Markov Decision Processes

Hal Daumé III

Computer Science
University of Maryland

me@hal3.name

CS 421: Introduction to Artificial Intelligence

21 Feb 2012

Many slides courtesy of
Dan Klein, Stuart Russell,
or Andrew Moore

Announcements

- Mid-course corrections:
  - http://u.hal3.name/ic.pl?q=midcourse

Reinforcement Learning

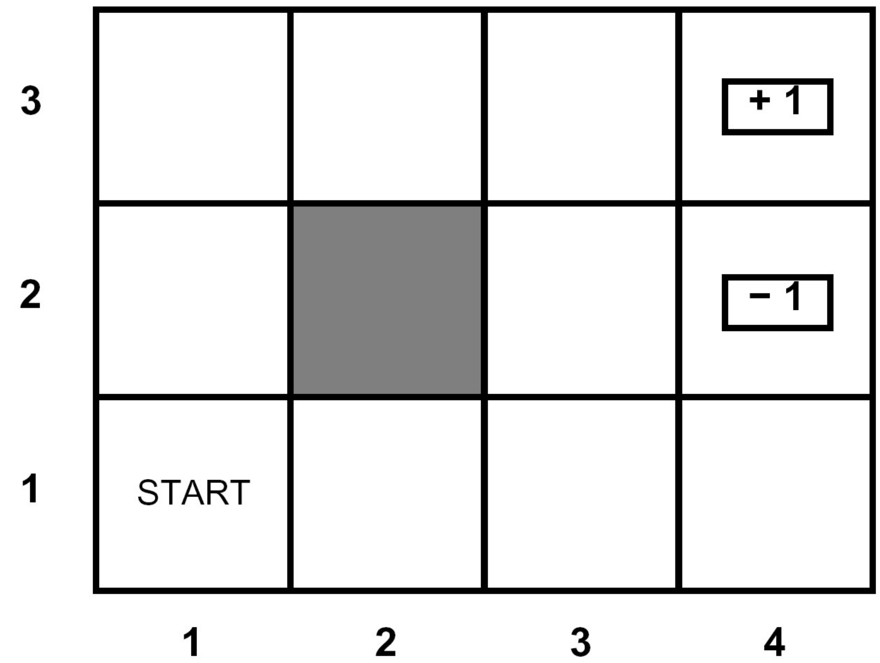
- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must learn to act so as to maximize expected rewards
  - Change the rewards, change the learned behavior
- Examples:
  - Playing a game, reward at the end for winning / losing
  - Vacuuming a house, reward for each piece of dirt picked up
  - Automated taxi, reward for each passenger delivered

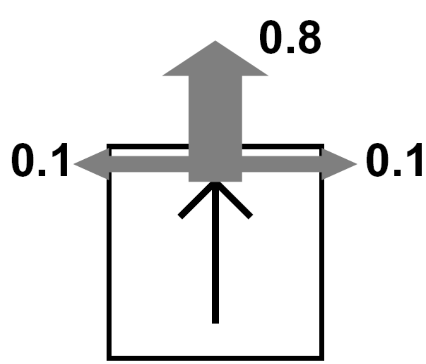
Human Reinforcement Learning

Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function $T(s,a,s')$
    - Prob that a from s leads to s'
    - i.e., $P(s' \mid s,a)$
    - Also called the model
  - A reward function $R(s, a, s')$
    - Sometimes just $R(s)$ or $R(s')$
  - A start state (or distribution)
  - Maybe a terminal state

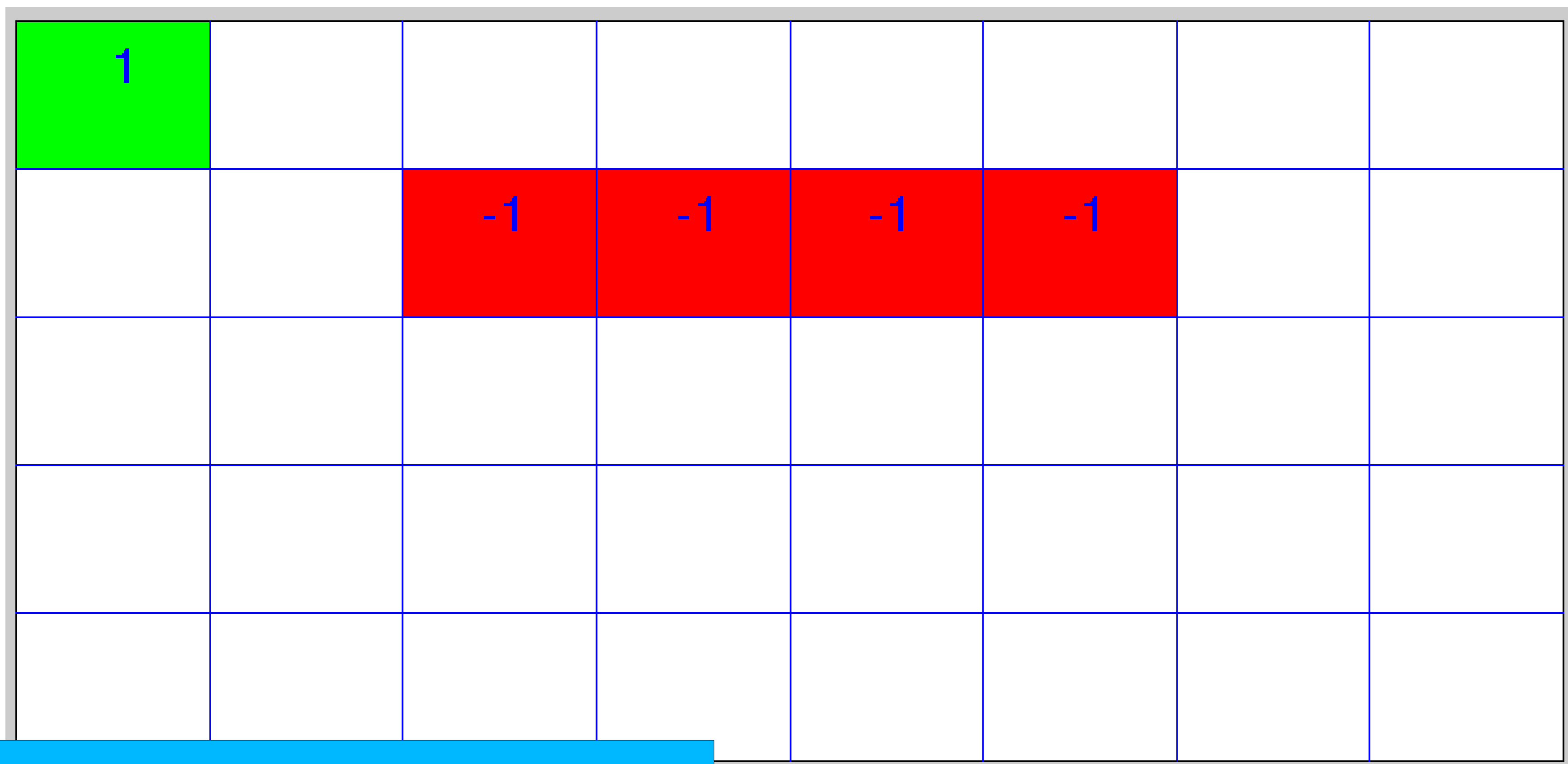
- MDPs are a family of non-deterministic search problems
  - Reinforcement learning: MDPs where we don't know the transition or reward functions





Map 0: Would you go across the top?

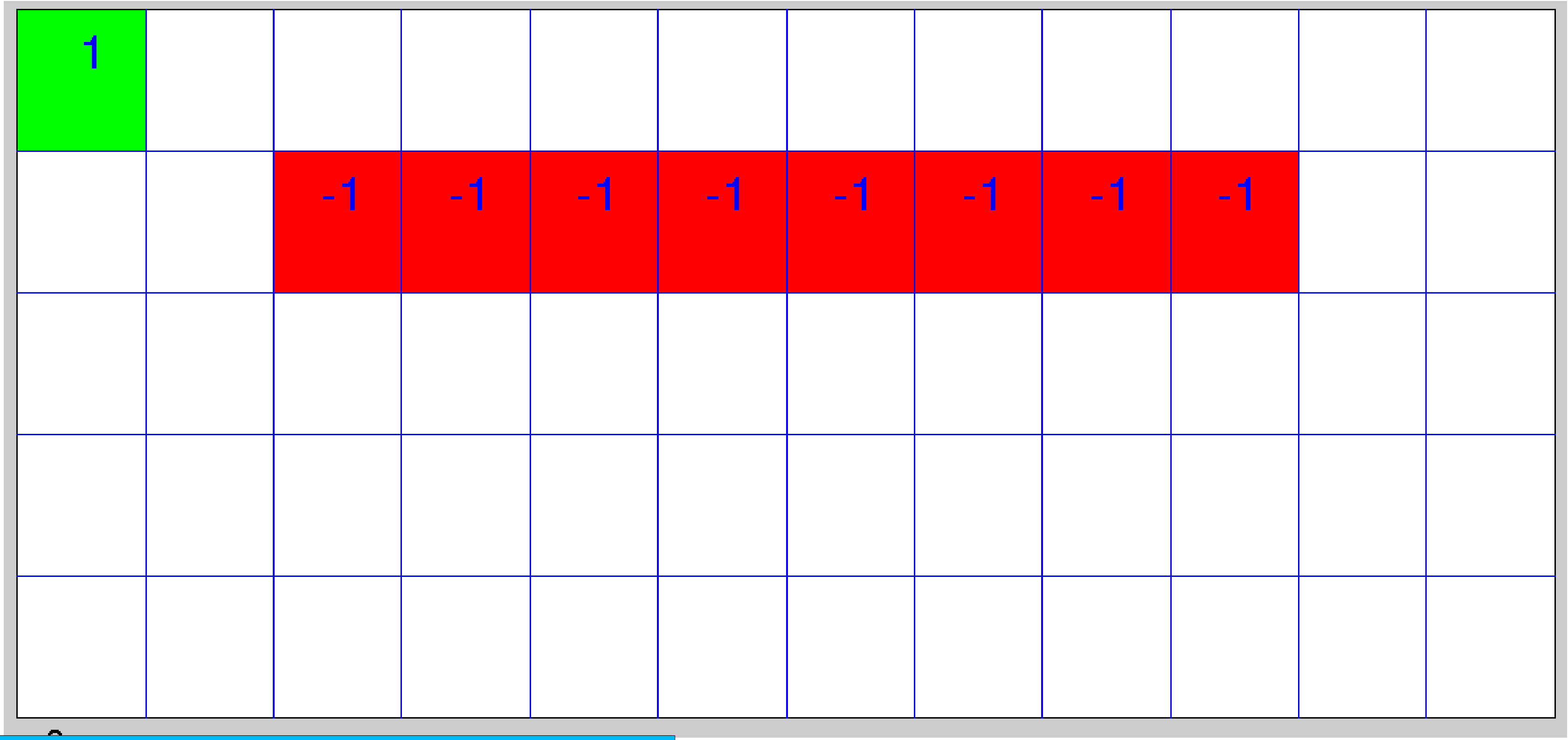
- Start in top-right, +$1 for top left, -$1 for red squares
- Costs *N* cents per step
- For what value *N* would you risk the “high road”?
  - Write something between 1 cent and 12 cents



u.hal3.name/ic.pl?q=map

Map 1: Would you go across the top?

- Start in top-right, +$1 for top left, -$1 for red squares
- Costs *N* cents per step
- For what value *N* would you risk the “high road”?
  - Write something between 1 cent and 12 cents

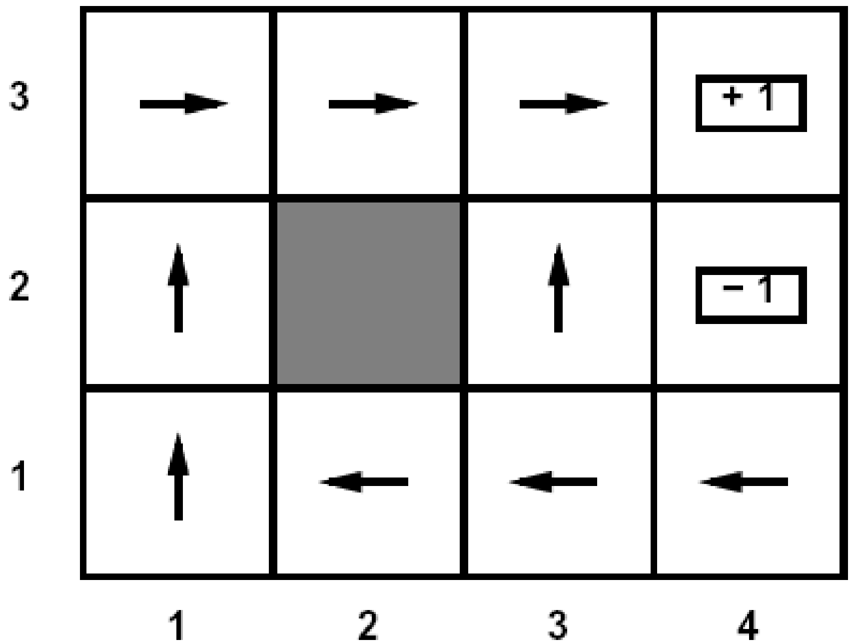


u.hal3.name/ic.pl?q=map

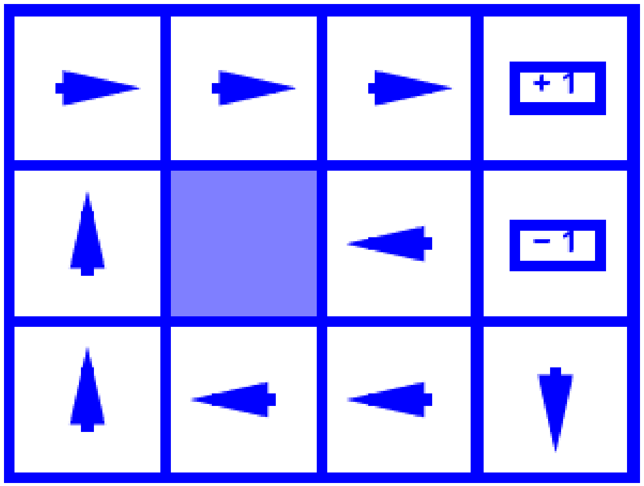
Solving MDPs

- In deterministic single-agent search problem, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy $\pi(s)$
  - A policy gives an action for each state
  - Optimal policy maximizes expected if followed
  - Defines a reflex agent

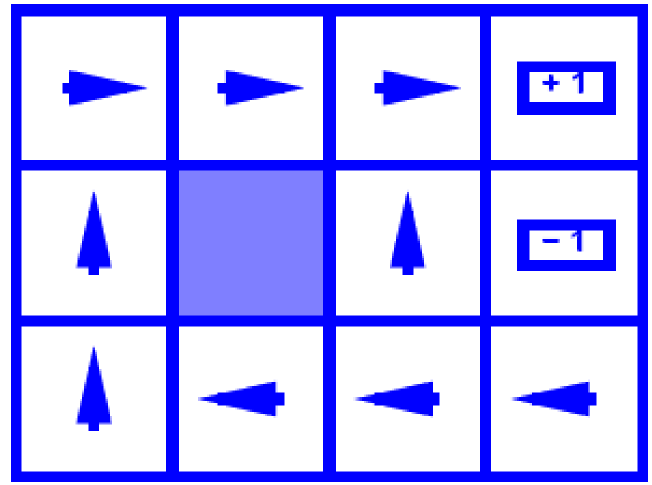
Optimal policy when R(s, a, s') = -0.04 for all non-terminals s



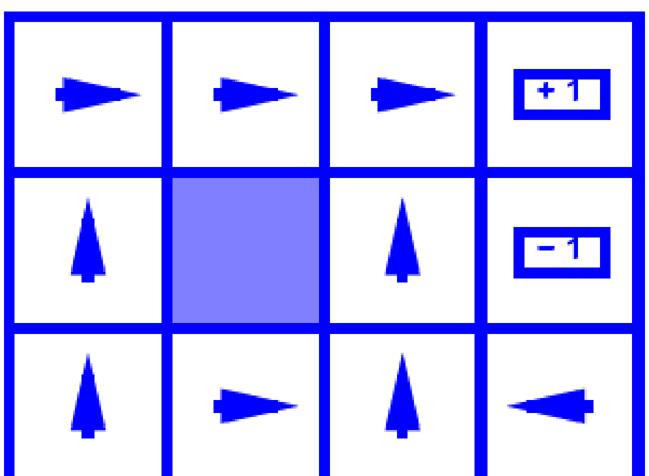
Example Optimal Policies



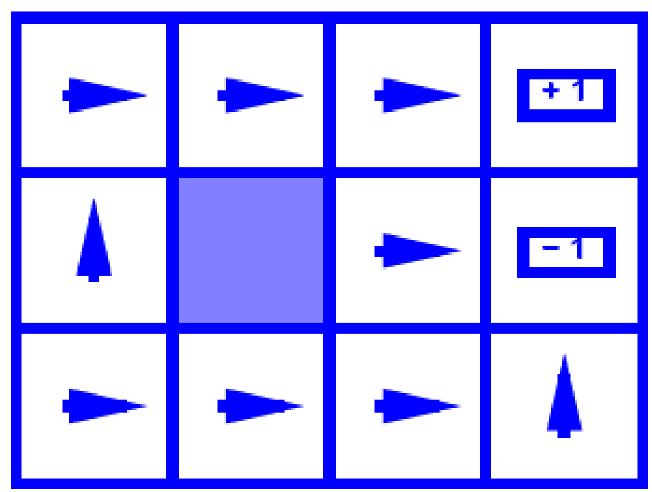
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4

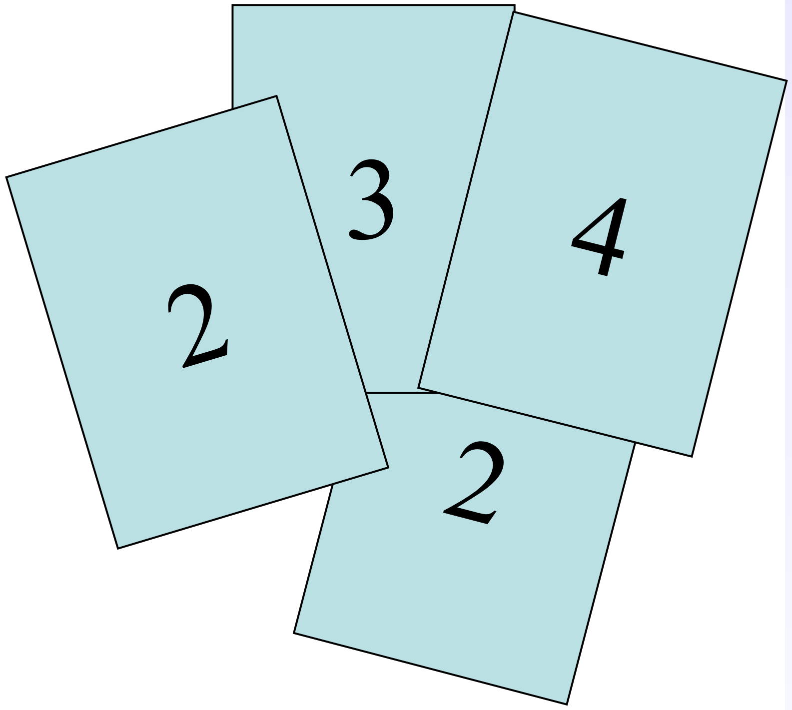


R(s) = -2.0

Example: High-Low

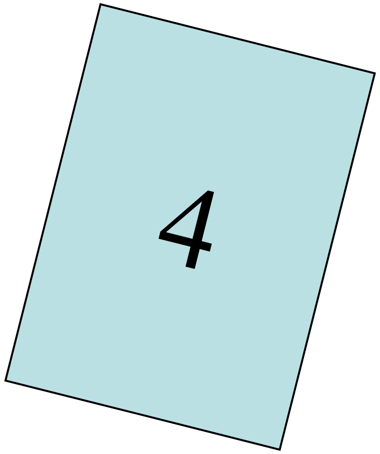
- Three card types: 2, 3, 4
- Infinite deck, twice as many 2's
- Start with 3 showing
- After each card, you say "high" or "low"
- New card is flipped
- If you're right, you win the points shown on the new card
- Ties are no-ops
- If you're wrong, game ends

- Differences from expectimax:
  - #1: get rewards as you go
  - #2: you might play forever!



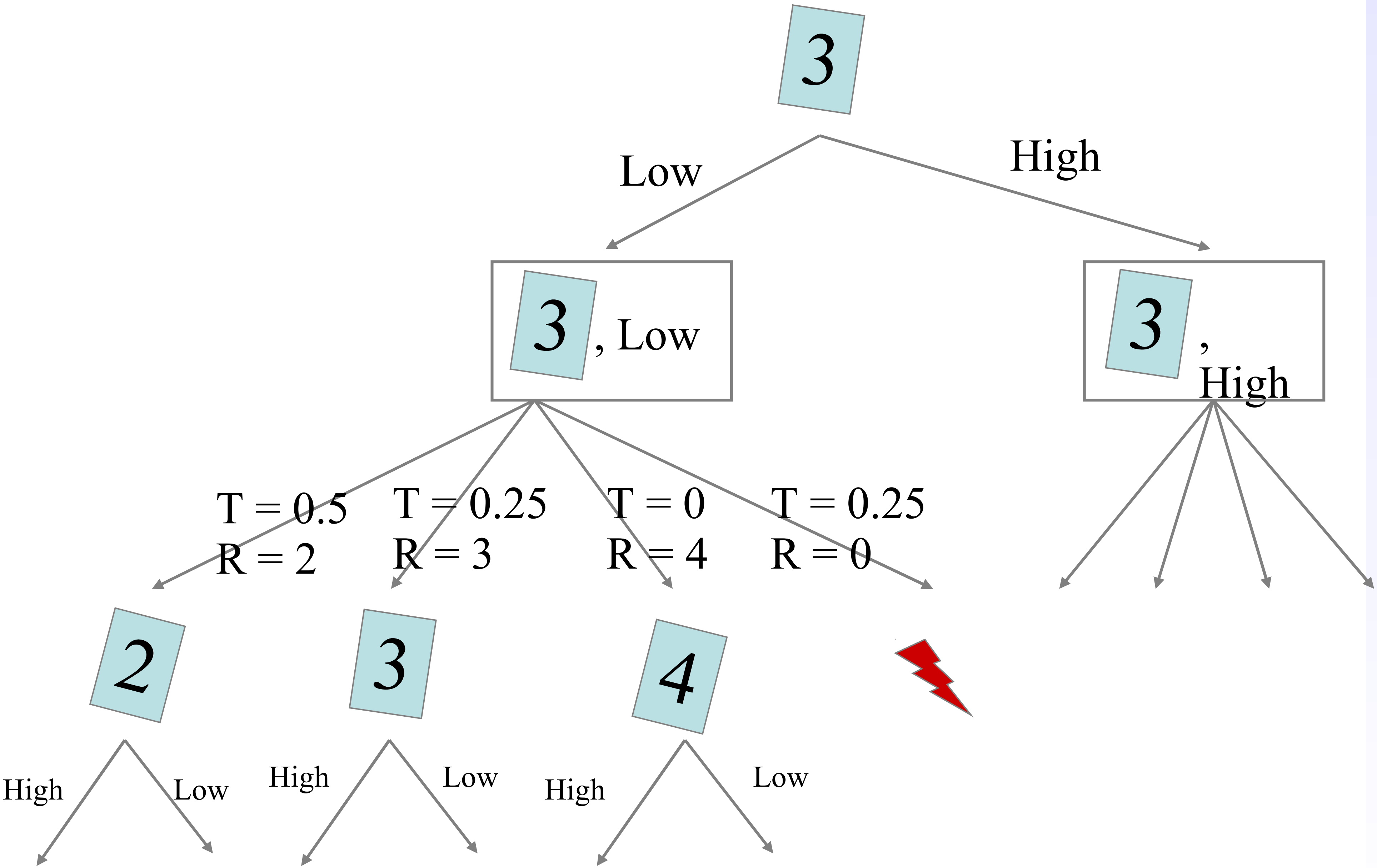
High-Low

- States: 2, 3, 4, done
- Actions: High, Low
- Model: T(s, a, s'):
  - P(s'=done | 4, High) = 3/4
  - P(s'=2 | 4, High) = 0
  - P(s'=3 | 4, High) = 0
  - P(s'=4 | 4, High) = 1/4
  - P(s'=done | 4, Low) = 0
  - P(s'=2 | 4, Low) = 1/2
  - P(s'=3 | 4, Low) = 1/4
  - P(s'=4 | 4, Low) = 1/4
  - …
- Rewards: R(s, a, s'):
  - Number shown on s' if s $\neq$ s'
  - 0 otherwise
- Start: 3



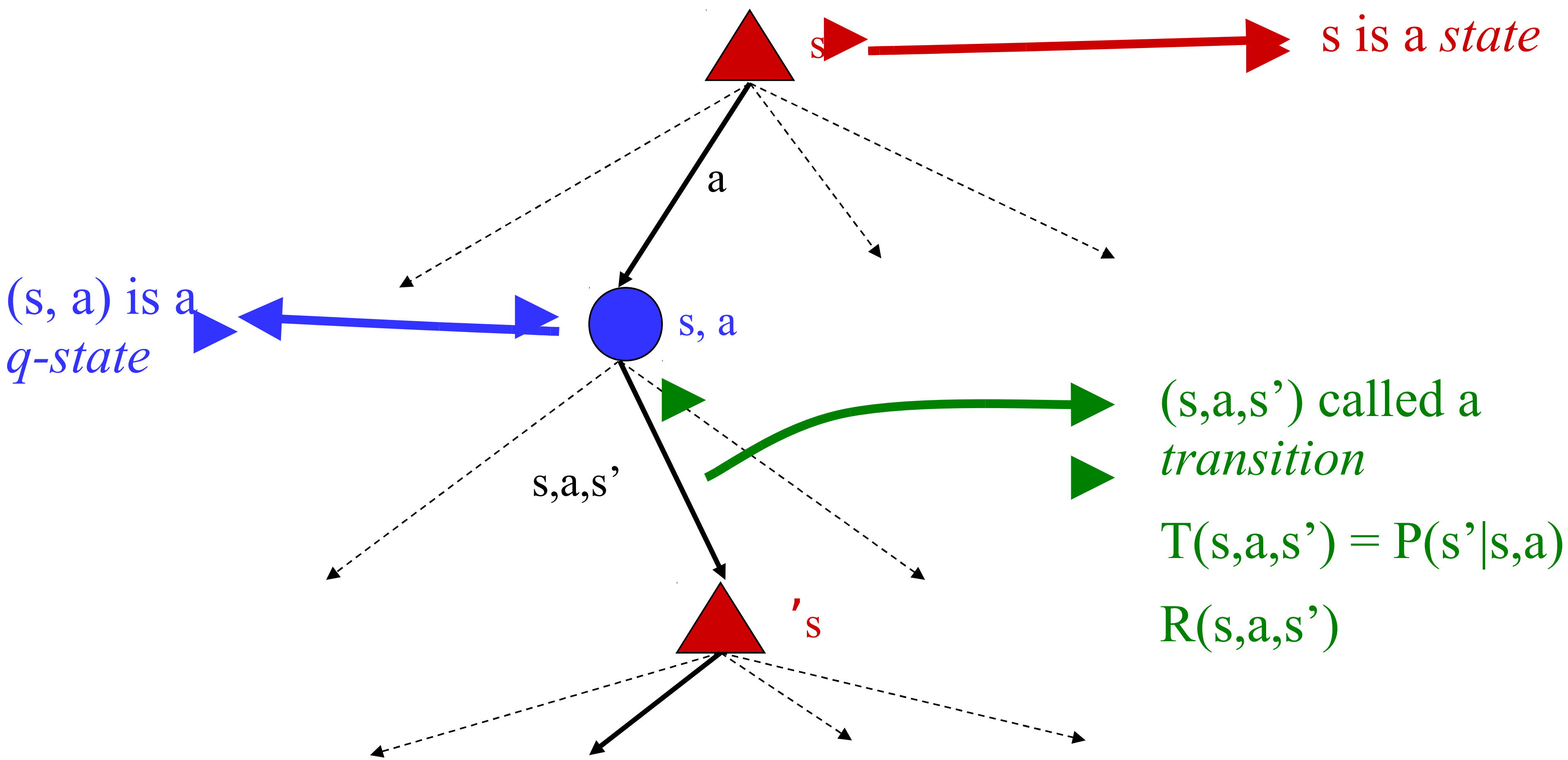
Note: could choose actions with search. How?

Example: High-Low



MDP Search Trees

- Each MDP state gives an expectimax-like search tree



Utilities of Sequences

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards
- Typically consider stationary preferences:

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

Assuming that reward depends only on state for these slides!

- Theorem: only two ways to define stationary utilities
  - Additive utility:

$$U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$$

  - Discounted utility:

$$U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$$

Infinite Utilities?!

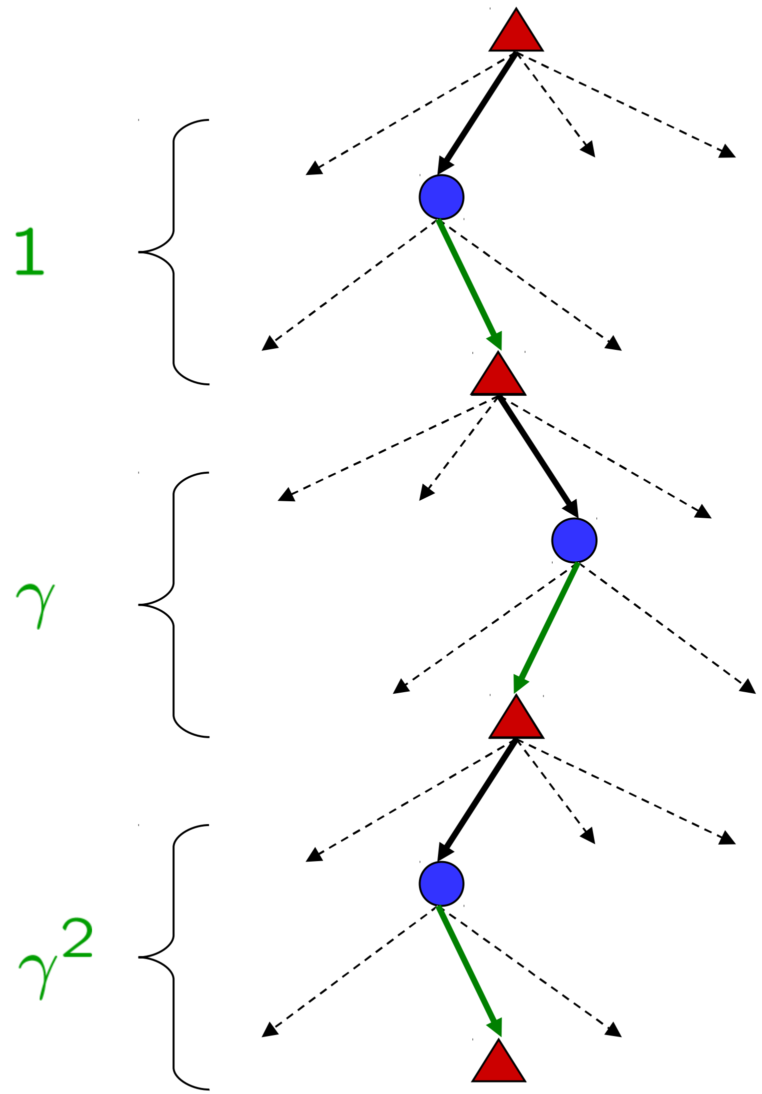
- Problem: infinite sequences with infinite rewards
- Solutions:
  - Finite horizon:
    - Terminate episodes after a fixed T steps
    - Gives nonstationary policy (π depends on time left)
  - Absorbing state(s): guarantee that for every policy, agent will eventually "die" (like "done" for High-Low)
  - Discounting: for $0 < \gamma < 1$

$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1-\gamma)$$

    - Smaller γ means smaller "horizon" – shorter term focus

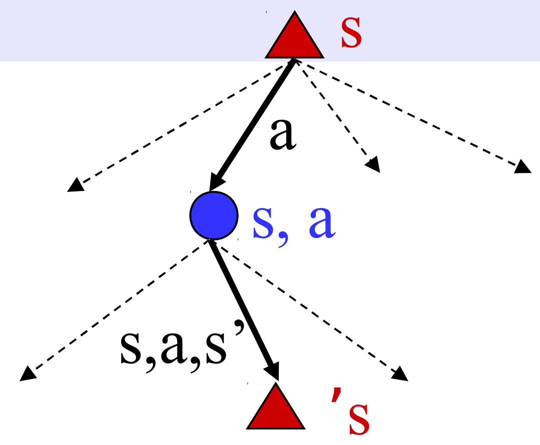
Discounting

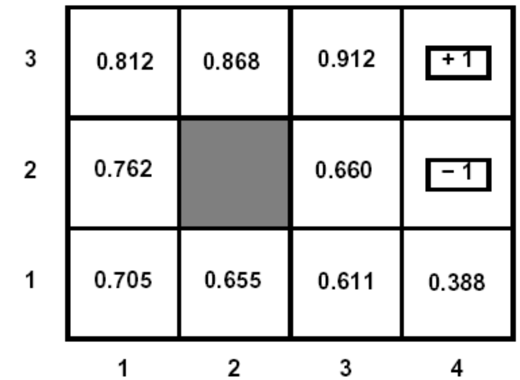
- Typically discount rewards by $\gamma < 1$ each time step
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge

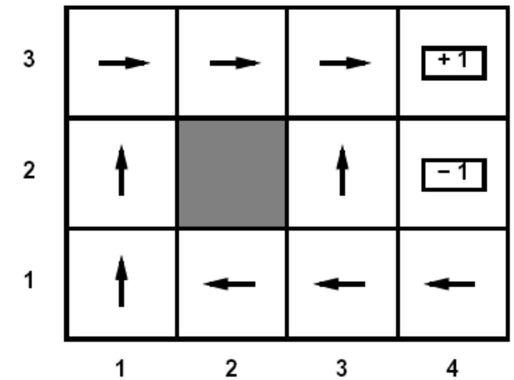


Optimal Utilities

- Fundamental operation: compute the optimal utilities of states s (all at once)
- Why? Optimal values define optimal policies!
- Define the utility of a state s:
  $V^*(s)$ = expected return starting in s and acting optimally
- Define the utility of a q-state (s,a):
  $Q^*(s,a)$ = expected return starting in s, taking action a and thereafter acting optimally
- Define the optimal policy:
  $\pi^*(s)$ = optimal action from state s



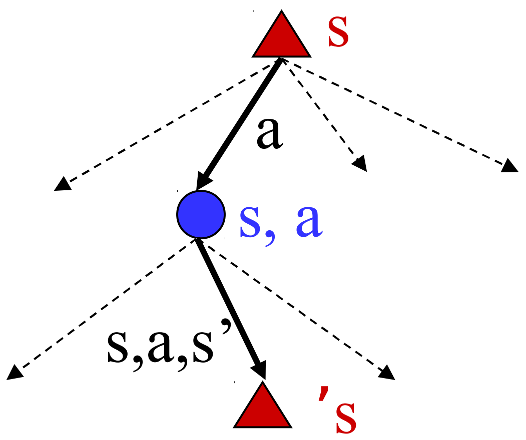




The Bellman Equations

- Definition of utility leads to a simple one-step lookahead relationship amongst optimal utility values:

  Optimal rewards = maximize over first action and then follow optimal policy



- Formally:

$$V^*(s) = \max_a Q^*(s,a)$$

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

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

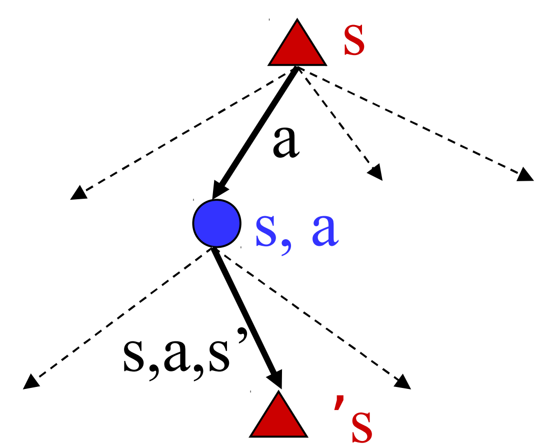
Solving MDPs

- We want to find the optimal policy π*
- Proposal 1: modified expectimax search, starting from each state s:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

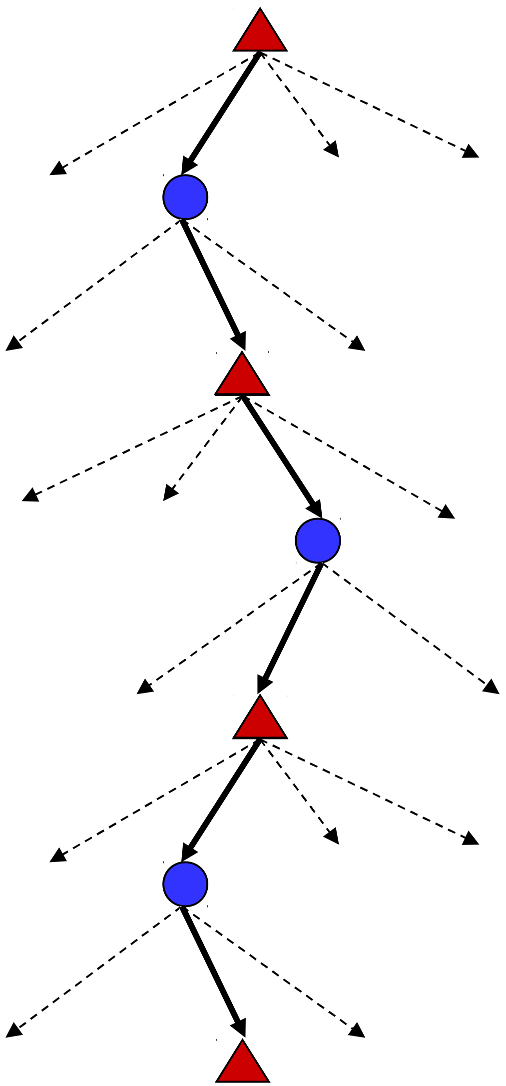
$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

$$V^*(s) = \max_a Q^*(s,a)$$



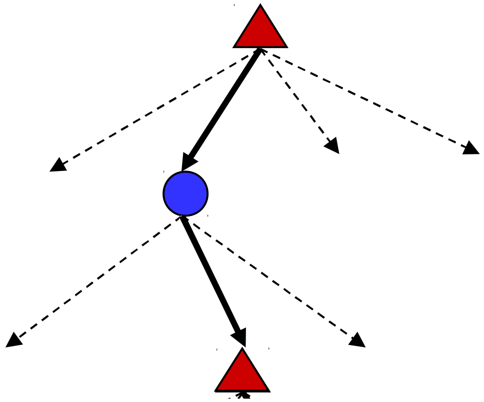
Why Not Search Trees?

- Why not solve with expectimax?
- Problems:
  - This tree is usually infinite (why?)
  - Same states appear over and over (why?)
  - We would search once per state (why?)
- Idea: Value iteration
  - Compute optimal values for all states all at once using successive approximations
  - Will be a bottom-up dynamic program similar in cost to memoization
  - Do all planning offline, no replanning needed!



Value Estimates

- Calculate estimates $V_k^*(s)$
  - Not the optimal value of s!
  - The optimal value considering only next k time steps (k rewards)
  - As $k \to \infty$, it approaches the optimal value
  - Why:
    - If discounting, distant rewards become negligible
    - If terminal states reachable from everywhere, fraction of episodes not ending becomes negligible
    - Otherwise, can get infinite expected utility and then this approach actually won't work



Memoized Recursion?

- Recurrences:

$$V_0^*(s) = 0$$

$$V_i^*(s) = \max_a Q_i^*(s,a)$$

$$Q_i^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_{i-1}^*(s') \right]$$

$$\pi_i(s) = \arg\max_a Q_i^*(s,a)$$

- Cache all function call results so you never repeat work
- What happened to the evaluation function?

Value Iteration

- Problems with the recursive computation:
  - Have to keep all the $V_k^*(s)$ around all the time
  - Don't know which depth $\pi_k(s)$ to ask for when planning
- Solution: value iteration
  - Calculate values for all states, bottom-up
  - Keep increasing k until convergence

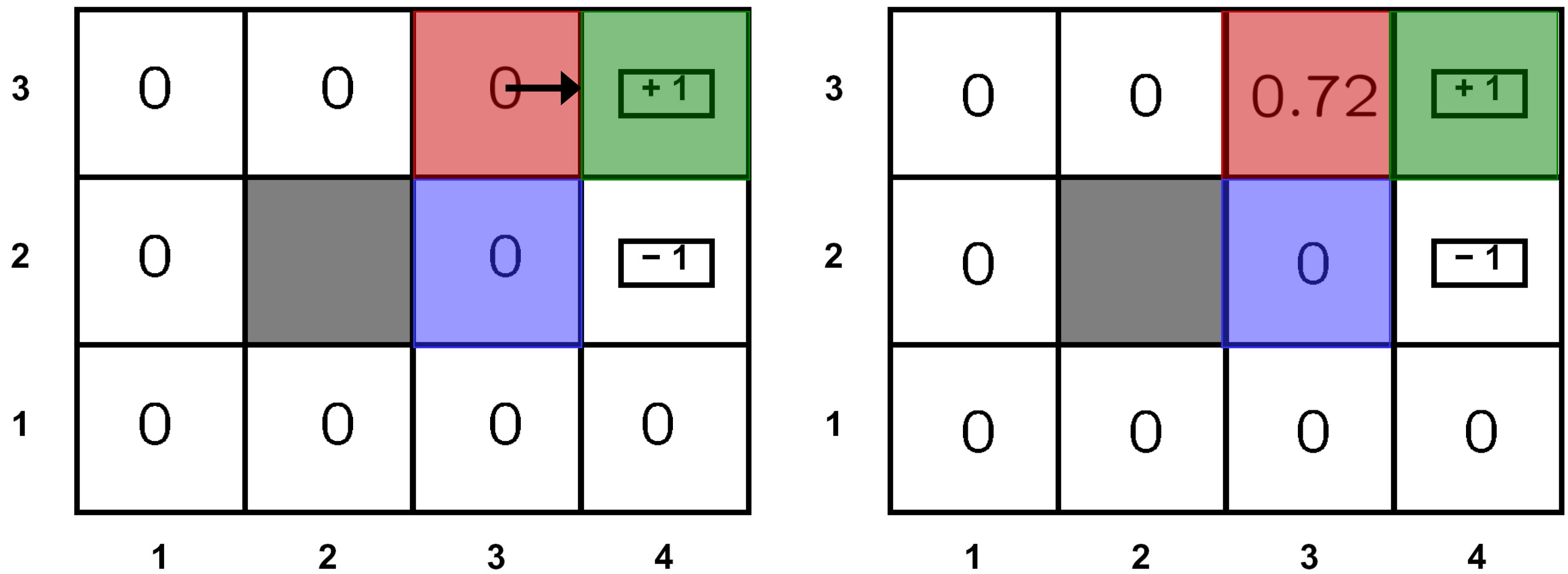
Value Iteration

- Idea:
  - Start with $V_0^*(s) = 0$, which we know is right (why?)
  - Given V_i^*, calculate the values for all states for depth i+1:

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

  - This is called a value update or Bellman update
  - Repeat until convergence
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

Example: Bellman Updates



$$V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma\, V_i(s') \right]$$

$$V_2(\langle 3,3\rangle) = \sum_{s'} T(\langle 3,3\rangle, \text{right}, s') \left[R(\langle 3,3\rangle) + 0.9\, V_1(s') \right]$$

$$= 0.9\, [0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0]$$

Example: Value Iteration

V_2

	1	2	3	4
3	0	0	0.72	+1
2	0		0	−1
1	0	0	0	0

V_3

	1	2	3	4
3	0	0.52	0.78	+1
2	0		0.43	−1
1	0	0	0	0

- Information propagates outward from terminal states and eventually all states have correct value estimates

Convergence*

- Define the max-norm: $||U|| = \max_s |U(s)|$
- Theorem: For any two approximations U and V

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

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

$$||U^{t+1} - U^t|| < \epsilon, \Rightarrow ||U^{t+1} - U|| < 2\epsilon\gamma/(1-\gamma)$$

  - I.e. once the change in our approximation is small, it must also be close to correct

Practice: Computing Actions

- Which action should we chose from state s:
  - Given optimal values V?

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

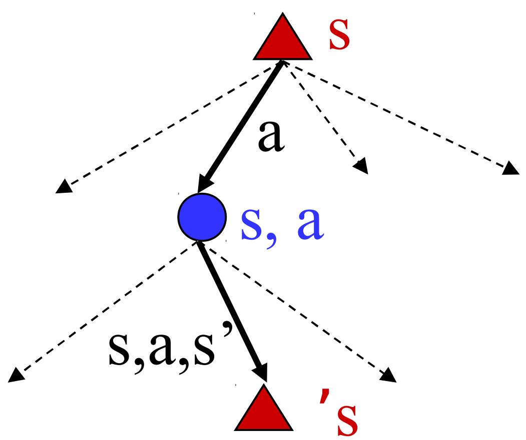
  - Given optimal q-values Q?

$$\arg\max_a Q^*(s,a)$$

  - Lesson: actions are easier to select from Q's!

Recap: MDPs

- Markov decision processes:
  - States S
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)
  - Start state s_0



- Quantities:
  - Returns = sum of discounted rewards
  - Values = expected future returns from a state (optimal, or for a fixed policy)
  - Q-Values = expected future returns from a q-state (optimal, or for a fixed policy)

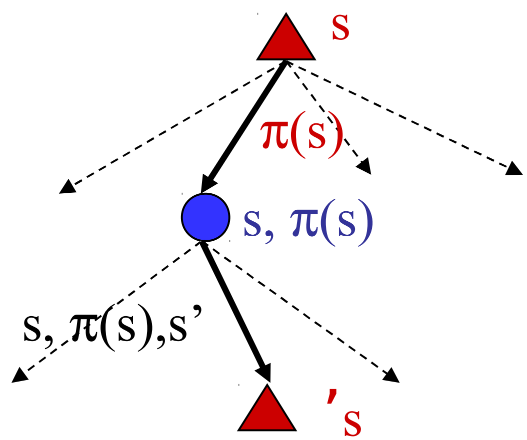
Utilities for Fixed Policies

- Another basic operation: compute the utility of a state s under a fix (general non-optimal) policy

- Define the utility of a state s, under a fixed policy π:

  Vπ(s) = expected total discounted rewards (return) starting in s and following π

- Recursive relation (one-step look-ahead / Bellman equation):



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

Policy Evaluation

- How do we calculate the V's for a fixed policy?
- Idea one: turn recursive equations into updates

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

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

- Idea two: it's just a linear system, solve with Matlab (or whatever)

Policy Iteration

- Alternative approach:
  - Step 1: Policy evaluation: calculate utilities for a fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step lookahead with resulting converged (but not optimal!) utilities
  - Repeat steps until policy converges
- This is policy iteration
  - It's still optimal!
  - Can converge faster under some conditions

Policy Iteration

- Policy evaluation: with fixed current policy π, find values with simplified Bellman updates:
  - Iterate until values converge

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

- Policy improvement: with fixed utilities, find the best action according to one-step look-ahead

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_k}(s') \right]$$

Comparison

- In value iteration:
  - Every pass (or “backup”) updates both utilities (explicitly, based on current utilities) and policy (possibly implicitly, based on current policy)
- In policy iteration:
  - Several passes to update utilities with frozen policy
  - Occasional passes to update policies
- Hybrid approaches (asynchronous policy iteration):
  - Any sequences of partial updates to either policy entries or utilities will converge if every state is visited infinitely often