Markov Decision Processes

Dr. Demetrios Glinos
University of Central Florida

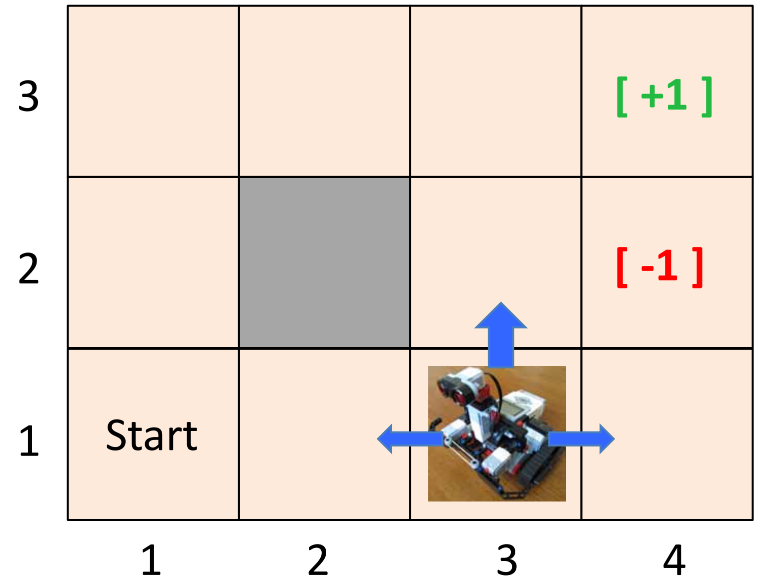
CAP4630 – Artificial Intelligence

Today

- Markov Decision Processes
- Value Iteration
- Policy Evaluation
- Policy Extraction
- Policy Iteration
- Summary of MDP Algorithms

The Grid World Problem Domain

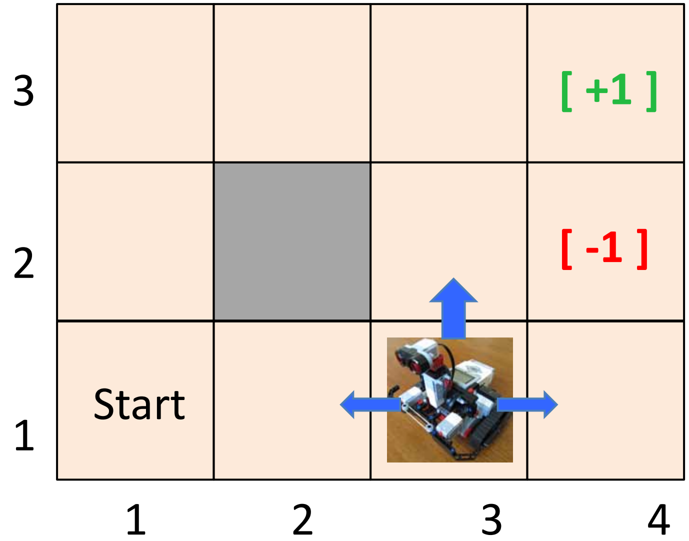
- Grid World
  - Agent lives in the grid
  - Walls block the agent's path
  - Agent cannot move onto gray cell
  - Agent can exit from terminal cells
- Movement is nondeterministic
  - 80% go in intended direction
  - 10% go in 90° clockwise direction
  - 10% go in 90° counterclockwise direction
- Agent gets a reward each step
  - Small "living" reward for nonterminal state
  - Large rewards at end (good & bad)



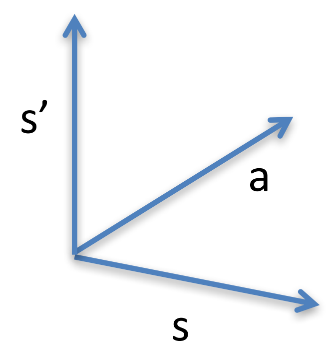
If agent tries to move into a wall or the gray cell, it stays where it started (no change)

Markov Decision Process

- An MDP is
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition model P(s’ | s, a)
  - A reward function R(s)
  - A start state s_0
  - Possibly one or more terminal states



- Transitions have the Markov property
  - P(s’ | s, a) does not depend on how the agent got to state s
  - we can think of the transition model as a 3-D table of probabilities (for now)

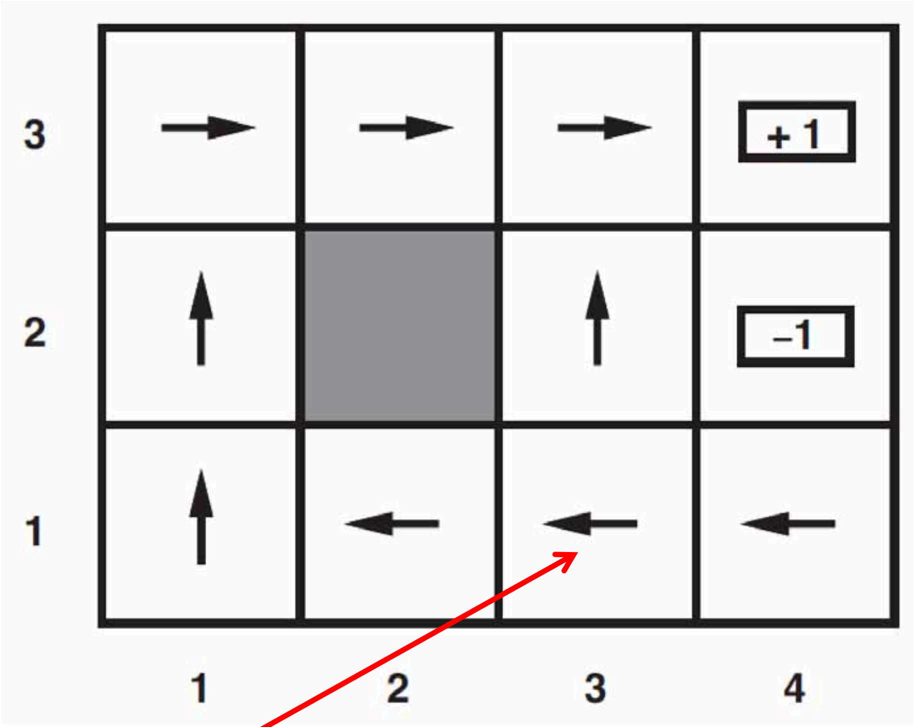


Policies

- Solution *cannot* be a fixed sequence of actions
- Solution must specify what to do for every state
- We call such a solution a **policy, π**
  - π(s) is the action to take for state s
  - much like a look up table or cipher code book
- Executing a policy produces nondeterministic (stochastic) results
- An optimal policy must *maximize expected utility*

Optimal Policy Example

- R(s) = - 0.04 (a small negative reward for entering each non-terminal state)

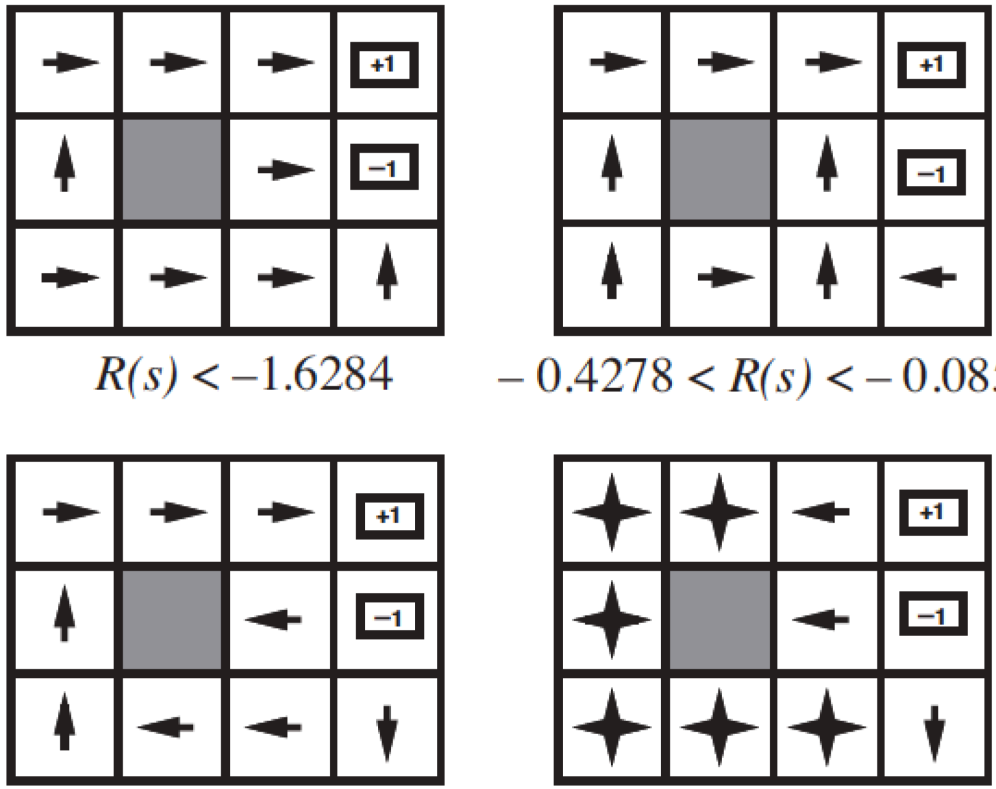


Q: Why go left from (3,1) ?

source: Fig. 17.2(a)

Balancing Risks and Rewards

These are all optimal for their respective reward function ranges



$R(s) < -1.6284$

$-0.4278 < R(s) < -0.0850$

$-0.0221 < R(s) < 0$

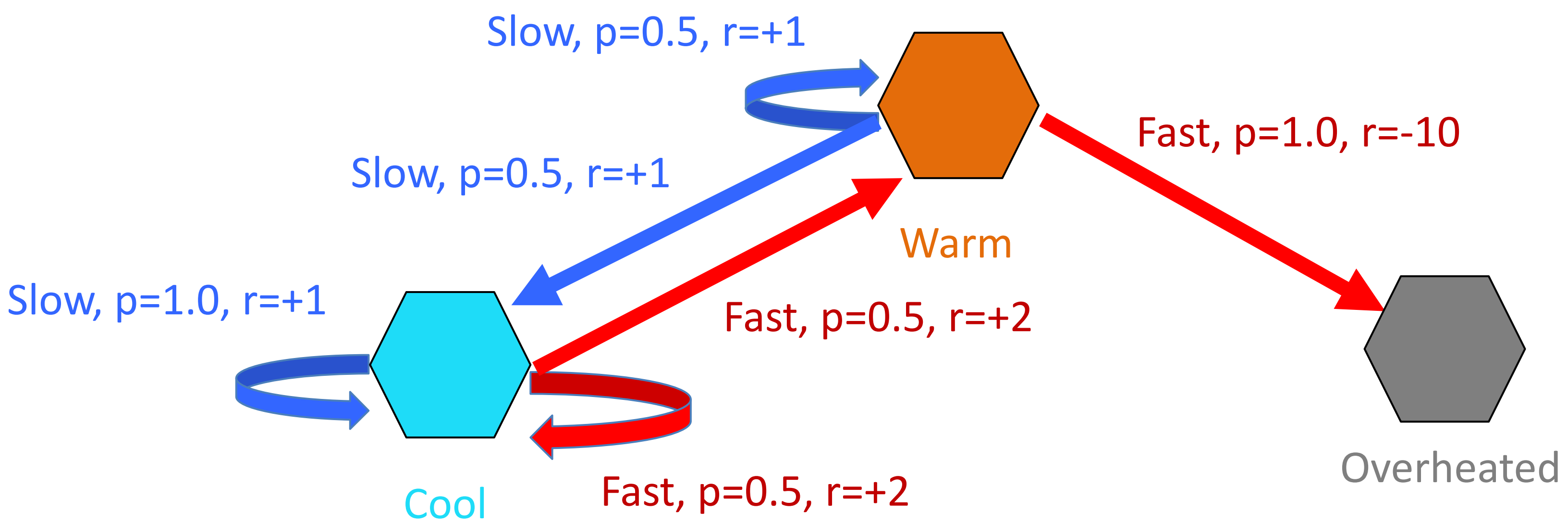
$R(s) > 0$

source: Fig. 17.2(b)

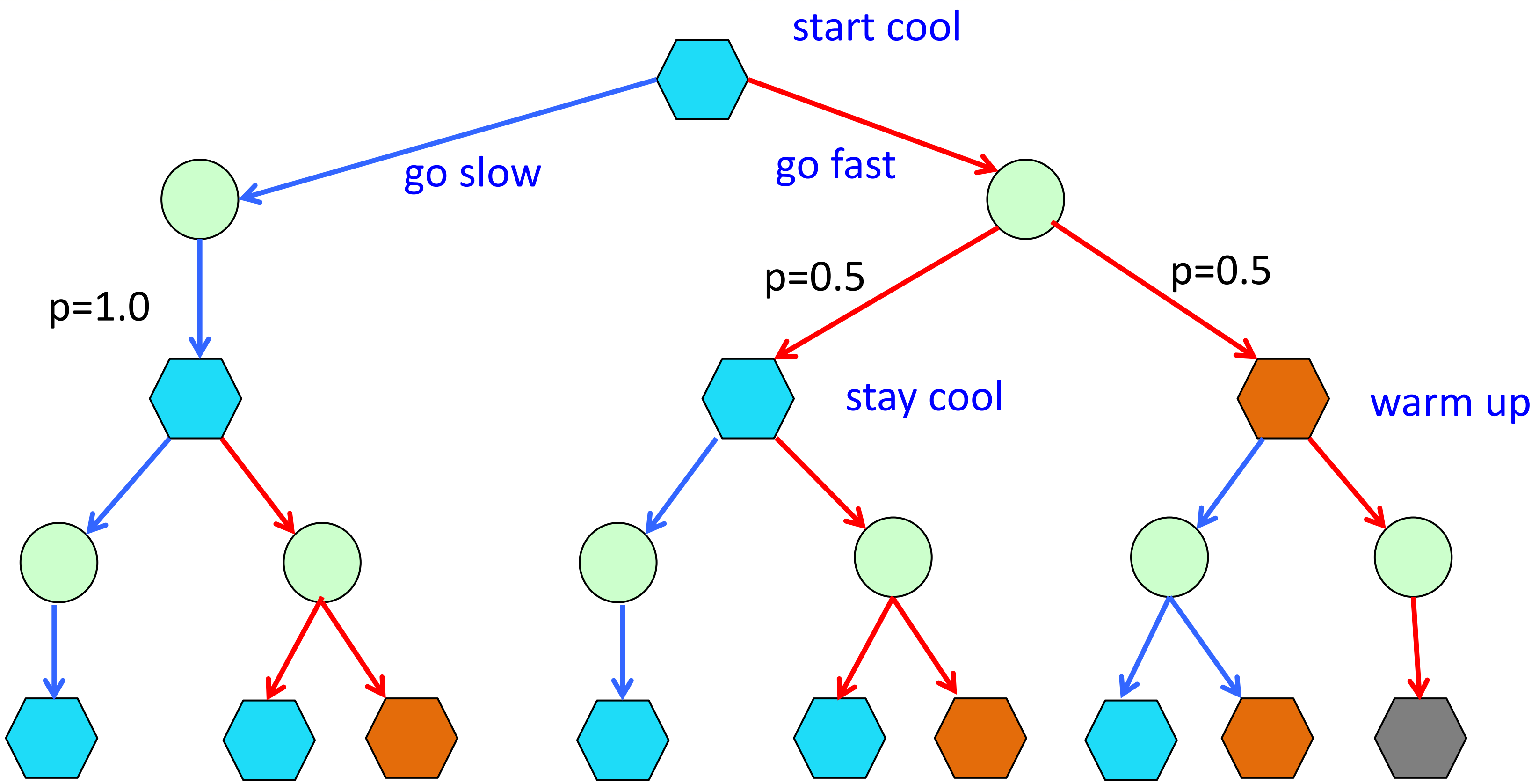
Example: Hiking the Grand Canyon

Grand Canyon Hiking Problem

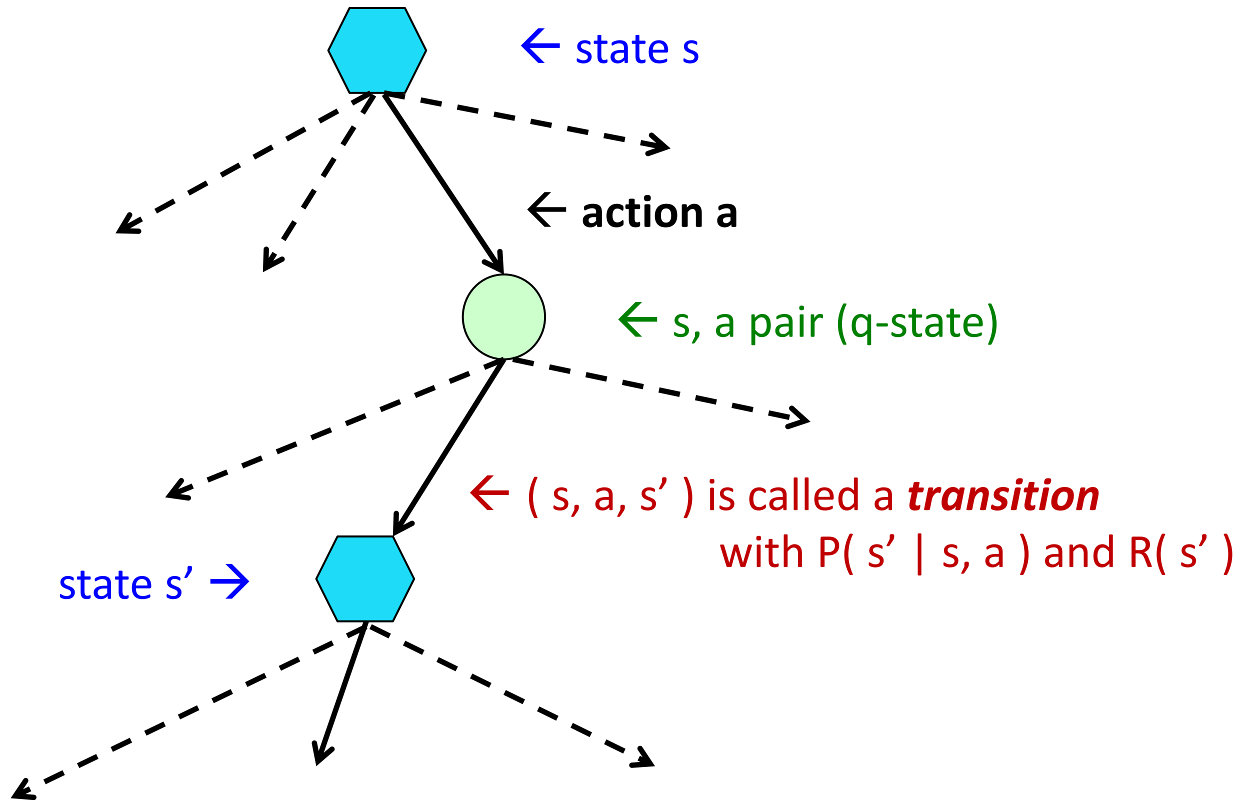
- Hiker wants to get to the bottom quickly, but must pace himself/herself
- Three states: Cool, Warm, Overheated
- Two actions: Slow, Fast
- Going faster gets double reward



Hiking Search Tree



MDP Search Trees



Each MDP state projects an expectimax search tree

Horizons

- We measure time by the number of moves (a sequence)
- We can have a **finite horizon** for decision making
  - After N moves, game over
  - $U_h([s_0, s_1, \ldots, s_{N+k}]) = U_h([s_0, s_1, \ldots, s_N])$, for all $k > 0$
  - e.g., for Grid World, if agent at (3,1) and N=3, the agent must head straight for the +1 terminal state, but for N=100, can take safe route to left

→ optimal policy for finite horizon is nonstationary (changes over time)

Compare:

> With an infinite horizon, optimal policy is stationary;
> it can (but is not required to) lead agent to a terminal state

Stationarity

- **Stationarity** is the assumption that the agent's preferences between state sequences are stationary

  → if agent prefers [s_1, s_2, ...] over [s'_1, s'_2, ...] then the agent should also prefer [s_0, s_1, s_2, ...] over [s_0, s'_1, s'_2, ...]

- Given stationarity, there are just two coherent ways to assign utilities to sequences

  - Additive rewards: $U_h([\, s_0, s_1, s_2, \ldots \,]) = R(s_0) + R(s_1) + R(s_2) + \ldots$

  - Discounted rewards: $U_h([\, s_0, s_1, s_2, \ldots \,]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$

    where **γ is a discount factor** between 0 and 1 and represents preference for sooner rather than later

Games Without End

- Problem: If sequences are infinitely long, undiscounted rewards will generally converge to +/- ∞
  - tough to compare policies

- Solutions:

  1. Discount with $\gamma < 1$ and rewards bounded by +/- R_{max}
     - the smaller the value of γ, the smaller the horizon
  2. Proper policy: set up the game so that every sequence will end up in a terminal state eventually
  3. Finite Horizon: similar to depth-limited search
     - but policies become nonstationary
  4. Average reward per turn as basis for comparison of policies

Today

- Markov Decision Processes
- Value Iteration
- Policy Evaluation
- Policy Extraction
- Policy Iteration
- Summary of MDP Algorithms

Terminology for Optimality

- Value (utility) of a state s:

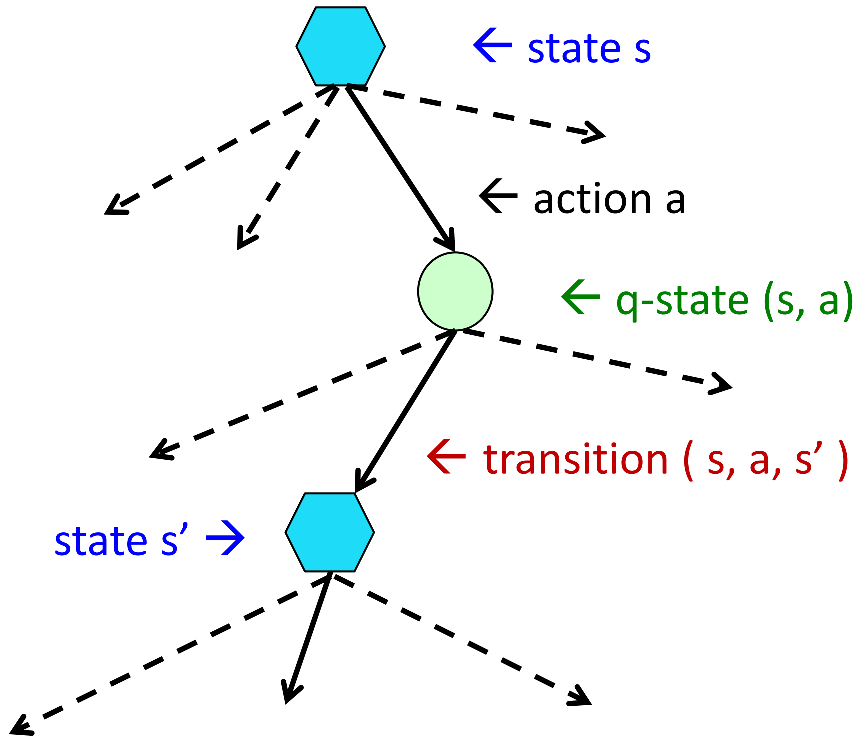
  V*(s) = expected utility acting optimally and starting from state s

- Value (utility) of a q-state (s, a):

  Q*(s, a) = expected utility acting optimally having taken action a from state s

- Optimal policy:

  π*(s) = the optimal action to take from state s

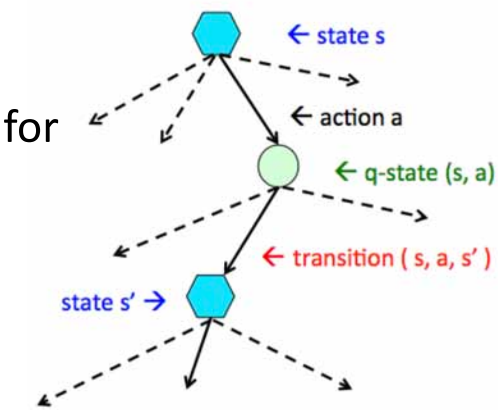


Recursive Definition of Value

Basic idea:

- Compute the *expectimax* value of a state
- this represents the expected utility under optimal action
- use average sum (expected value) of discounted rewards for future values

→ this gives us the "Bellman equations"



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

$$Q^*(s,a) = \sum_{s'} P(s' \mid s,a)[R(s') + \gamma V^*(s')]$$

Thus $$\boxed{V^*(s) = \max_a \sum_{s'} P(s' \mid s,a)[R(s') + \gamma V^*(s')]}$$

and $$\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} V^*(s)$$

Value Iteration Concept

- Bellman equations form a system of equations
  - n equations, one for each state
  - n unknown state utilities

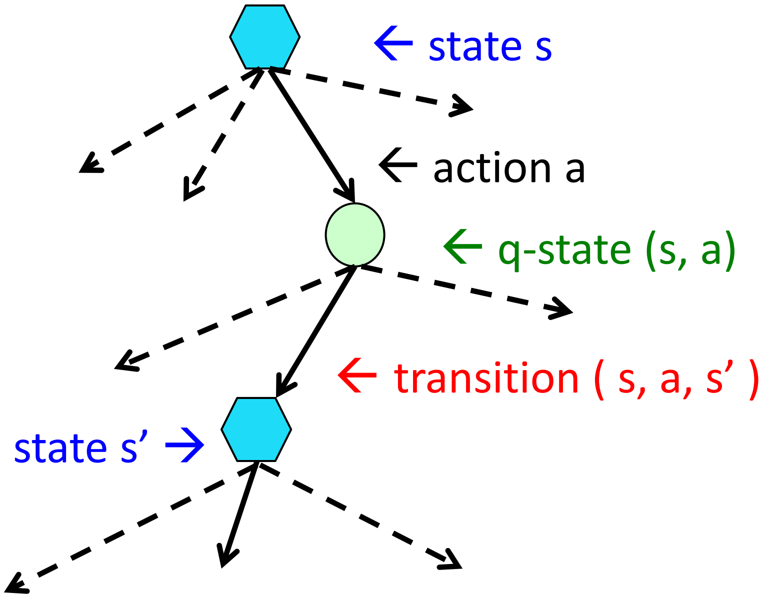
$$V^*(s) = \max_a \sum_{s'} P(s' \mid s,a)[R(s') + \gamma V^*(s')]$$

- This system is nonlinear
  - since use the "max" operator
  - can't use linear algebra methods
  - use iteration instead

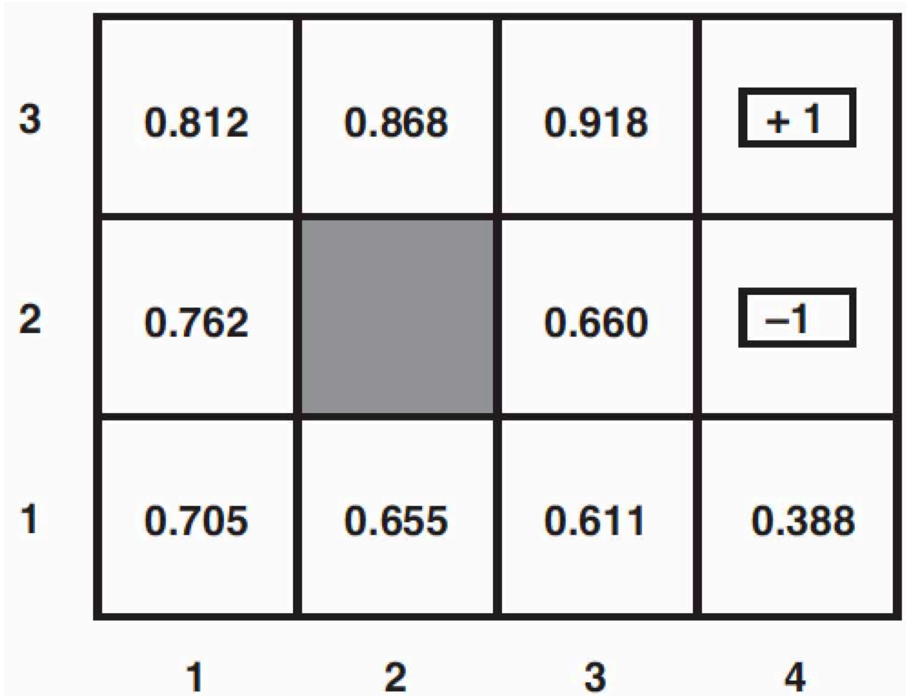
- Basic idea (similar to Jacobi iteration for linear systems)
  1. start with arbitrary initial values for utilities
  2. calculate new values for utilities based on current values
  3. repeat until convergence criteria satisfied

Value Iteration Algorithm

- Start with zero vector: $V_0(s) = 0$, for all s
- Compute for all s: $V_{k+1}(s) = \max_a \sum_{s'} P(s' \mid s, a)[R(s') + \gamma V_k(s')]$
- Repeat until convergence
  - convergence guaranteed if $\gamma < 1.0$
  - solution is unique
- Complexity of each iteration: $O(S^2 A)$



Grid World V*(s) Values



source: Fig. 17.3

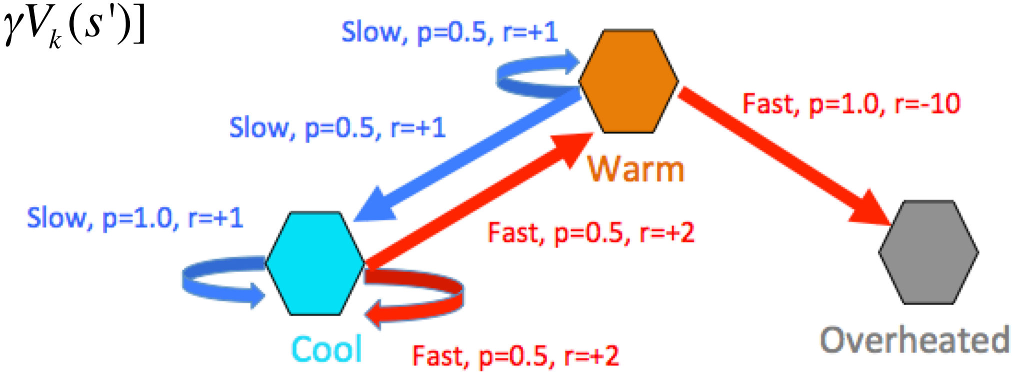
Above utilities are for $\gamma = 1$ and R(s) = -0.04 for nonterminal states.
Values reflect proximity to +1 terminal state and possible outcomes

Q: What is the optimal policy starting from (1,1) ? From (1,3) ?

Example: Value Iteration

$$V_{k+1}(s) = \max_{a} \sum_{s'} P(s' \mid s,a)[R(s') + \gamma V_k(s')]$$

Assume no discount (γ = 1)
Rewards as shown



Example calculation for V_1(cool) :

Start with V_0(cool) = V_0(warm) = V_0(overheated) = 0

V_1(cool) = $\max_a$ [P(cool|cool,fast)(R(cool) + V_0(cool)) + P(warm|cool,fast)(R(warm) + V_0(warm)),

P(cool|cool,slow)(R(cool) + V_0(cool)) + P(warm|cool,slow)(R(warm) + V_0(warm))]

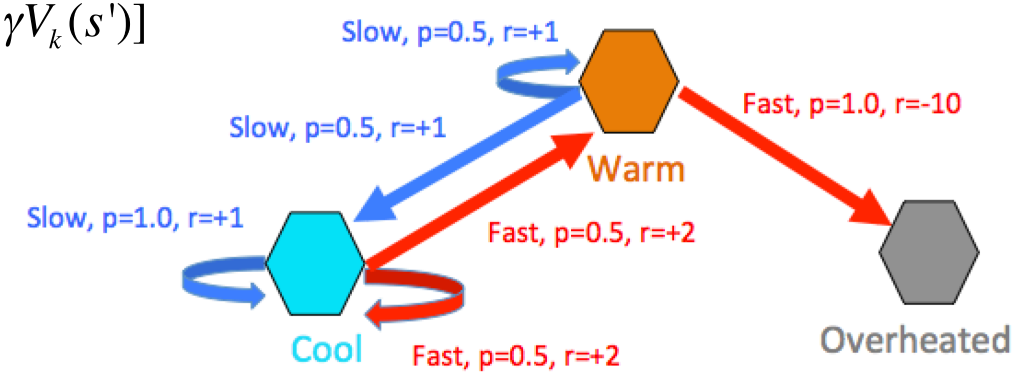
= $\max_a$ [(0.5)(2 + 0) + (0.5)(2 + 0) , (1.0)(1 + 0) + (0.0)(2 + 0)]

= $\max_a$ [$2_{a=fast}$, $1_{a=slow}$] = 2 (fast)

Example: Value Iteration

$$V_{k+1}(s) = \max_a \sum_{s'} P(s' \mid s,a)[R(s') + \gamma V_k(s')]$$

Assume no discount (γ = 1)
Rewards as shown



Iteration	Cool	Warm	Overheated
V_0	0	0	0
V_1	2 (fast)	1 (slow)	0
V_2	3.5 (fast)	2.5 (slow)	0

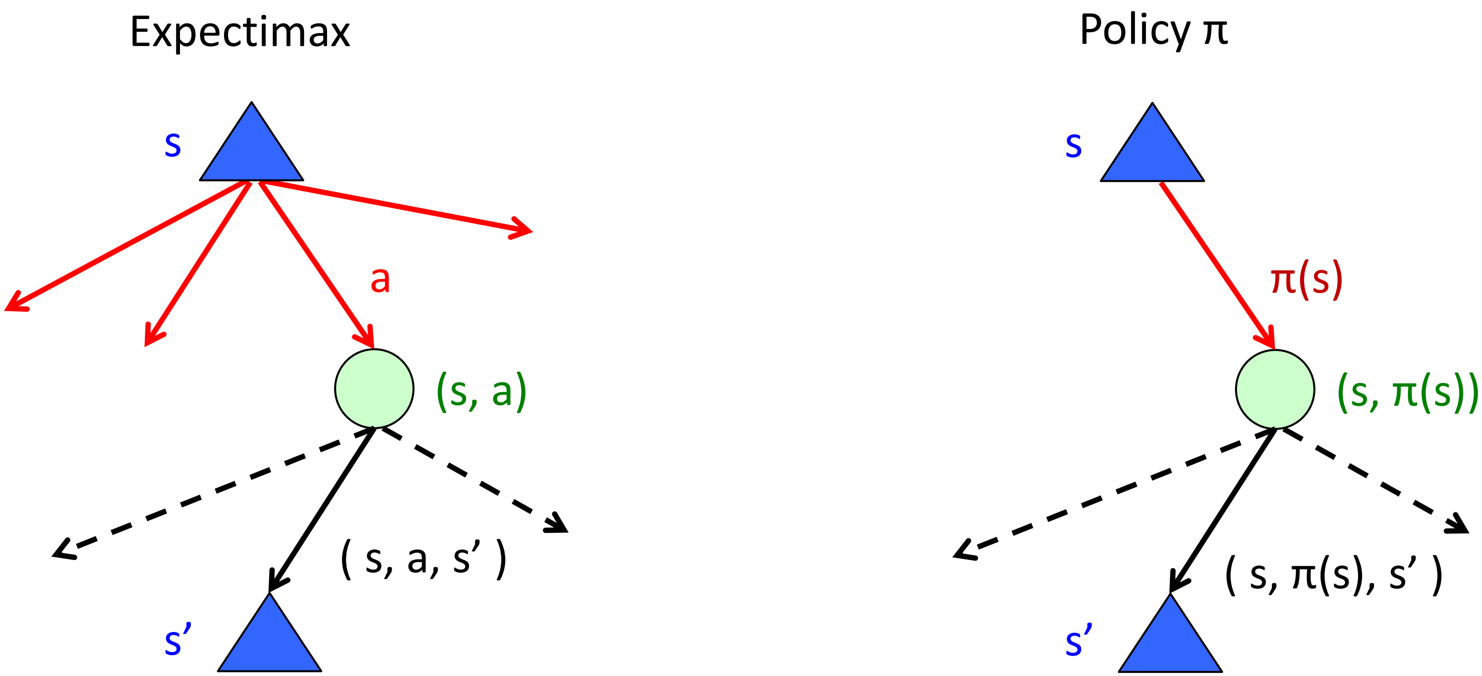
Final policy: Go as fast as possible without risking overheating (i.e., if cool, go fast, but if warm, go slow)

Homework: Verify results in table for V_1(warm), V_2(cool), and V_2(warm)

Today

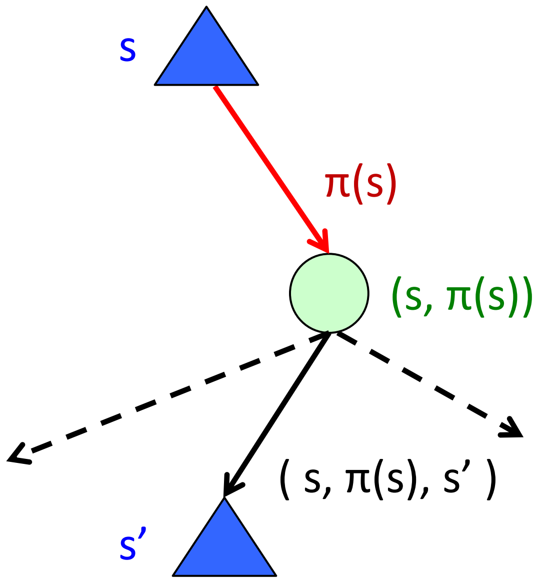
- Markov Decision Processes
- Value Iteration
- Policy Evaluation
- Policy Extraction
- Policy Iteration
- Summary of MDP Algorithms

A Policy is Fixed



- Expectimax computes max over all actions to compute optimal value
- Policy computes a value that is not necessarily optimal, but tree much simpler

Utilities for a Policy



- Utility of a state s under a fixed policy π is expected total discounted rewards starting from s and following policy π, which we denote by $V^{\pi}(s)$
- Bellman equation for a fixed policy:

$$V^{\pi}(s) = \sum_{s'} P(s' \mid s, \pi(s))[R(s') + \gamma V^{\pi}(s')]$$

Example: Policy Evaluation

[-1]	[+1]	[-1]
[-1]	-0.04	[-1]
[-1]	-0.04	[-1]
[-1]	-0.04	[-1]

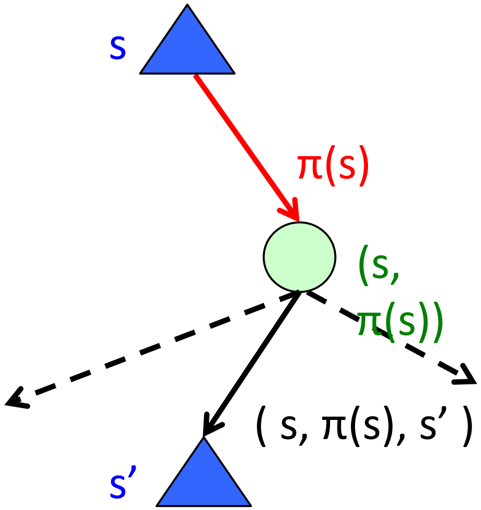
Policy 1:

- Always go left

Policy 2:

- Always go forward

Calculating the Values



- Basic idea: Iterate, just like for value iteration

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

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} P(s' \mid s, \pi(s))[R(s') + \gamma V_k^{\pi}(s')]$$

- Complexity: $O(S^2)$ per iteration

- This is just a linear system! (total complexity $O(kS^2) \approx O(n^3)$)

Today

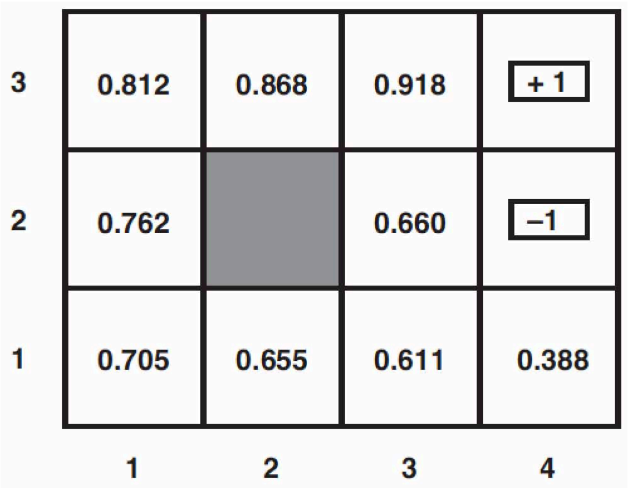
- Markov Decision Processes
- Value Iteration
- Policy Evaluation
- Policy Extraction
- Policy Iteration
- Summary of MDP Algorithms

Extracting Actions from Values

Suppose we are given the optimal values V*(s)

Q: What is the policy for this data?

Q: What action should we take in each state?



source: Fig. 17.3

- We do a one-step expectimax to find out:

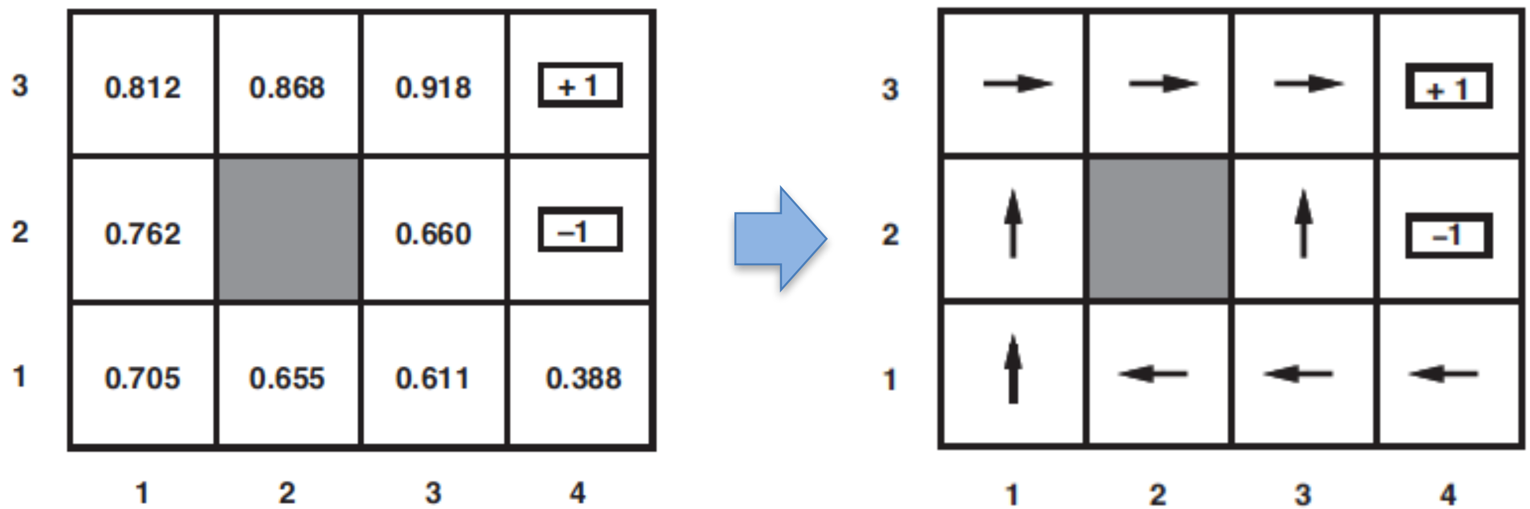
$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} V^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a)[R(s') + \gamma V^*(s')]$$

Q: Why is only one step needed?

Policy Extraction

Values

Policy



source: Fig. 17.3

source: Fig. 17.2(a)

Above utilities (values) are for γ = 1 and R(s) = -0.04 for nonterminal states.

Values reflect proximity to +1 terminal state and possible outcomes

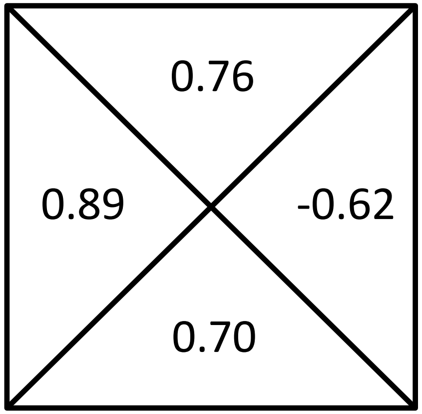
Extracting Actions from Q-Values

- This situation is even easier
  - Q-values give us values for each action directly
  - Trivial to choose the largest value

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

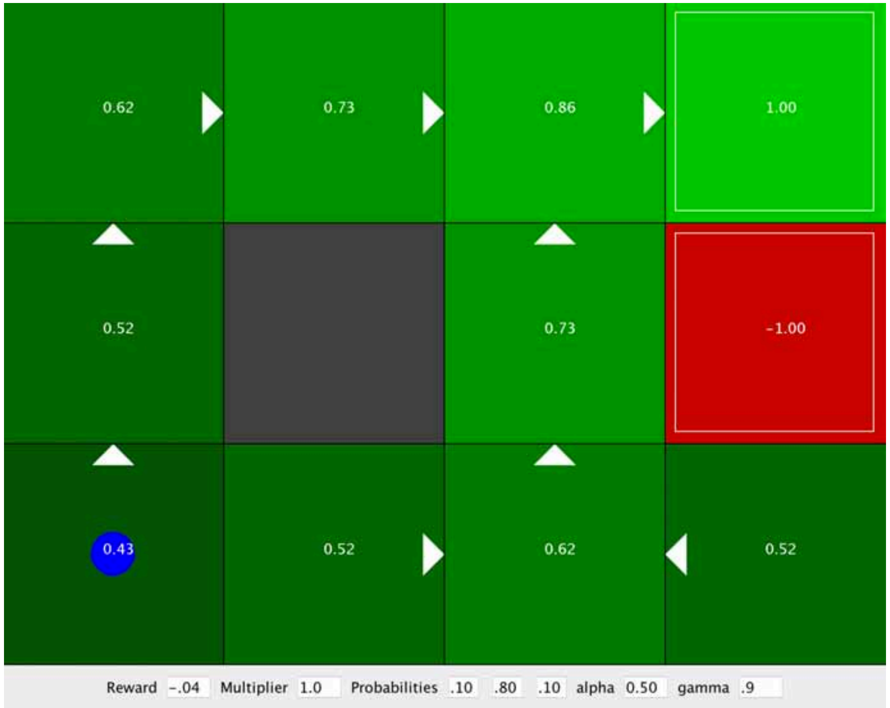
The policy for state s is to take the action from s that has the highest q-value

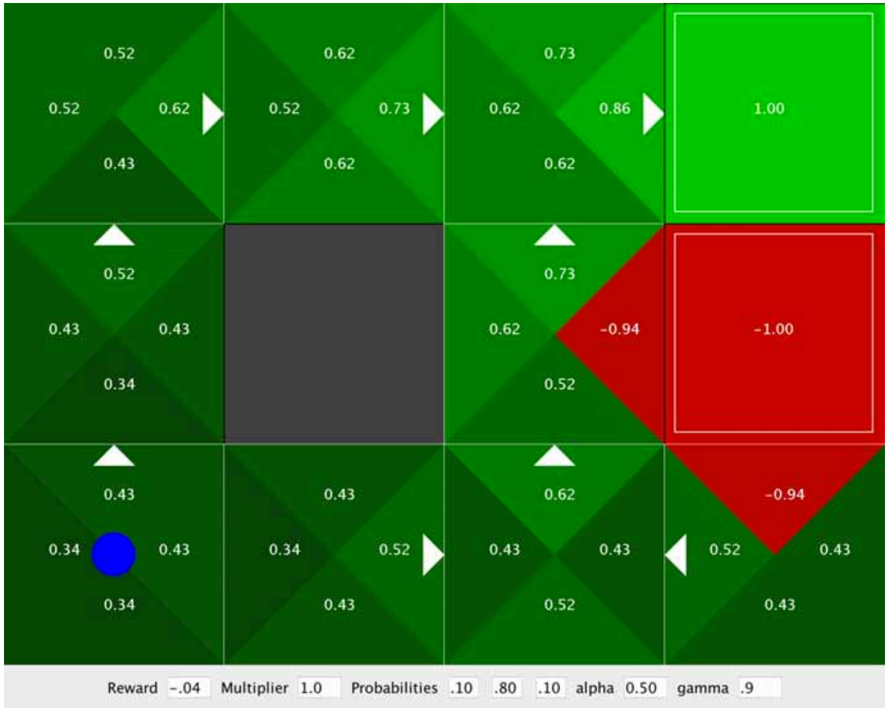
This produces the same policy that we obtain by moving to the adjacent state with the highest state value



(q-values for a sample Grid World cell)

Policy Extraction Comparison





Policies extracted from values and q-values are the same

Today

- Markov Decision Processes
- Value Iteration
- Policy Evaluation
- Policy Extraction
- Policy Iteration
- Summary of MDP Algorithms

Motivation

- Value iteration algorithm is somewhat slow: O (S^2A) per iteration
- The “max” (hence, the action to choose) at each state changes slowly
- The policy often converges well before the values
- Approach: iterate the policy, not the values

Policy Iteration

- Basic idea: Iterate, but not everything at once ("asynchronous policy iteration")
  - Step 1: Policy Evaluation
    - Calculate utilities for a given policy until convergence
  - Step 2: Policy Improvement
    - Update policy using one-step look-ahead
    - Inputs (for the "future" values) are the values found in Step 1
- This is still optimal:
  - As long as compute all values infinitely many times in the limit, this will converge to the optimal

Computing Values and Actions

- Step 1 (Evaluation):
  - Given a fixed current policy π_i, iterate (over k) until convergence:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} P(s' \mid s, \pi_i(s))[R(s') + \gamma V_k^{\pi_i}(s')]$$

- Step 2 (Improvement):
  - For fixed values (computed in Step 1), get a better policy using one-step look-ahead policy extraction:

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a)[R(s') + \gamma V^{\pi_i}(s')]$$

- Repeat Steps 1 and 2 until the policy converges

Summary of MDP Algorithms

Task	Algorithm
Compute optimal values:	use value iteration or policy iteration
Compute values for a particular policy:	use policy evaluation
Compute policy from values:	use policy extraction

Similarities:

All are variations of the Bellman equations
All use one-step look-ahead expectimax

Differences:

max over actions v. fixed policy