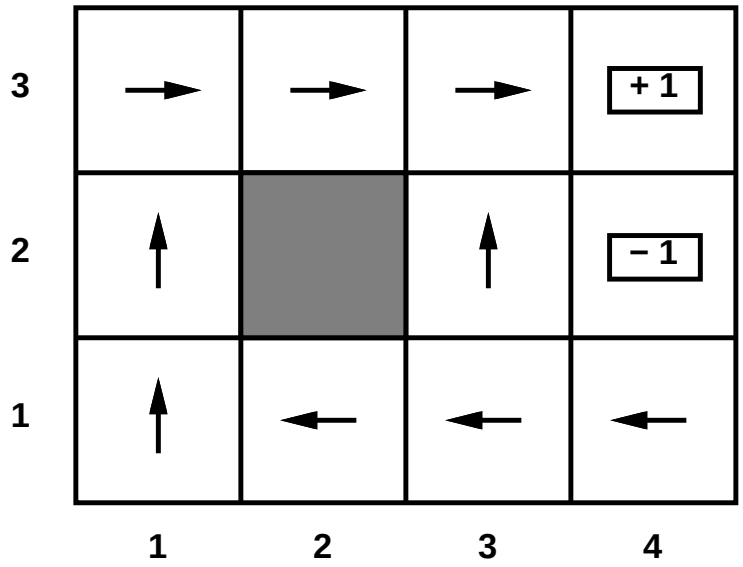
Solving MDPs

Policy $\pi(s)$: mapping from states to actions

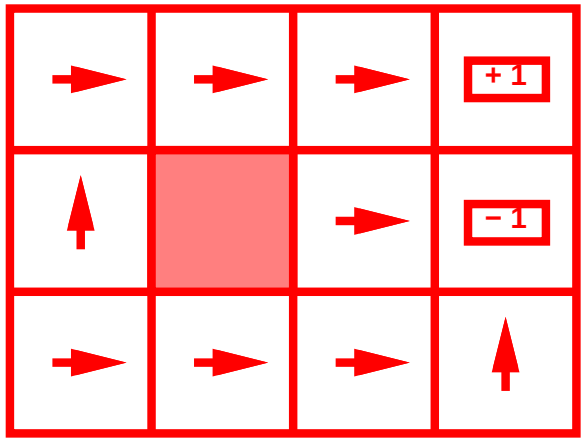
Aim is to find an optimal *policy* $\pi^*(s)$, i.e., best action for every possible state (because can't predict where one will end up)

Optimality defined quantiatively, e.g., expected sum of future rewards

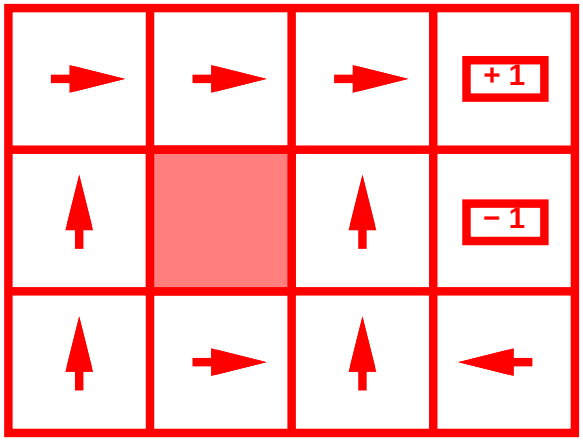
When penalty for nonterminal actions (r) is –0.04, optimal policy is:



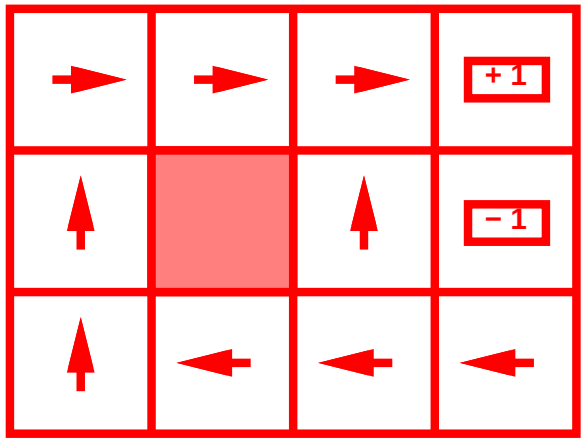
Risk and reward



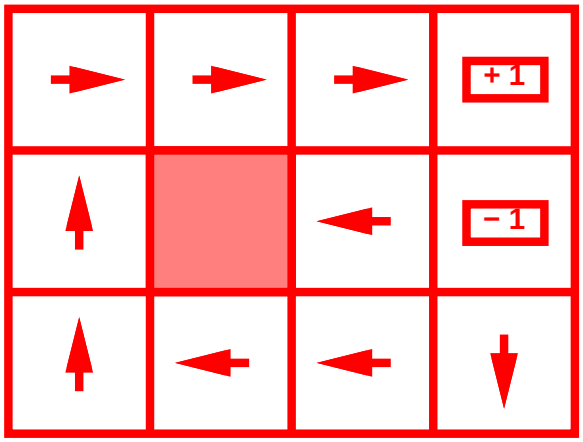
r = [− ∞ : −1.6284]



r = [−0.4278 : −0.0850]



r = [−0.0480 : −0.0274]



r = [−0.0218 : 0.0000]

Utility of state sequences

Need to understand preferences between *sequences* of states

Typically consider stationary preferences on state sequences:

$$[s_0, s_1, s_2, \ldots] \succ [s_0, s'_1, s'_2, \ldots] \Leftrightarrow [s_1, s_2, \ldots] \succ [s'_1, s'_2, \ldots]$$

I.e., if you prefer one future to another, then you'd still prefer that future if it started today.

Only two utility functions achieve this constraint:

1) *Additive* utility function:
   $U([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots$
2) *Discounted* utility function:
   $U([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$
   where γ is the discount factor

Additive utilities and infinite lifetimes

Problem: infinite lifetimes $\Rightarrow$ additive utilities are infinite ($+\infty$ or $-\infty$)

1) Absorbing state(s): If agent is guaranteed to eventually reach a terminal state for any π, expected utility of every state is finite.

2) Finite horizon: termination at a *fixed time* T
   $\Rightarrow$ nonstationary policy: $\pi(s)$ depends on time left

3) Discounting with finite reward: assuming $\gamma < 1$, $R(s) \leq R_{\max}$,

$$U([s_0, \ldots s_\infty]) = \Sigma_{t=0}^{\infty} \gamma^t R(s_t) \leq R_{\max}/(1-\gamma)$$

4) Maximize average reward per time step

$$U([s_0, \ldots s_\infty]) = \lim_{T \to \infty} \frac{1}{T} \Sigma_{t=0}^{T} R(s_t)$$

Theorem: optimal policy has constant avg. reward after initial transient

Utility of states

Roughly, utility of a *state* (a.k.a. its *value*) is the expected utility of state sequences that might follow it.

E.g., having $1M in your bank account

E.g., driving down the highway with bald tires

E.g., being at Flatiron Crossing on Saturday afternoon (good or bad?)

Expected discounted sum of rewards under policy π:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s\right]$$

True utility of a state, $U(s) \equiv U^{\pi^*}(s)$,
assumes agent executes optimal policy.

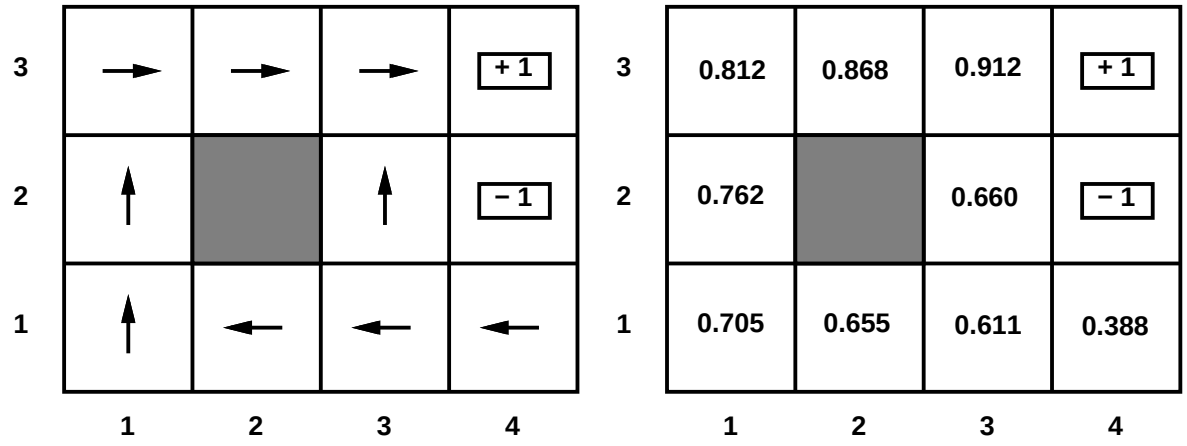
$R(s)$ is immediate reward, given by environment (or internal systems);
$U(s)$ is long-term reward, estimated by agent

Utility of states (contd.)

Note that $U(s)$ can be computed from π^*; and π^* can be computed from $U(s)$, using Maximum Expected Utility (MEU) principle:

$$\pi^*(s) = \text{argmax}_a \sum_{s'} T(s, a, s')U(s')$$

Example:



left at (3,1): $.655 * .8 + .611 * .1 + .660 * .1 = .651$
up at (3,1): $.660 * .8 + .388 * .1 + .655 * .1 = .632$

Dynamic programming: the Bellman equation

Definition of utility of states leads to a simple relationship among utilities of neighboring states:

expected sum of rewards = current reward
$+ \gamma \times$ expected sum of rewards after taking best action

Bellman (1957) equation:

$$U(s) = R(s) + \gamma \max_a \Sigma_{s'} U(s') T(s, a, s')$$

$$\begin{aligned} U(1,1) = & -0.04 \\ & + \gamma \max\{0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), && \textit{up} \\ & \qquad 0.9U(1,1) + 0.1U(1,2) && \textit{left} \\ & \qquad 0.9U(1,1) + 0.1U(2,1) && \textit{down} \\ & \qquad 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\} && \textit{right} \end{aligned}$$

One equation per state $= n$ nonlinear equations in n unknowns

Value iteration algorithm

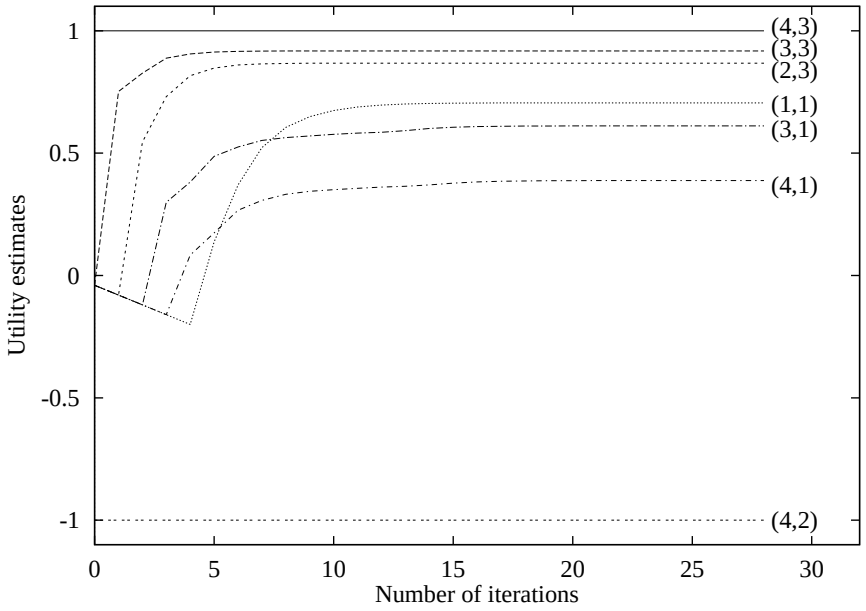
Idea: Start with arbitrary utility values

Update to make them locally consistent with Bellman eqn.

Everywhere locally consistent $\Rightarrow$ global optimality

Repeat for every s simultaneously until "no change"

$$U(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} U(s') T(s, a, s') \qquad \text{for all } s$$



Convergence

Define the max-norm: $||U|| \equiv \max_s |U(s)|$,
so $||U - V||$ = maximum difference between U and V

Let U^t and U^{t+1} be successive approximations to the true utility U

Theorem: For any two approximations U^t and V^t

$$||U^{t+1} - V^{t+1}|| \leq \gamma \, ||U^t - V^t||$$

I.e., any distinct approximations must get closer to each other.
In particular, any approximation must get closer to the true U
and value iteration converges to a unique, stable, optimal solution

Theorem: if $||U^{t+1} - U^t|| < \epsilon$, then $||U^{t+1} - U|| < 2\epsilon\gamma/(1-\gamma)$
I.e., once the change in U^t becomes small, we are almost done.

MEU policy using U^t may be optimal long before convergence of values.

Policy iteration

Howard (1960): search for optimal policy and utility values simultaneously

Algorithm:

$\pi \leftarrow$ an arbitrary initial policy
repeat until no change in π
    compute utilities given π
    update π as if utilities were correct (i.e., local MEU)

To compute utilities given a fixed π (value determination):

$$U(s) = R(s) + \gamma \sum_{s'} U(s') T(s, \pi(s), s') \qquad \text{for all } s$$

i.e., n simultaneous linear equations in n unknowns, solve in $O(n^3)$

Modified policy iteration

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

Idea: Perform approximate value-determination step

Use a few steps of value iteration (but with action suggested by π, not MEU action), and starting from the value function produced the last time.

Often converges much faster than pure value or policy iteration.

Leads to much more general algorithms where Bellman value updates and Howard policy updates can be performed locally in any order

Reinforcement learning algorithms operate by performing such updates based on the observed transitions made in an initially unknown environment.

Partial observability

Until now, have assumed environment is *fully observable*.

Real world is partially observable (POMDP).

POMDP has an observation model $O(s, e)$ defining the probability that the agent obtains evidence e when in state s

Agent does not know which state it is in

$\Rightarrow$ makes no sense to talk about policy $\pi(s)$.

Rather, consider belief state, $b(s)$, a probability distribution over states

Update rule:

$$b'(s') = \alpha O(s', o) \Sigma_s T(s, a, s') b(s)$$
$$P(s'|e, a) = \alpha P(e|s') \Sigma_s P(s'|s, a) P(s)$$

Theorem (Astrom, 1965): the optimal policy in a POMDP can be expressed as a function of belief state, $\pi(b)$

Partial observability (contd.)

Can convert a POMDP into an MDP in belief-state space, where $\tau(b, a, b')$ is the probability that the new belief state is b' given that the current belief state is b and the agent does a.

If there are n states, b is an n-dimensional real-valued vector

$\Rightarrow$ solving POMDPs is very hard!

Not only is real world POMDP, but O and T are intiially unknown functions.