CS440/ECE448 Lecture 25: Markov Decision Processes

Slides by Svetlana Lazebnik, 11/2016

Modified by Mark Hasegawa-Johnson, 11/2017

Markov Decision Processes

- In HMMs, we see a sequence of observations and try to reason about the underlying state sequence
  - There are no actions involved
- But what if we have to take an action at each step that, in turn, will affect the state of the world?

Markov Decision Processes

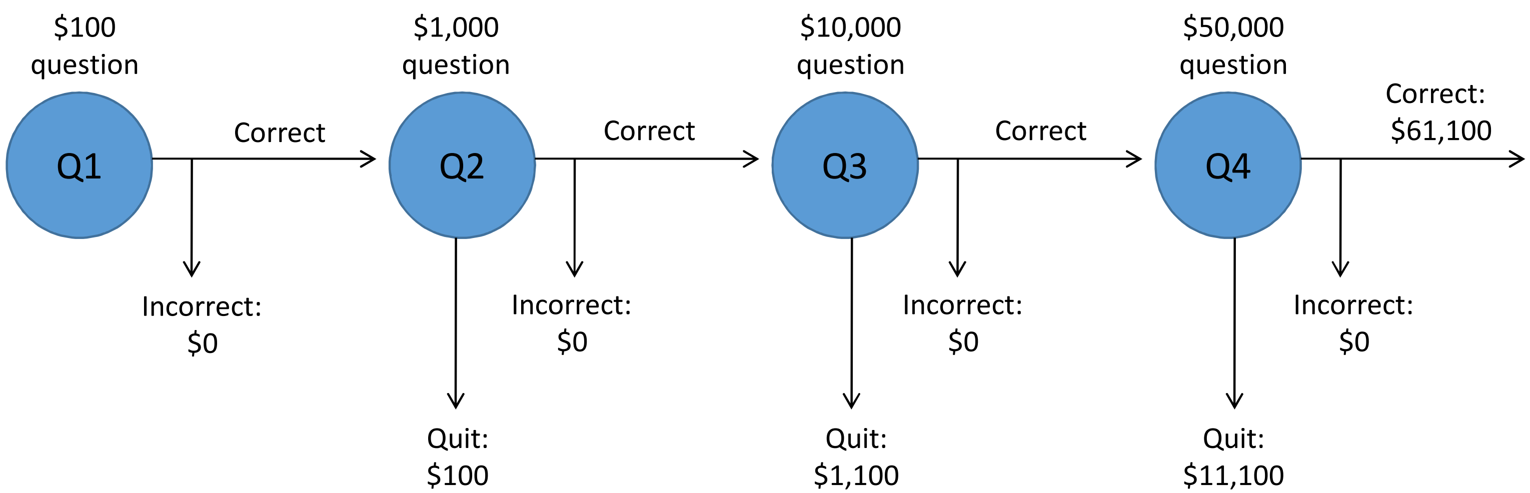
- Components that define the MDP. Depending on the problem statement, you either know these, or you learn them from data:
  - **States** s, beginning with initial state s_0
  - **Actions** a
    - Each state s has actions $A(s)$ available from it
  - **Transition model** $P(s' \mid s, a)$
    - *Markov assumption*: the probability of going to s' from s depends only on s and a and not on any other past actions or states
  - **Reward function** $R(s)$
- **Policy – the "solution" to the MDP:**
  - $\pi(s) \in A(s)$: the action that an agent takes in any given state

Overview

- First, we will look at how to “solve” MDPs, or find the optimal policy when the transition model and the reward function are known
- Second, we will consider **reinforcement learning**, where we don’t know the rules of the environment or the consequences of our actions

Game show

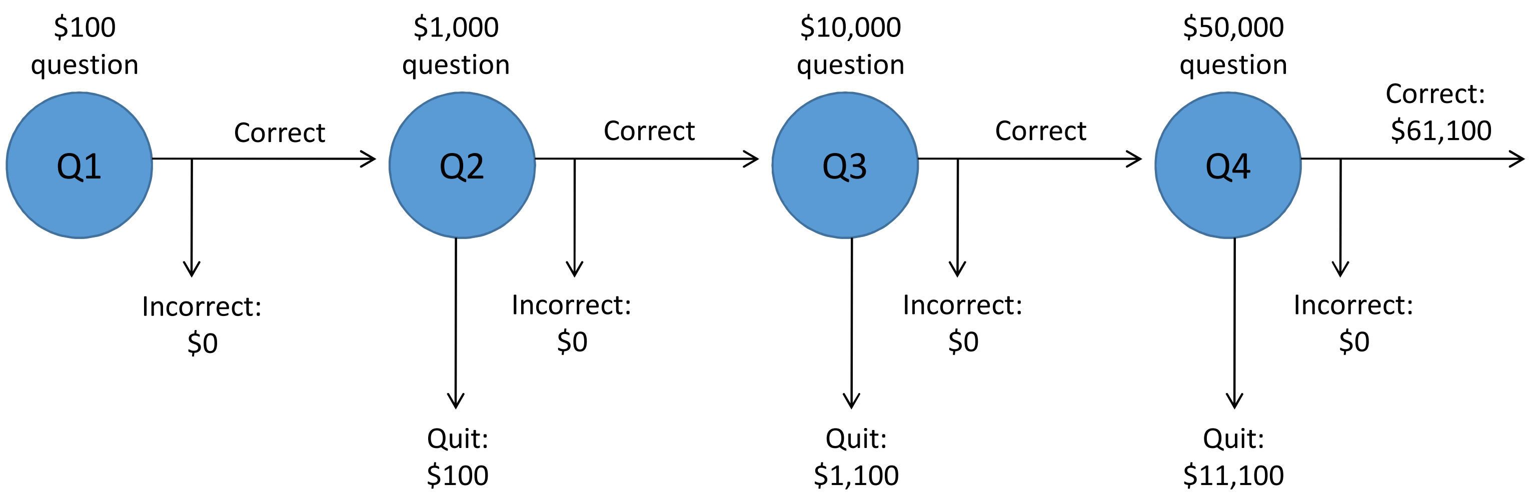
- A series of questions with increasing level of difficulty and increasing payoff
- Decision: at each step, take your earnings and quit, or go for the next question
  - If you answer wrong, you lose everything



Game show

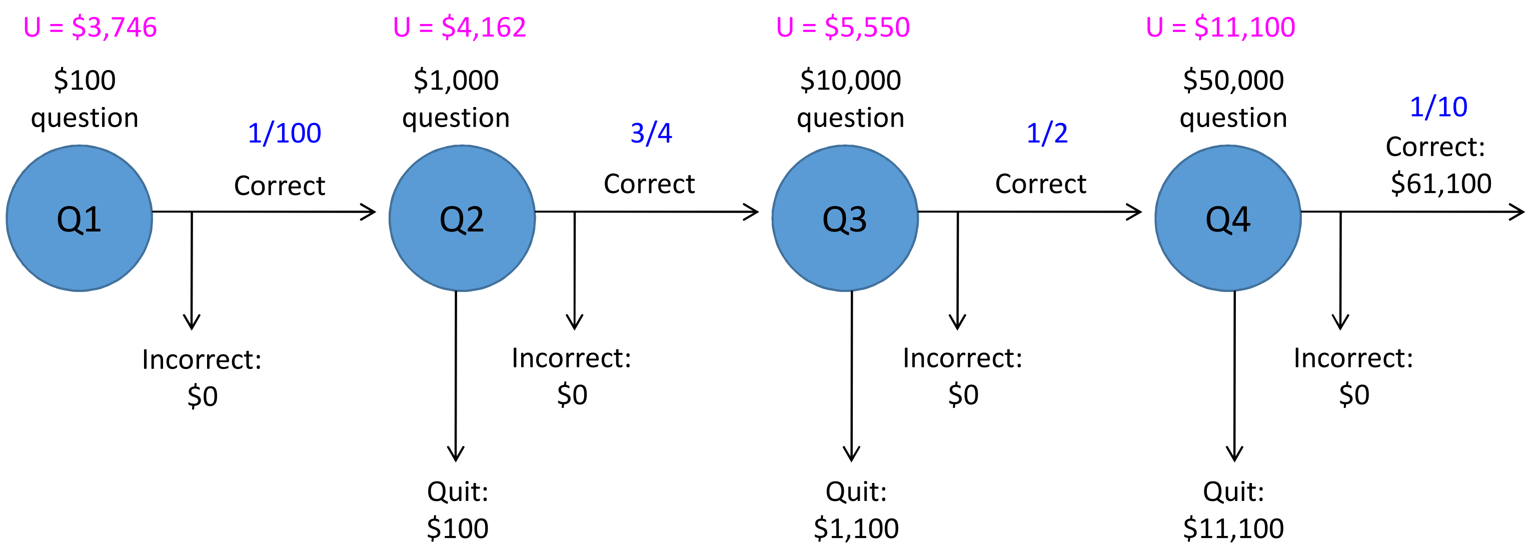
- Consider $50,000 question
  - Probability of guessing correctly: 1/10
  - Quit or go for the question?
- What is the expected payoff for continuing?

  0.1 * 61,100 + 0.9 * 0 = 6,110
- What is the optimal decision?

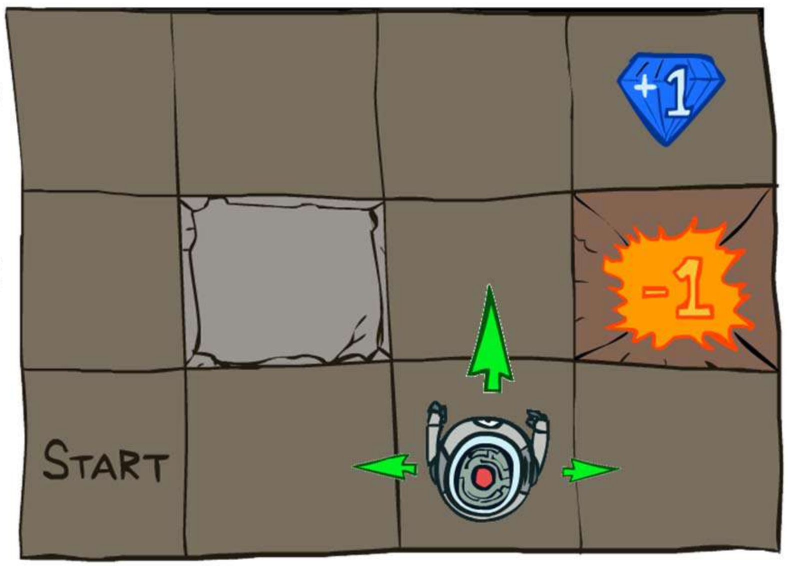


Game show

- What should we do in Q3?
  - Payoff for quitting: $1,100
  - Payoff for continuing: 0.5 * $11,100 = $5,550
- What about Q2?
  - $100 for quitting vs. $4,162 for continuing
- What about Q1?

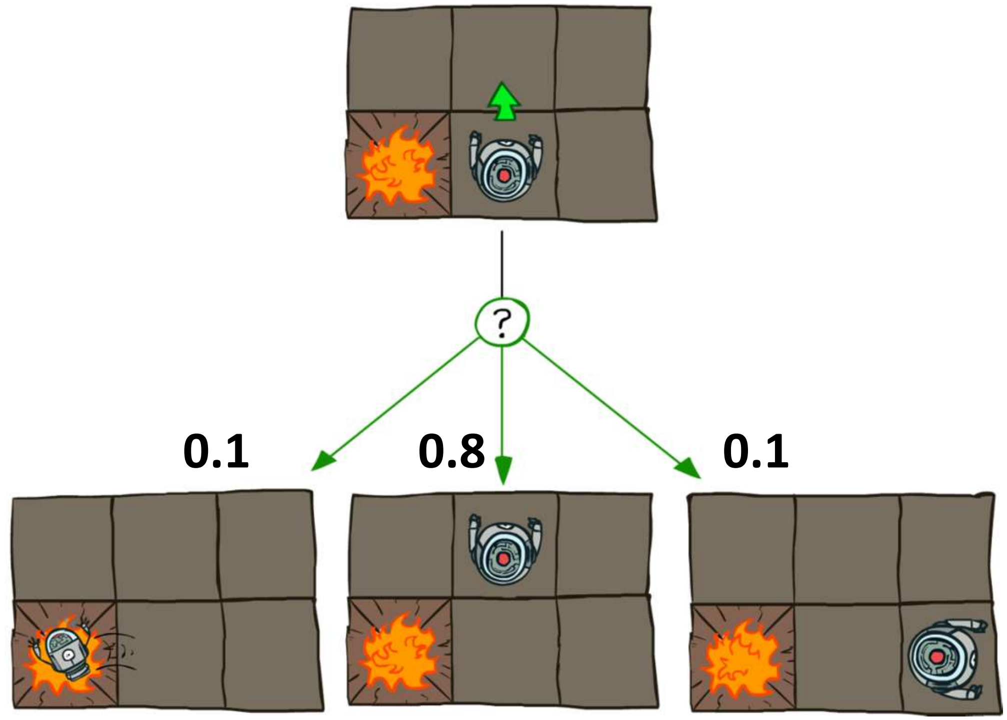


Grid world



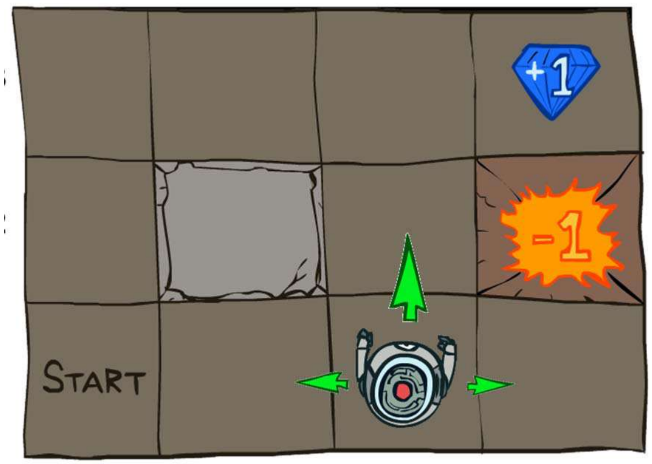
R(s) = -0.04 for every non-terminal state

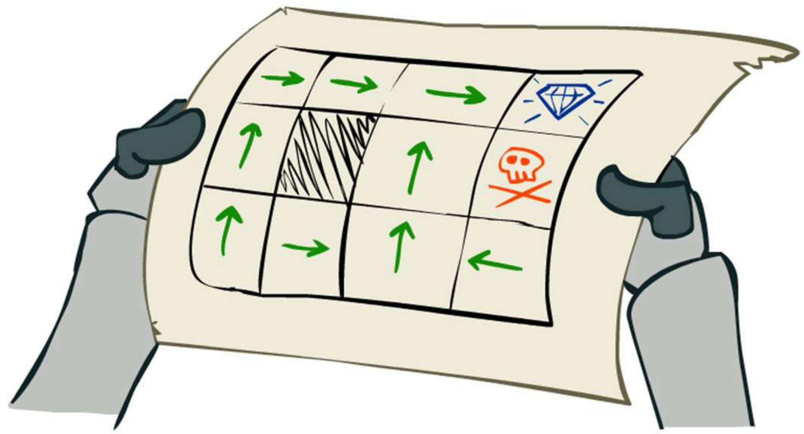
Transition model:



Source: P. Abbeel and D. Klein

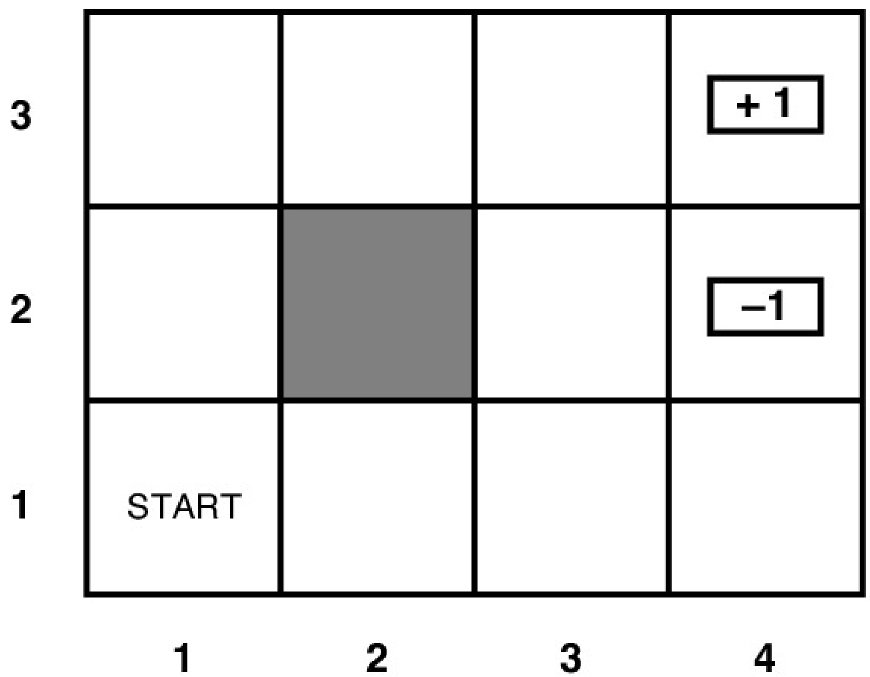
Goal: Policy



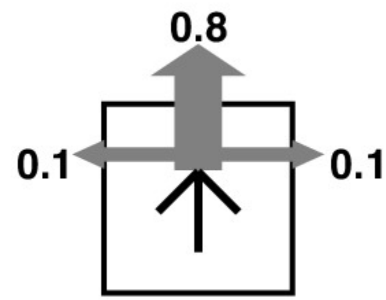


Source: P. Abbeel and D. Klein

Grid world

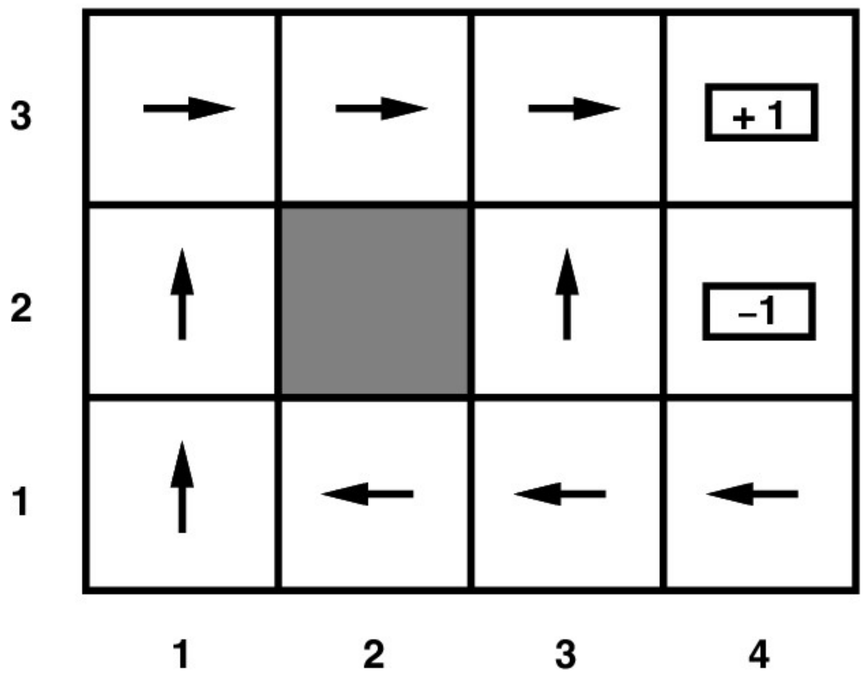


Transition model:



R(s) = -0.04 for every non-terminal state

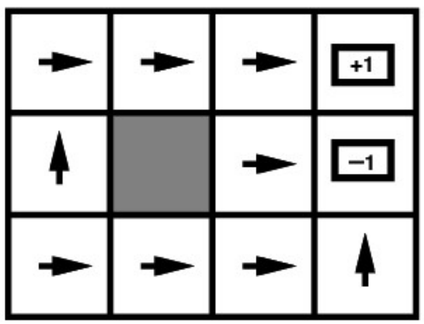
Grid world



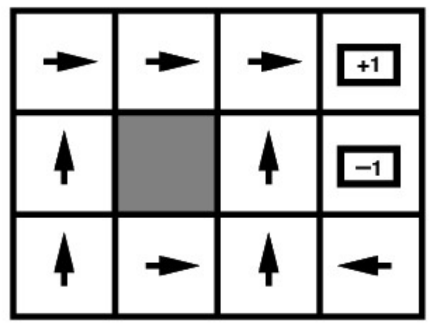
Optimal policy when R(s) = -0.04 for every non-terminal state

Grid world

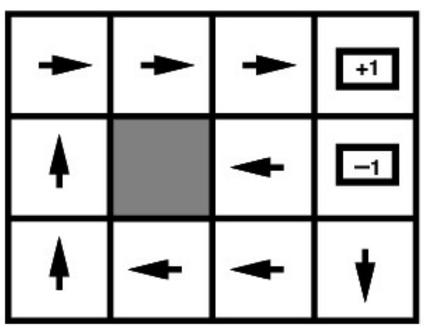
- Optimal policies for other values of R(s):



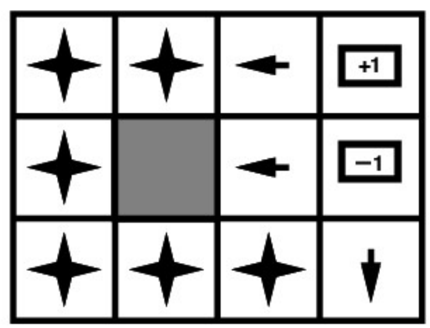
$R(s) < -1.6284$



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



$-0.0221 < R(s) < 0$



$R(s) > 0$

Solving MDPs

- MDP components:
  - **States** s
  - **Actions** a
  - **Transition model** $P(s' \mid s, a)$
  - **Reward function** $R(s)$
- The solution:
  - **Policy** $\pi(s)$: mapping from states to actions
  - How to find the optimal policy?

Maximizing expected utility

- The optimal policy π(s) should maximize the *expected utility* over all possible state sequences produced by following that policy:

$$\sum_{\substack{state\ sequences \\ starting\ from\ s_0}} P\big(sequence|s_0, a = \pi(s_0)\big)U(sequence)$$

- How to define the utility of a state sequence?
  - Sum of rewards of individual states
  - Problem: infinite state sequences

Utilities of state sequences

- Normally, we would define the utility of a state sequence as the sum of the rewards of the individual states
- **Problem:** infinite state sequences
- **Solution:** *discount* the individual state rewards by a factor γ between 0 and 1:

$$U([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

$$= \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \frac{R_{\max}}{1-\gamma} \qquad (0 < \gamma < 1)$$

  - Sooner rewards count more than later rewards
  - Makes sure the total utility stays bounded
  - Helps algorithms converge

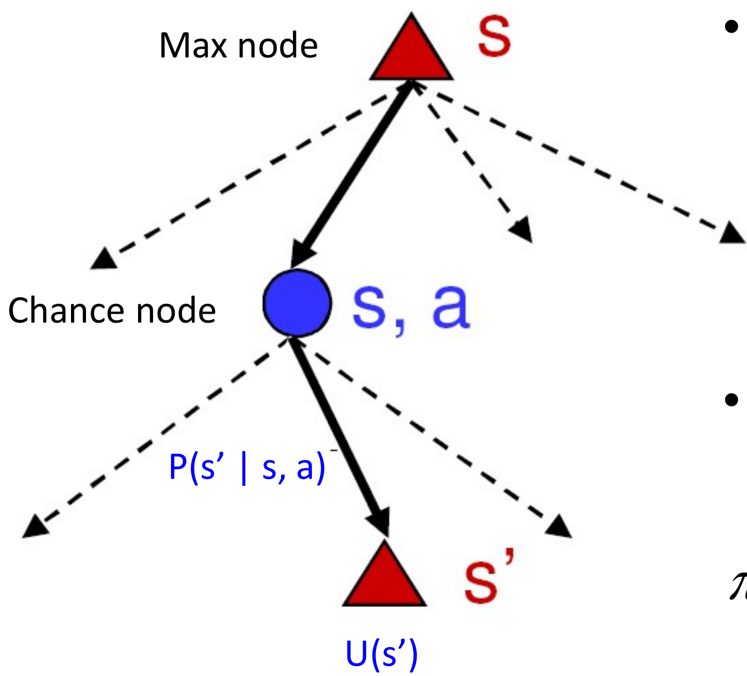
Utilities of states

- Expected utility obtained by policy π starting in state s:

$$U^{\pi}(s) = \sum_{\substack{state\ sequences \\ starting\ from\ s}} P\big(sequence|s, a = \pi(s)\big)U(sequence)$$

- The “true” utility of a state, denoted U(s), is the *best possible* expected sum of discounted rewards
  - if the agent executes the *best possible* policy starting in state s
- Reminiscent of minimax values of states…

Finding the utilities of states



- What is the expected utility of taking action **a** in state **s**?

$$\sum_{s'} P(s' \mid s, a) U(s')$$

- How do we choose the optimal action?

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

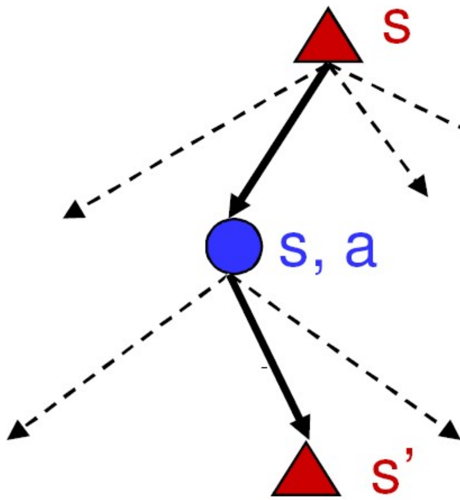
- What is the recursive expression for **U(s)** in terms of the utilities of its successor states?

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a) U(s')$$

The Bellman equation

- Recursive relationship between the utilities of successive states:

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



Receive reward R(s)

Choose optimal action a

End up here with P(s' | s, a)
Get utility U(s')
(discounted by γ)

The Bellman equation

- Recursive relationship between the utilities of successive states:

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

- For *N* states, we get *N* equations in *N* unknowns
  - Solving them solves the MDP
  - Nonlinear equations -> no closed-form solution, need to use an iterative solution method (is there a globally optimum solution?)
  - We could try to solve them through expectiminimax search, but that would run into trouble with infinite sequences
  - Instead, we solve them algebraically
  - Two methods: **value iteration** and **policy iteration**

Method 1: Value iteration

- Start out with every $U(s) = 0$
- Iterate until convergence
  - During the ith iteration, update the utility of each state according to this rule:

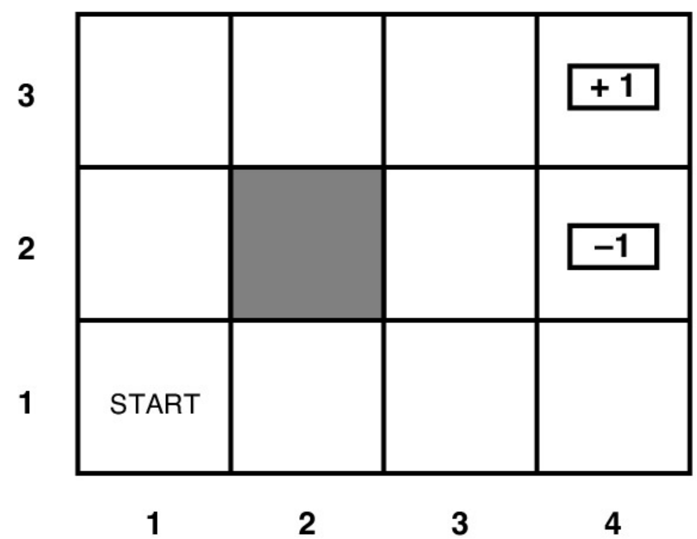
$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s')$$

- In the limit of infinitely many iterations, guaranteed to find the correct utility values
  - In practice, don't need an infinite number of iterations…

Value iteration

- What effect does the update have?

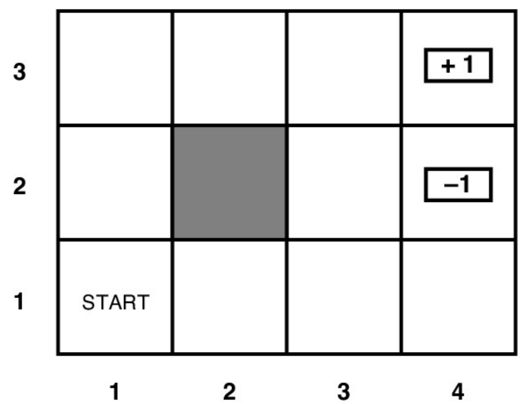
$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s')$$

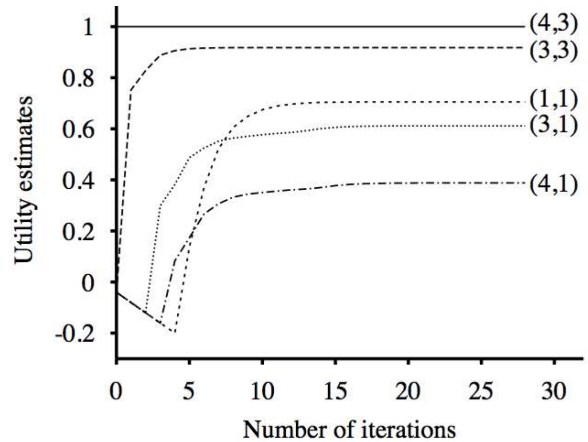


Value iteration demo

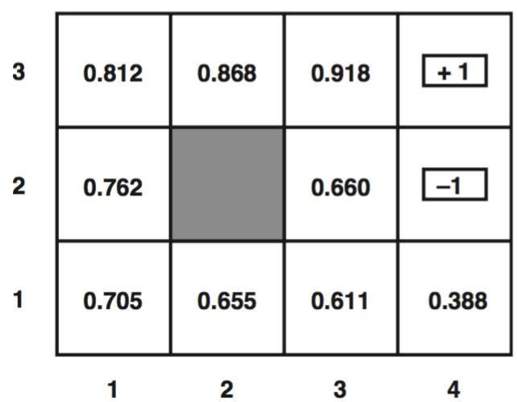
Value iteration

Input (non-terminal R=-0.04)

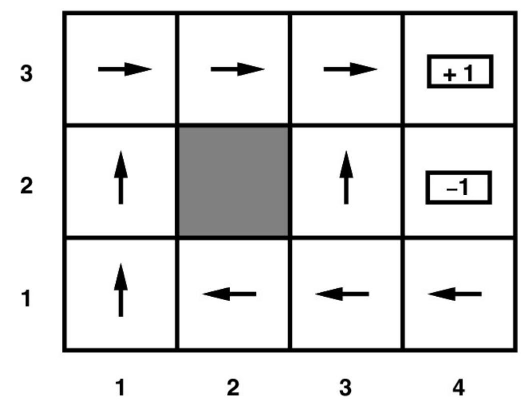




Utilities with discount factor 1



Final policy



Method 2: Policy iteration

- Start with some initial policy π_0 and alternate between the following steps:
  - **Policy evaluation:** calculate $U^{\pi_i}(s)$ for every state s
  - **Policy improvement:** calculate a new policy π_{i+1} based on the updated utilities

- Notice it's kind of like hill-climbing in the N-queens problem.
  - Policy evaluation: Find ways in which the current policy is suboptimal
  - Policy improvement: Fix those problems

- Unlike Value Iteration, this is guaranteed to converge in a finite number of steps, as long as the state space and action set are both finite.

Method 2, Step 1: Policy evaluation

- Given a fixed policy π, calculate $U^{\pi}(s)$ for every state s

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi(s)) U^{\pi}(s')$$

- $\pi(s)$ is fixed, therefore $P(s'|s, \pi(s))$ is an $s' \times s$ matrix, therefore we can solve a linear equation to get $U^{\pi}(s)$!
- Why is this "Policy Evaluation" formula so much easier to solve than the original Bellman equation?

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

Method 2, Step 2: Policy improvement

- Given $U^{\pi}(s)$ for every state s, find an improved $\pi(s)$

$$\pi^{i+1}(s) = \underset{a \in A(s)}{\arg\max} \sum_{s'} P(s' | s, a) U^{\pi_i}(s')$$

Summary

- MDP defined by states, actions, transition model, reward function
- The “solution” to an MDP is the policy: what do you do when you’re in any given state
- The Bellman equation tells the utility of any given state, and incidentally, also tells you the optimum policy. The Bellman equation is N nonlinear equations in N unknowns (the policy), therefore it can’t be solved in closed form.
- Value iteration:
  - At the beginning of the (i+1)’st iteration, each state’s value is based on looking ahead i steps in time
  - … so finding the best action = optimize based on (i+1)-step lookahead
- Policy iteration:
  - Find the utilities that result from the current policy,
  - Improve the current policy