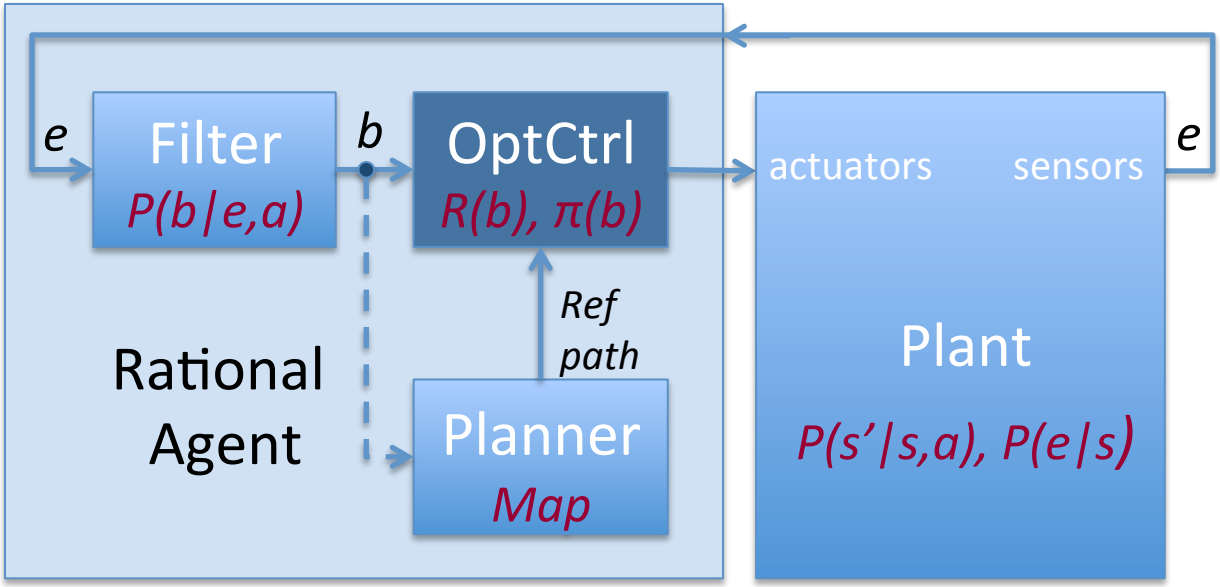
Making Complex Decisions

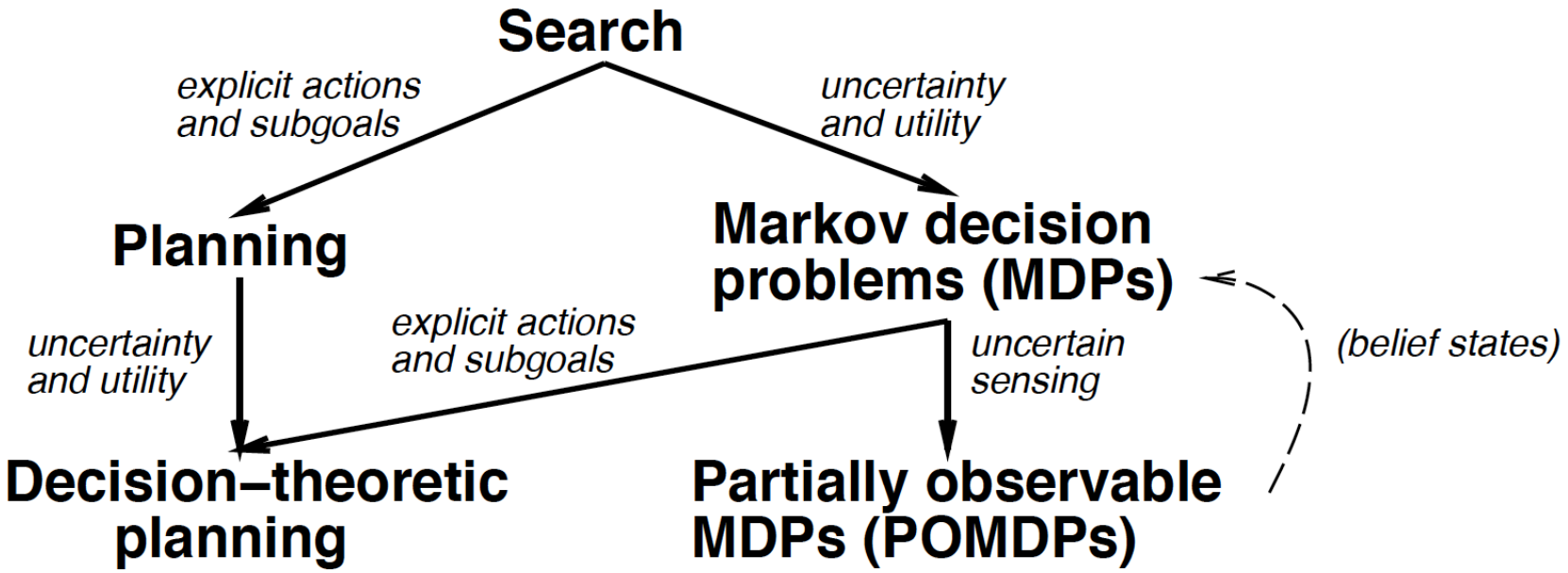
Chapter 17



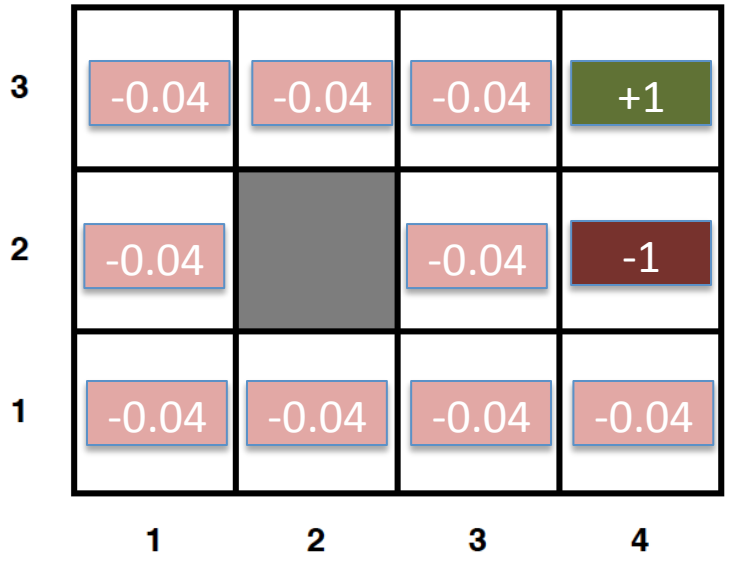
Outline

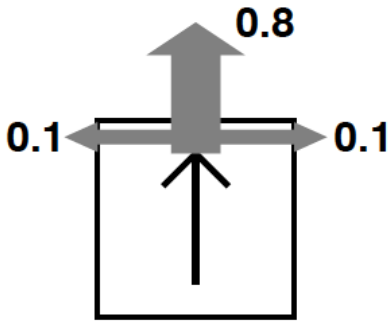
- Sequential decision problems
- Value iteration
- Policy iteration
- Partially Observable MDPs

Sequential Decision Problems



Example Markov Decision Process





$P((i,j+1) \mid (i,j), \text{Up}) = 0.8$

$T_0 = P(S = (1,1)) = 1$

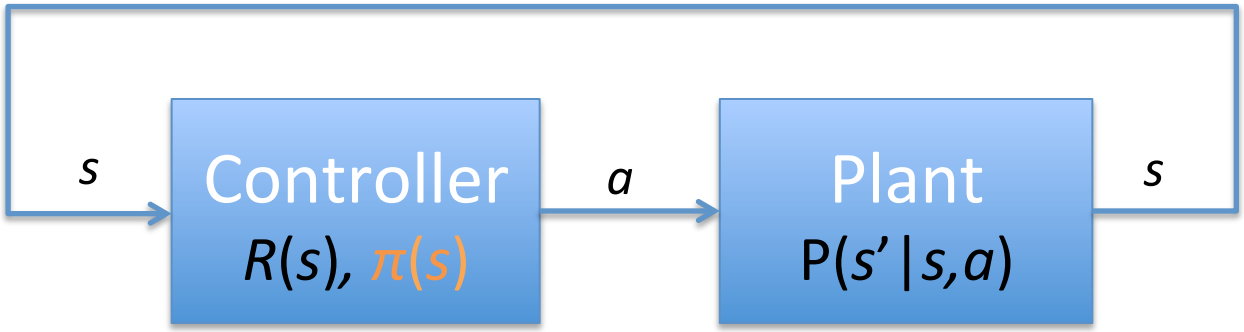
States $(i,j) \in \{1..4\} \times \{1..4\} \setminus (2,2)$, Actions $a \in \{\text{Up},\text{Down},\text{Left},\text{Right}\}$

Model $T(s,a,s') = P(s' \mid s,a) =$ Probability that a in s leads to s'

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

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

Markov Decision Process (Regulator)



States s, Actions a

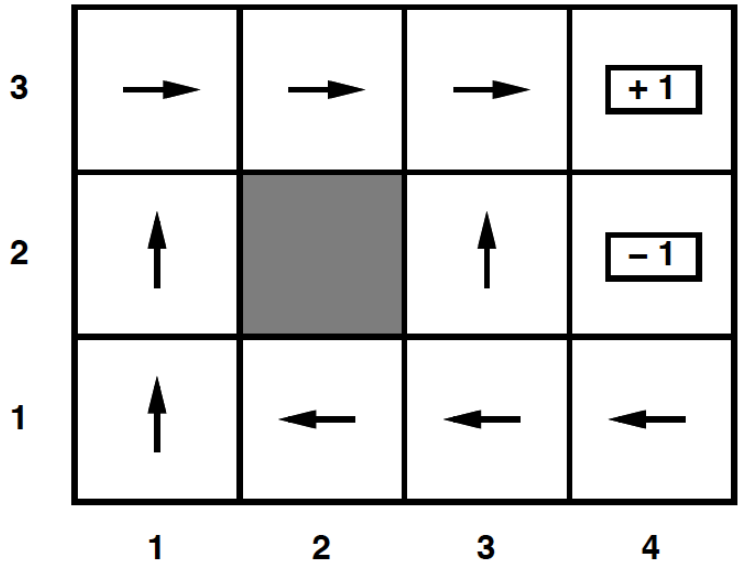
Model $T(s,a,s') = \mathrm{P}(s' \mid s,a)$, $T_0 = \mathrm{P}(s)$

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

Compute optimal policy (controller) $\pi(s)$

Solving MDPs

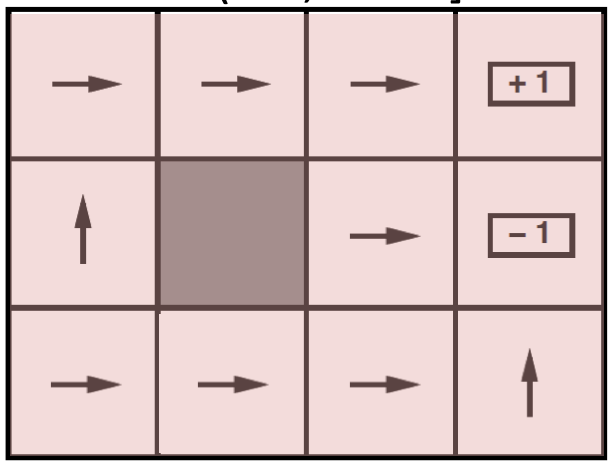
- In search problems, aim is to find an *optimal sequence*
- In MDPs, aim is to find an *optimal policy (controller) π(s)*
  - Best action for every possible state *s*
    (because can't predict where one will end up)
- Optimal policy maximizes the *expected sum of rewards*
- Optimal policy when state penalty *R*(*s*) is –0.04:



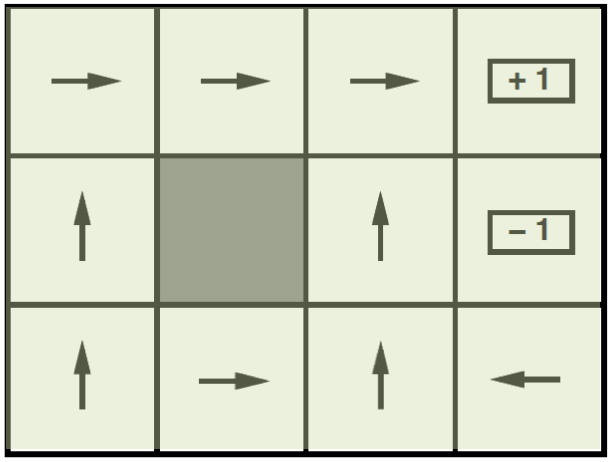
What can you observe about this policy ?

Risk and Reward

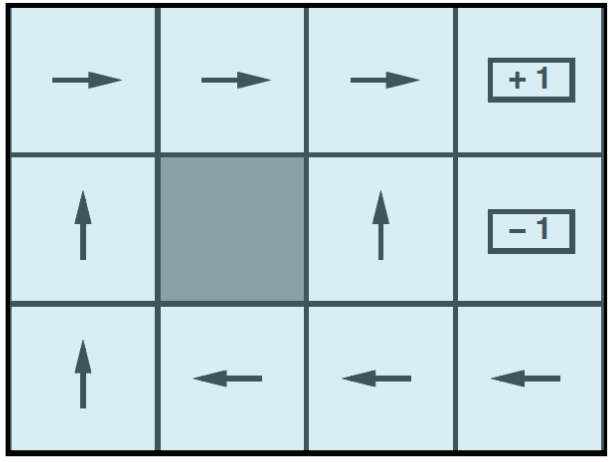
$R \in (-\infty, -1.62]$



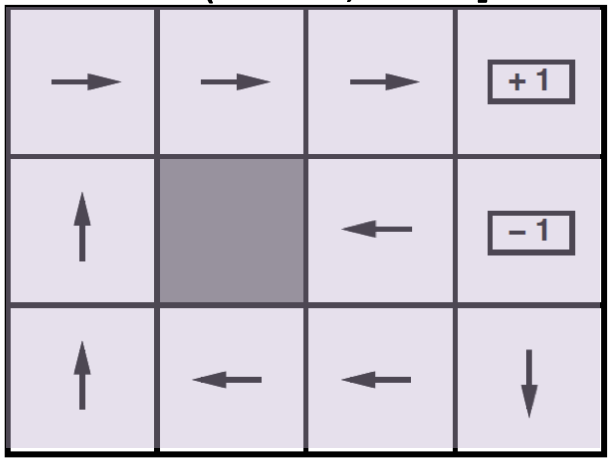
$R \in [-0.42, -0.08]$



$R \in [-0.04, -0.02]$



$R \in (-0.02,\ 0.00]$



Utility of State Sequences

Need to understand *preferences* between *sequences* of states

Typically consider *stationary* preferences on reward sequences:

$$[r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots] \Leftrightarrow [r_0, r_1, r_2, \ldots] \succ [r'_0, r'_1, r'_2, \ldots]$$

Theorem: there are *only two ways* to combine rewards over time.

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 $\gamma \in (0,1)$ is the discount factor

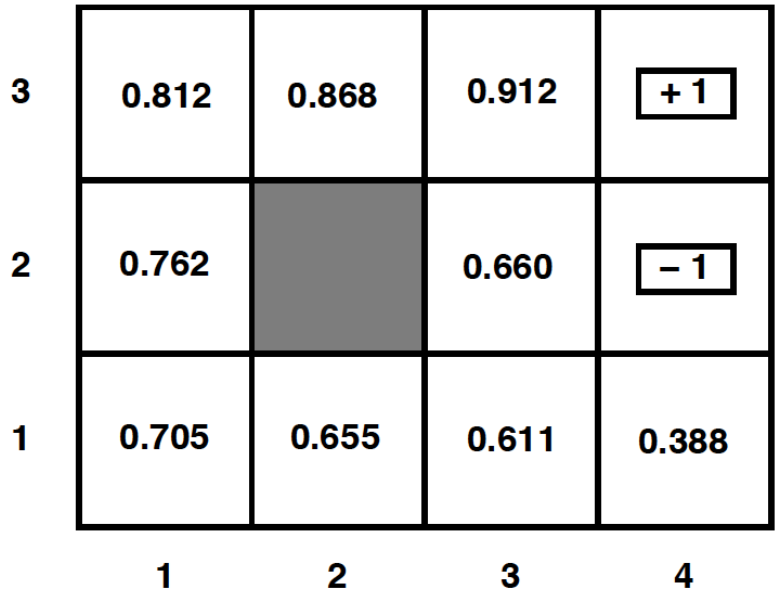
Utility of States

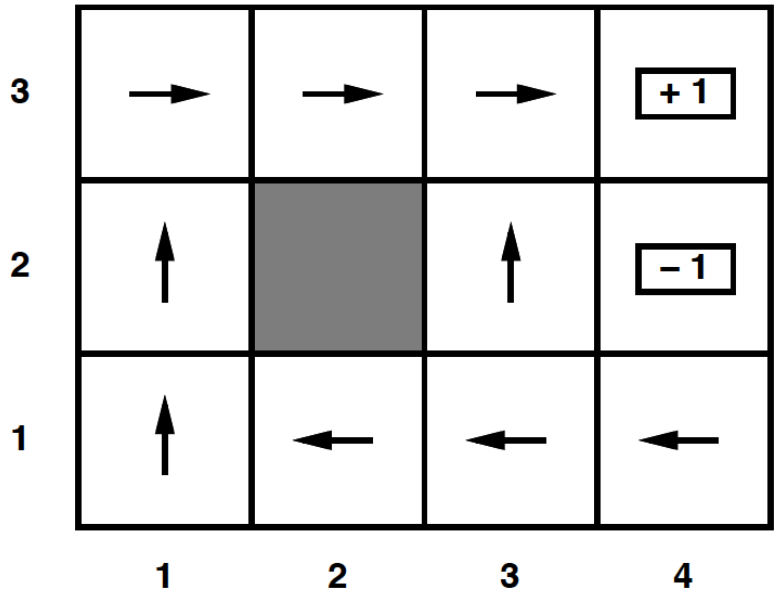
Utility of a state (a.k.a. its *value*) is defined to be

$U(s)$ = expected (discounted) sum of rewards (until termination) assuming optimal actions

Given the utilities of the states, choosing the best action is just MEU:

Maximize the expected utility of the immediate successors





Utility of State Sequences

Problem: infinite lifetimes $\Rightarrow$ additive utilities are infinite

1) Finite horizon: termination at a *fixed time T*

$\Rightarrow$ nonstationary policy: $\pi(s)$ depends on time left

2) Absorbing state(s): w/ prob. 1, agent eventually "dies" for any π

$\Rightarrow$ expected utility of every state is finite

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

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

4) Maximize system gain = average reward per time step

Theorem: optimal policy has constant gain after initial transient

Example: taxi driver's daily scheme cruising for passengers

Dynamic programming: 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 equation (1957):

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

Example:

$$U(1,1) = -0.04 + \gamma \max_a \{$$

$0.8\times U(1,2)+0.1\times U(2,1)+0.1\times U(1,1)$,	Up
$0.9\times U(1,1)+0.1\times U(1,2)$,	Left
$0.9\times U(1,1)+0.1\times U(2,1)$,	Down
$0.8\times U(2,1)+0.1\times U(1,2)+0.1\times U(1,1)$ }	Right

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

Value Iteration Algorithm (VI)

Main idea:

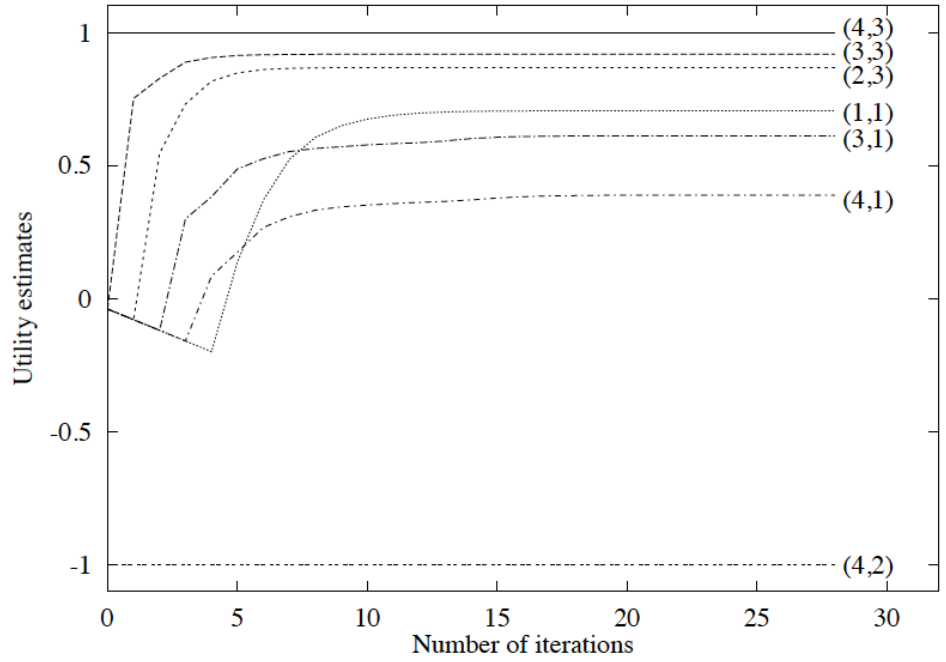
Start with arbitrary utility values

Update to make them locally consistent with Bellman eqn.

Everywhere locally consistent $\Rightarrow$ global optimality

Repeat for every *s* simultaneously until "no change"

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} U_i(s')T(s,a,s') \quad \text{for all s}$$



Convergence

Define the max-norm $||U|| = \max_s |U(s)|$, 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

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

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 $||U^{t+1} - U^t|| \leq \varepsilon$ then $||U^{t+1} - U|| \leq 2\varepsilon\gamma/(1-\gamma)$

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 (PI)

Idea: Search for optimal *policy and utility values* simultaneously

Algorithm:

$\pi \leftarrow$ an arbitrary initial policy

repeat until no change in π

compute $U(s)$ given π (here π is kept fix)

update π given U (here U is kept fix, local MEU)

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

$$U(s) \leftarrow R(s) + \gamma \sum_{s'} U(s')T(s, \pi(s), s') \quad \forall s$$

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

To update the policy π perform local MEU:

$$\pi(s) = (\max_{a \in A(s)} \sum_{s'} U(s')T(s, a, s') > \sum_{s'} U(s')T(s, \pi(s), s'))? \ \operatorname{argmax}_{a \in A(s)} \sum_{s'} U(s')T(s, a, s') : \pi(s) \quad \forall s$$

Modified Policy Iteration

PI: often converges in few iterations, but each is expensive

Idea:

- Use a few steps: of value iteration but with π fixed
- Starting from: the value function produced last time
- To produce: an approximate value determination step

Often converges much faster: than pure VI or PI

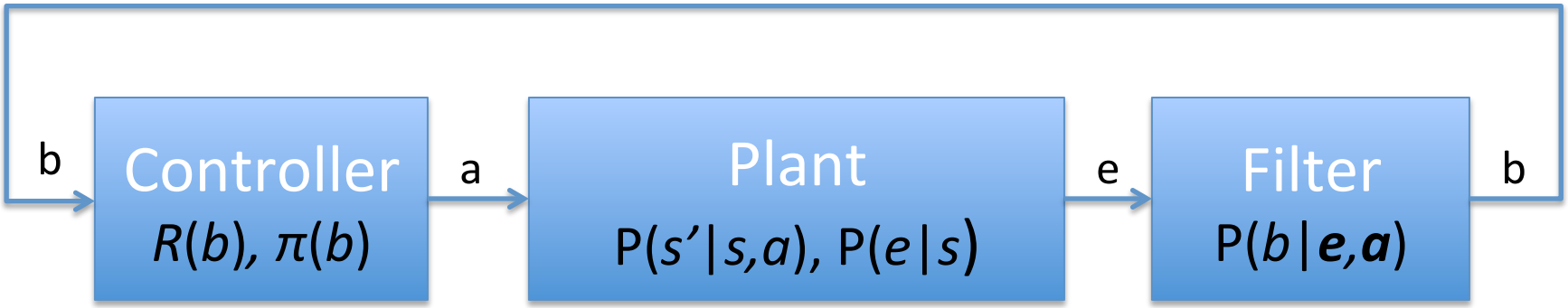
Leads to much more general algorithms where:

- 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 (POMDP)



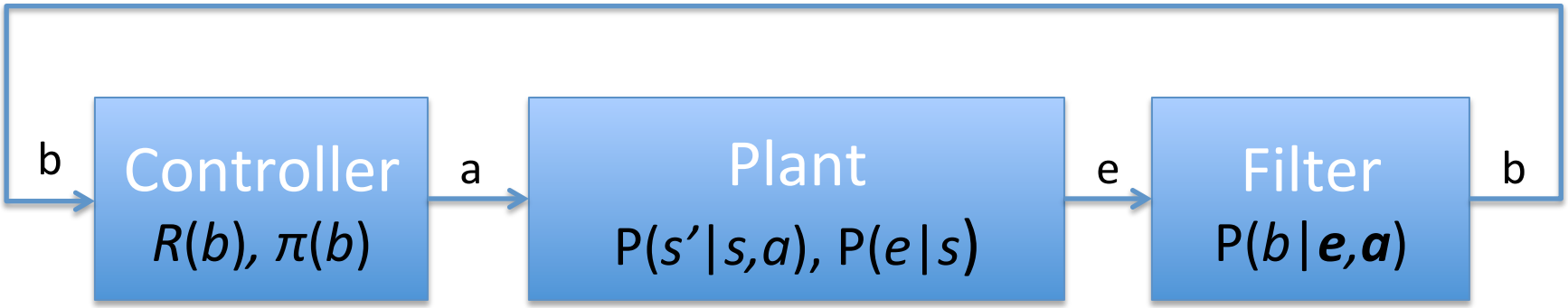
Partially observable MDPs extend MDPs by having:

An observation model: $O(s,e) = P(e|s)$, defining
The probability: that the agent obtains evidence evidence e in state s

Agent does not know which state it is in

$\Rightarrow$ Makes no sense: to talk about policy $\pi(s)$!

Partial Observability (POMDP)



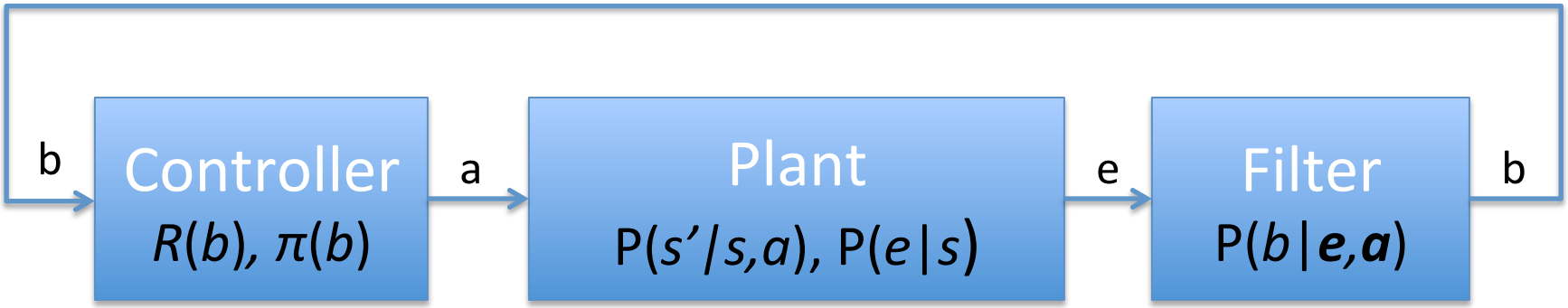
Theorem (Astrom): The optimal policy in a POMDP is a

Function $\pi(b)$ where b is a
Belief state, ie. a probability distribution over states

Can convert a POMDP into an MDP in belief-state space, where

$T(b,a,b') = P(b' \mid b,a)$ is the probability that
New BS is b' given that current BS is b and the agent does a.
Filtering: one performs filtering before applying control.

Partial Observability (Contd.)



Solutions automatically incude *information-gathering* behavior

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

$\Rightarrow$ Solving POMDP is very hard, actually PSPACE hard!

The real world is a POMDP (with initially unknown T and O)