Complex sequential decisions II

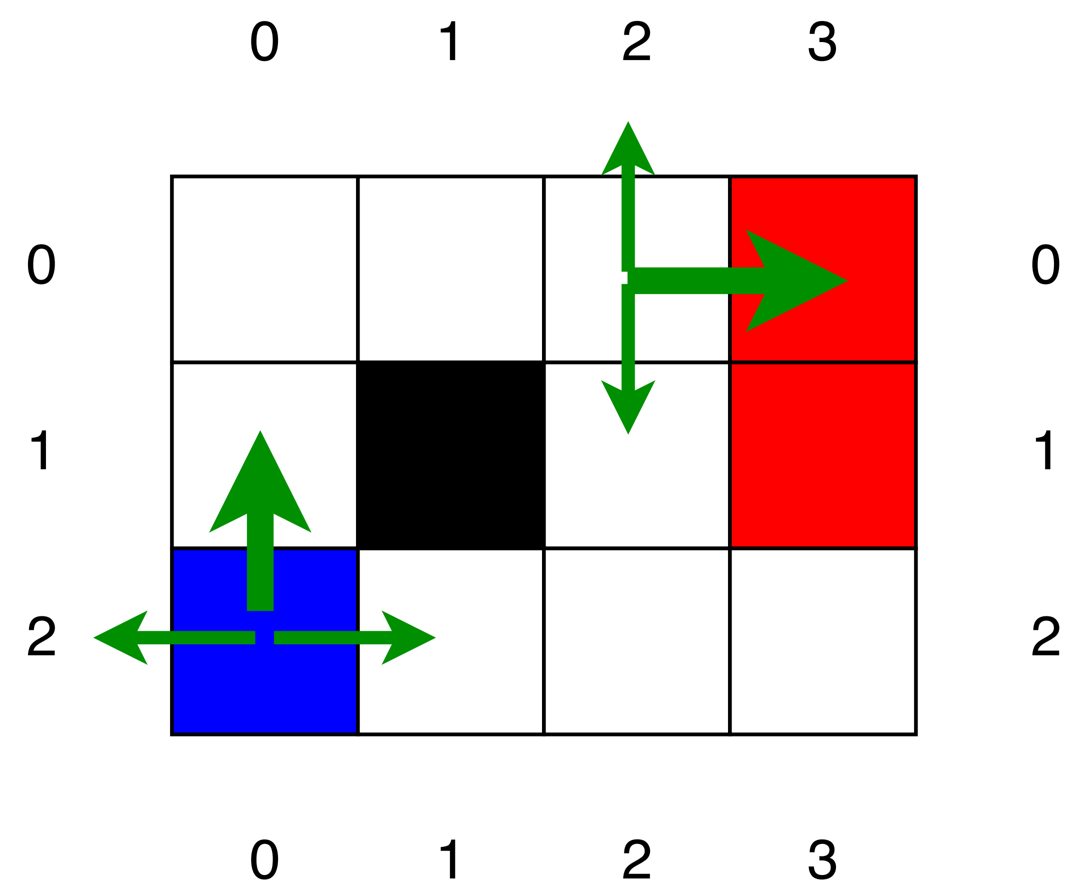
Tomas Svoboda, BE5B33KUI

2017-04-10

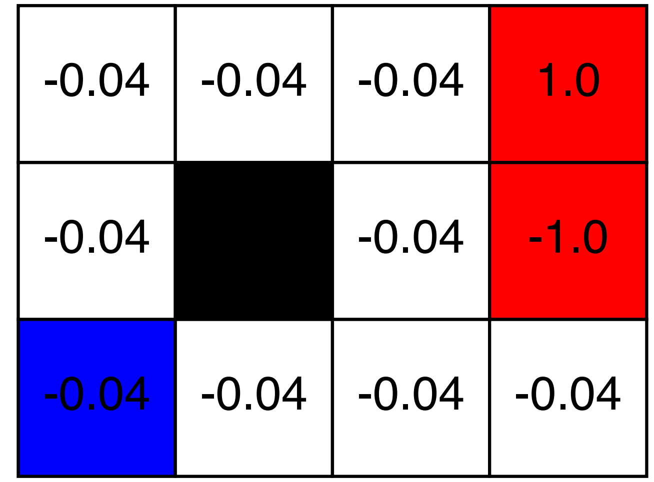
Slide material partly from CS 188: Artificial Intelligence at UCB
by Dan Klein, and Pieter Abbeel, used with permision

Uncertain movement in a grid world

- If there is a wall - agent bounces and stays in place
- **Rewards** each time step:
  - Small “living” reward each step (can be negative)
  - Big rewards at the end
- **Goal**: maximize sum of (discounted) rewards



MDP recap:



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

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

MDP quantities:

Policy: map (dictionary) of states to actions

Utility: sum of discounted rewards

Utility of a state: expected future utility from that state

State utilities, putting Rewards into the sums

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a)(R(s) + \gamma U(s'))$$

Q-state, chance state

- The value (utility) of a state s:

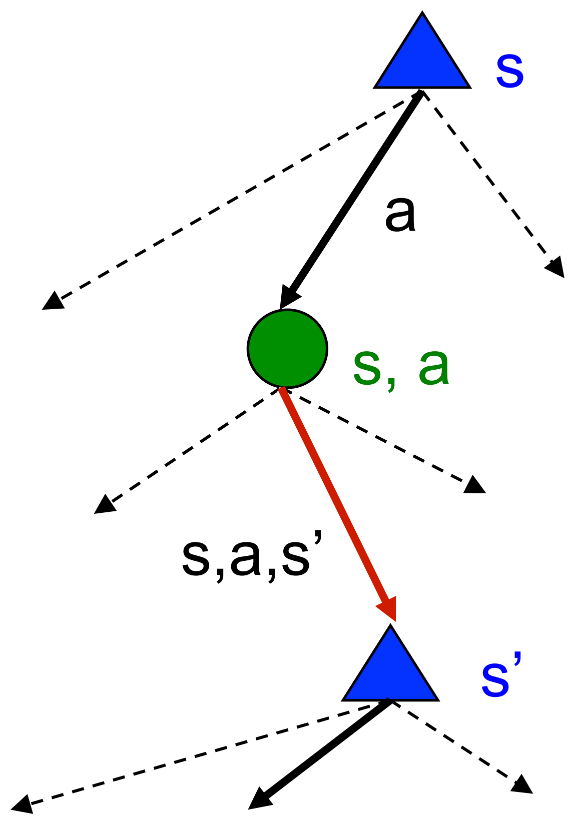
  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a):

  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

  $\pi^*(s)$ = optimal action from state s

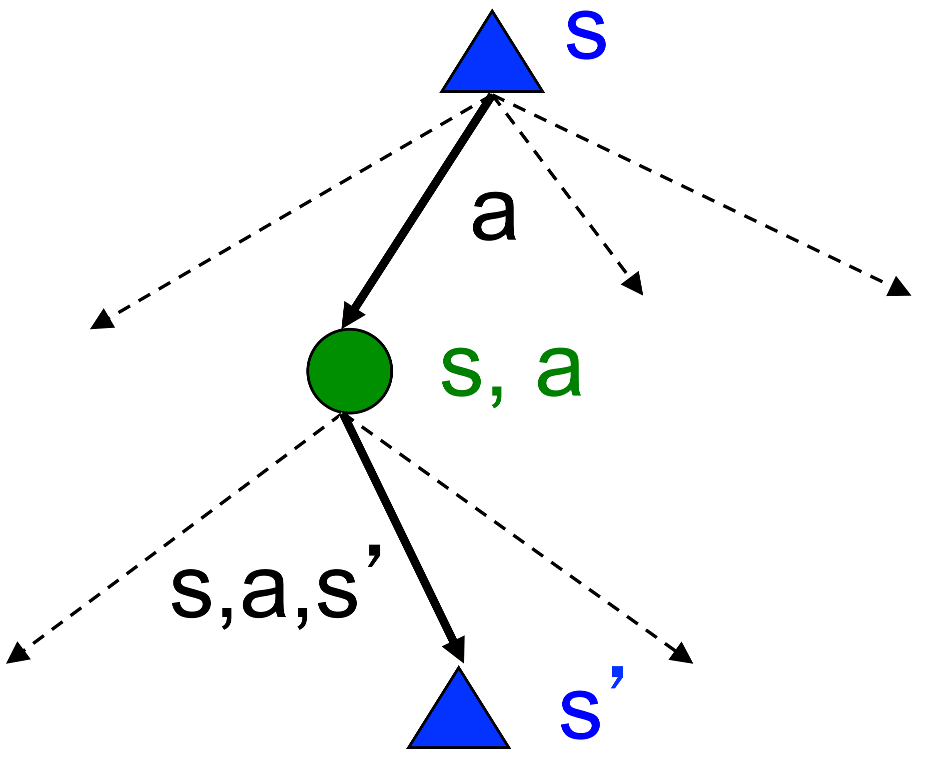


s is a *state*

(s, a) is a *q-state*

(s,a,s') is a *transition*

Value of states



$$V^*(s) = \max_{a \in A(s)} Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\big[R(s,a,s') + \gamma V^*(s')\big]$$

$$V^*(s) = \max_{a \in A(s)} \sum_{s'} T(s,a,s')\big[R(s,a,s') + \gamma V^*(s')\big]$$

0.81	0.87	0.92	1.0
0.76		0.66	-1.0
0.71	0.66	0.61	0.39

Value iteration

Bellman equations *characterize* the optimal values

$$V^*(s) = \max_{a \in A(s)} \sum_{s'} T(s,a,s') \big[R(s,a,s') + \gamma V^*(s') \big]$$

Value iteration *computes* it

$$V_{i+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} T(s,a,s') \big[R(s,a,s') + \gamma V_i(s') \big]$$

What is the complexity (for one iteration)?

Recap

	0	1	2	3	
0	-0.04	-0.04	-0.04	1.0	0
1	-0.04		-0.04	-1.0	1
2	-0.04	-0.04	-0.04	-0.04	2
	0	1	2	3	

	0	1	2	3	
0	0.81	0.87	0.92	1.0	0
1	0.76		0.66	-1.0	1
2	0.71	0.66	0.61	0.39	2
	0	1	2	3	

1. Estimate state values (utilities)

2. Extract policy

Policy extraction

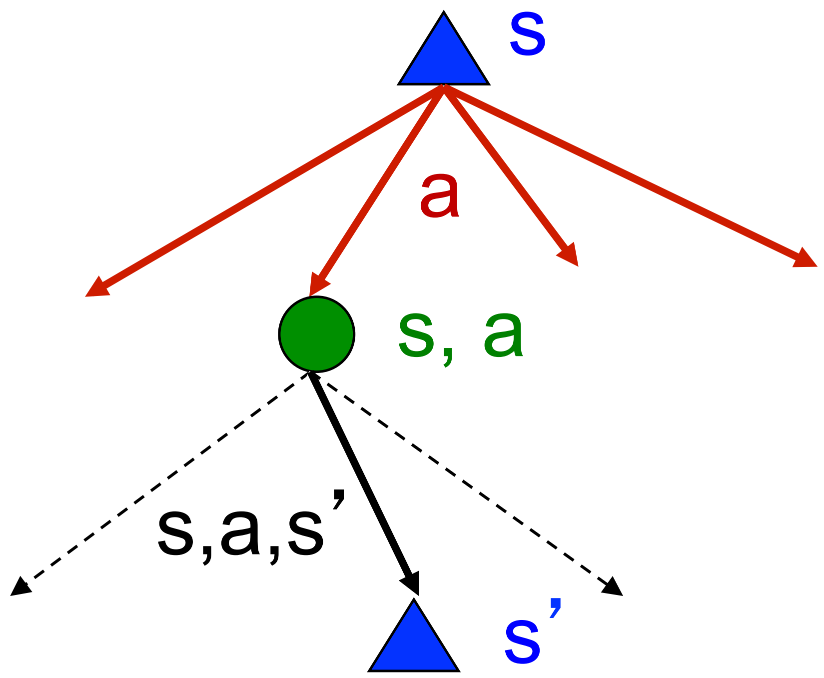
	0	1	2	3	
0	0.81	0.87	0.92	1.0	0
1	0.76		0.66	-1.0	1
2	0.71	0.66	0.61	0.39	2

$$\pi^*(s) = \underset{a \in A(s)}{\text{arg max}} \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

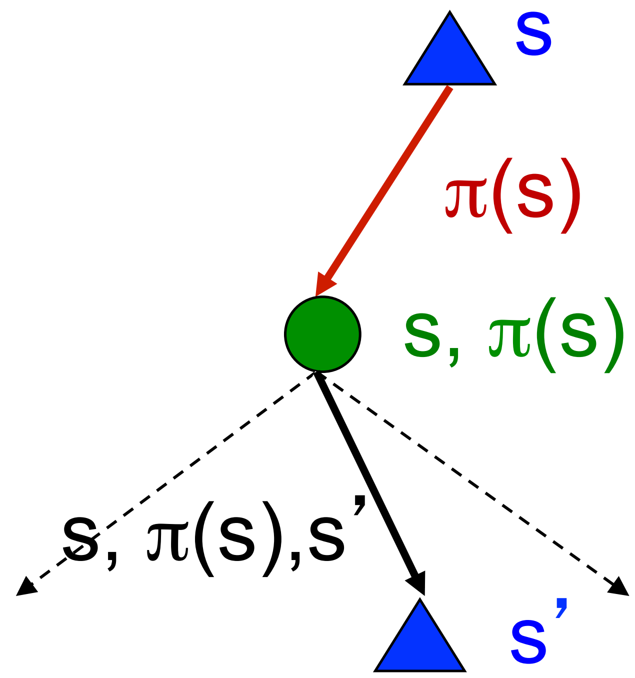
$$\pi^*(s) = \underset{a \in A(s)}{\text{arg max}}\, Q^*(s,a)$$

Fixed policies

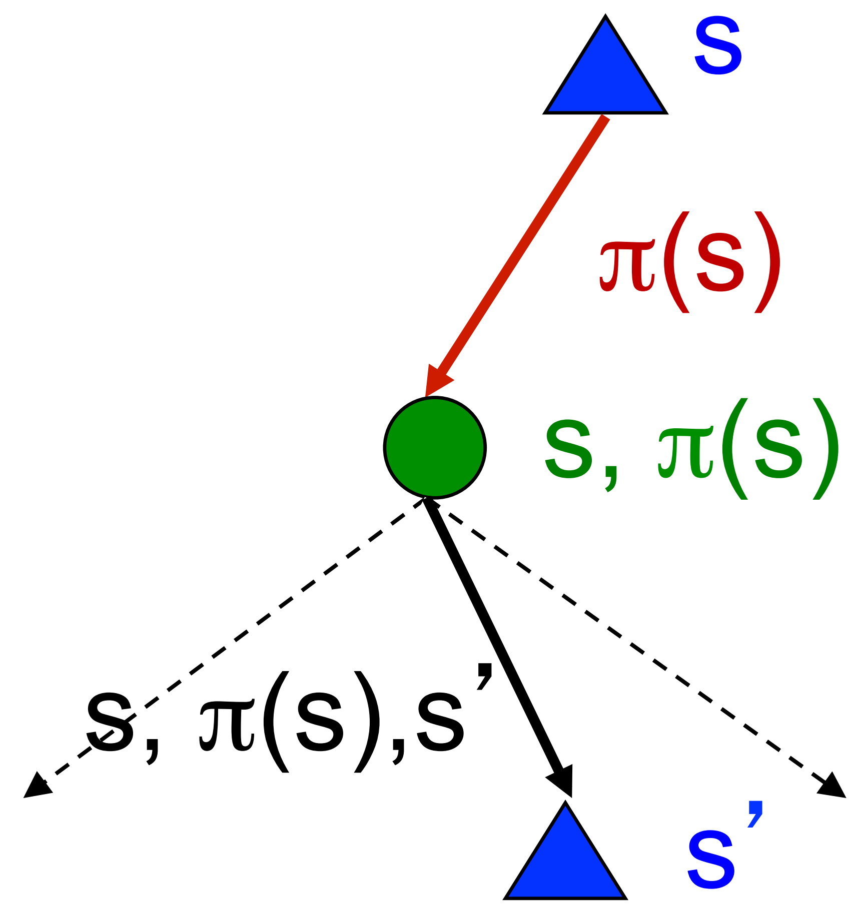
Do the optimal action



Do what π says to do



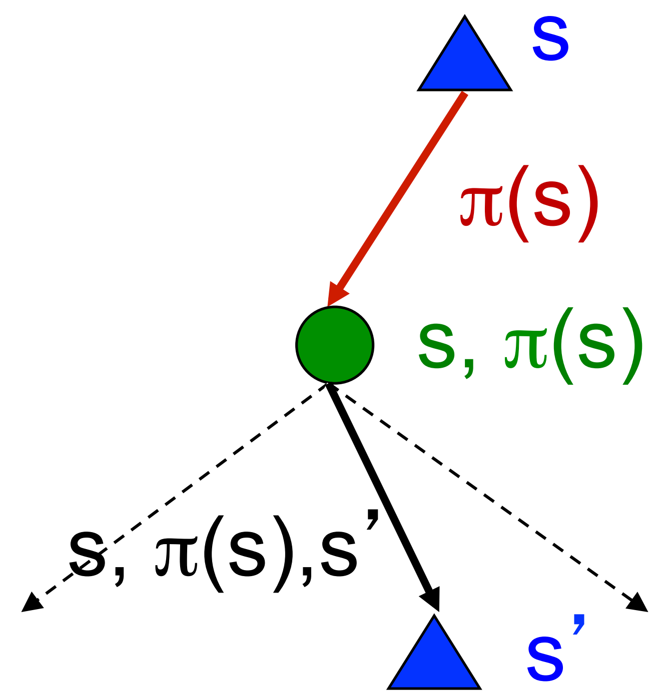
Utilities for a Fixed policy



one-step look ahead / Bellmann equation

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')\big[R(s, \pi(s), s') + \gamma V^{\pi}(s')\big]$$

Policy Evaluation



$$V_0^{\pi}(s) = 0$$

$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')\left[R(s, \pi(s), s') + \gamma V_i^{\pi}(s')\right]$$

Other options for policy evaluation?

Policy iteration

- Step 1: **Policy evaluation**: calculate utilities for some fixed policy
- Step 2: **Policy improvement**: update policy using one-step look-ahead with the utilities computer in Step1
- Repeat steps until policy converges

Policy iteration

Policy evaluation. Iterate until converge

$$V_{i+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \big[R(s, \pi_i(s), s') + \gamma V_i^{\pi_i}(s') \big]$$

Policy improvement. One-step look-ahead

$$\pi_{i+1}(s) = \operatorname*{arg\,max}_{a \in A(s)} \sum_{s'} T(s, a, s') \big[R(s, a, s') + \gamma V^{\pi_i}(s') \big]$$

Comparison

- Both Value iteration and Policy iteration compute optimal values
- Value iteration
  - every iteration update values (and thus also policy)
- in Policy iteration
  - update utilities with fixed policy (fast)
  - a new policy is chosen (like a value iteration pass)
  - new policy better
- Both are dynamic programs for solving MDPs