

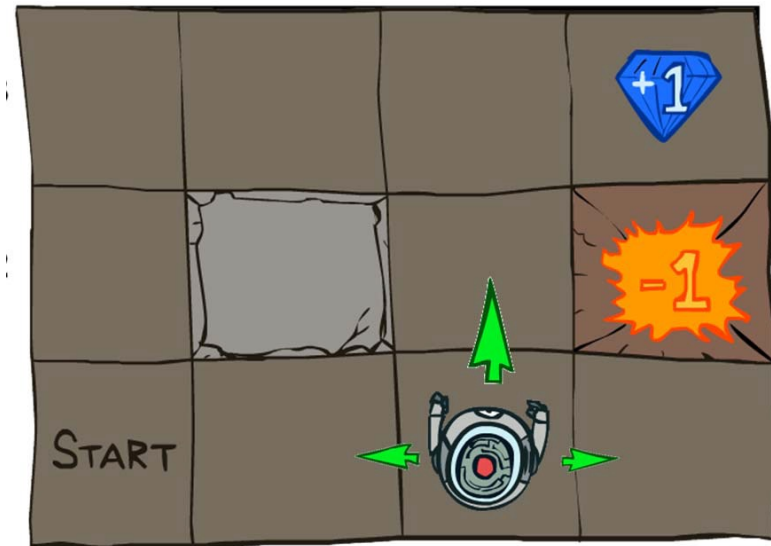
CS440/ECE448 Lecture 32: Markov Decision Processes

Mark Hasegawa-Johnson, 4/2022

Including slides by Svetlana Lazebnik, 11/2016

Including many figures by Peter Abbeel and Dan Klein, UC Berkeley
CS 188

CC-BY-4.0: You may re-use or redistribute if you cite the source.



Grid World

Invented and drawn by Peter Abbeel and Dan
Klein, UC Berkeley CS 188

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:

- Like an HMM, we have **States** s , beginning with initial state s_0
- Unlike an HMM, we also have **Actions** a
 - Each state s has actions $A(s)$ available from it
- Unlike an HMM, the **Transition model** $P(s' | s, a)$ depends on both the state you're in, and the action you perform.
 - *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

Markov Decision Processes

What is our purpose? In an HMM, we're trying to understand the world. In an MDP, we're trying to influence the world. Why?

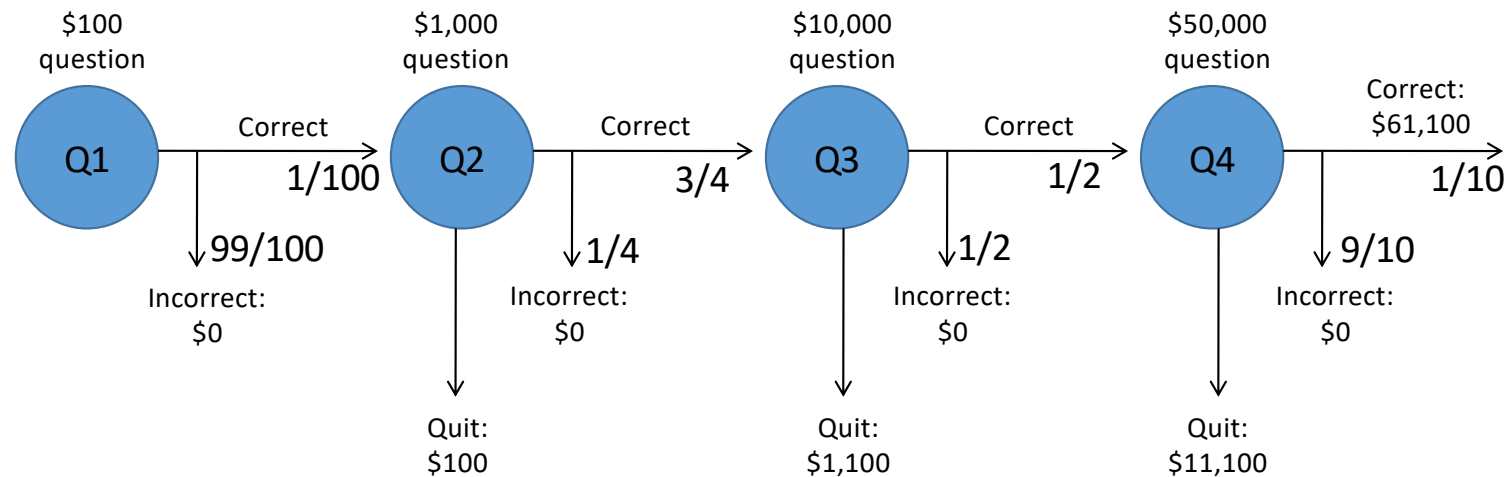
- **Reward function $R(s)$.** As in a two-player game, we assume that some states are better than others. Our goal is to influence the world so that we end up in a better state.
- **The solution: A Policy, $\pi(s) \in A(s)$:** the action that the agent takes in any given state. As in a two-player game, we are trying to solve for a policy that is optimum, in the sense that it maximizes our expected reward.

Outline

1. What we need to know: transition model, reward function.
2. The closed-form optimum solution: The Bellman equation.
3. Iterative solutions that approximate the Bellman equation in polynomial time
 1. Value Iteration
 2. Policy Iteration

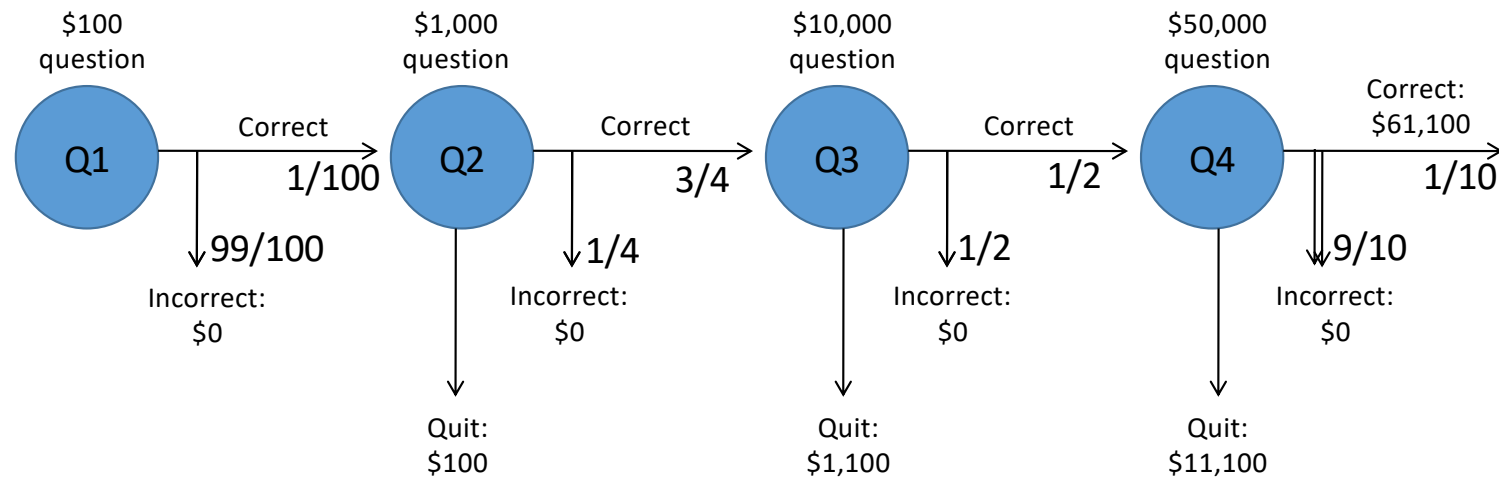
Example: Game show

- 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



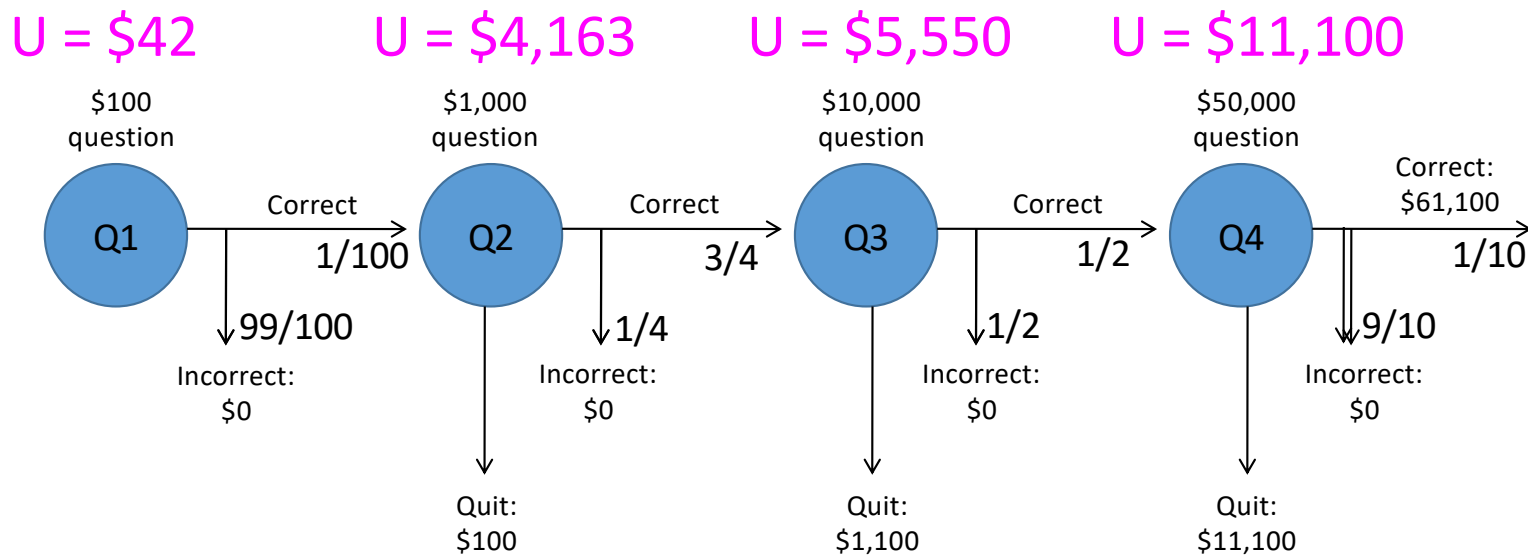
Example: Game show

- 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?



Example: Game show

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



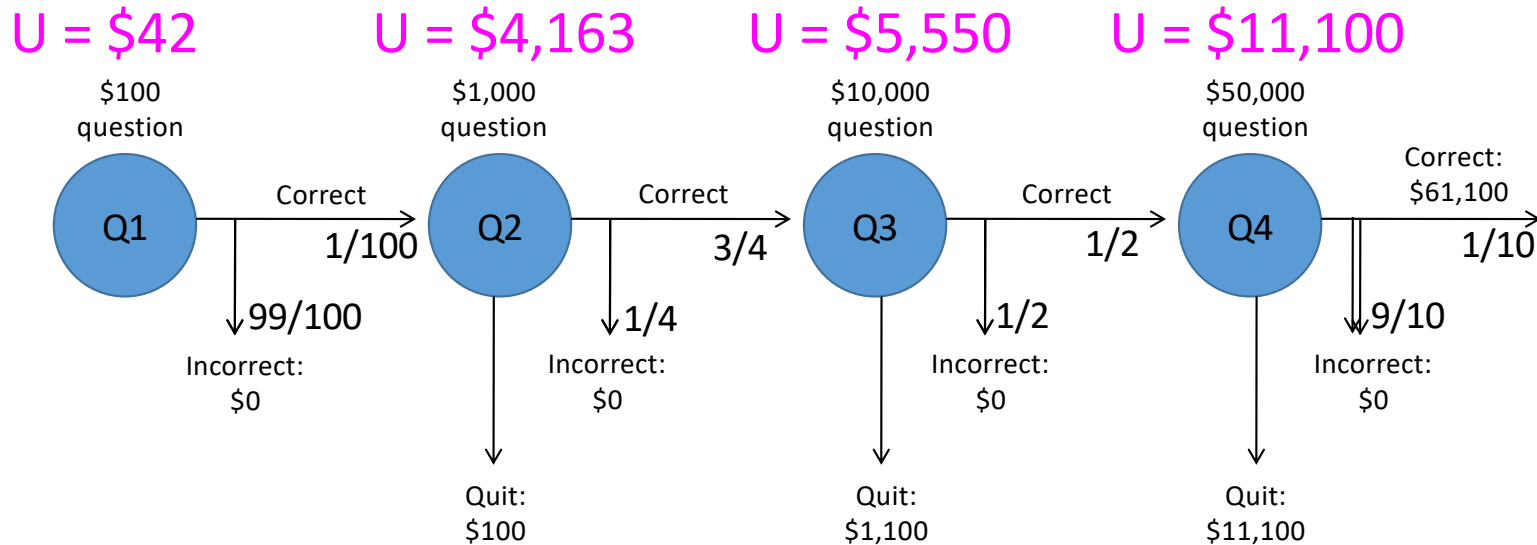
Example: Game show

Let's talk about two different ways of calculating our winnings:

- $R(s)$ = the reward that we receive for reaching state s
- $U(s)$ = the total expected utility or value of state s

$U(s)$ is the **expected sum** of all current and future rewards, given that we have reached state s , assuming we know the **best actions to perform** from here on out.

- Calculating $U(s)$ is kind of like expectiminimax. But first, let's look at another example...



Solving MDPs

- MDP components:
 - **States** s
 - **Actions** a
 - **Transition model** $P(s' | s, a)$
 - **Reward function** $R(s)$
- The solution:
 - **Policy** $\pi(s)$: mapping from states to actions
 - Game show example
 - $\pi(Q1) =$ take another question
 - $\pi(Q2) =$ take another question
 - $\pi(Q3) =$ take another question
 - $\pi(Q4) =$ quit while you're ahead
 - How to find the optimal policy in an arbitrary environment?

Outline

1. What we need to know: transition model, reward function.
2. The closed-form optimum solution: The Bellman equation.
3. Iterative solutions that approximate the Bellman equation in polynomial time
 1. Value Iteration
 2. Policy Iteration

Maximizing expected utility

- The optimal policy $\pi(s)$ should maximize the *expected sum of all future rewards* that we can achieve if we start from state s and, for the remainder of time, *always perform the optimal action in any future state*
- This quantity is called the “utility” of state s , $U(s)$:
 - Maximum, over all possible policies, of the...
 - Expected value of the...
 - Sum of all future rewards received from now until forever.
- Problem #1: if $R(s) \geq 0$ for every s , and “forever” is “forever,” then $U(s)$ is infinite for many different policies, so we can’t decide which policy is optimal

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:

$$U([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$
$$= \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \frac{R_{\max}}{1-\gamma} \quad (0 < \gamma < 1)$$

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

Utilities of states

- Problem #2: how do we compare the *expected value* of the reward sequence, if we don't know (yet) which actions we should perform?
- Answer: define a policy-dependent utility, $U^\pi(s)$:

$$U^\pi(s_0) = \sum_{s_1, s_2, \dots} P(s_1, s_2, \dots | a_t = \pi(s_t)) U(s_0, s_1, s_2, \dots)$$

- Then the optimal utility is “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...

Optimal utility

- Problem #3: how do we choose the ***optimum policy***?
- Answer: the optimum policy is the one that maximizes the utility (the expected sum of all future rewards) :

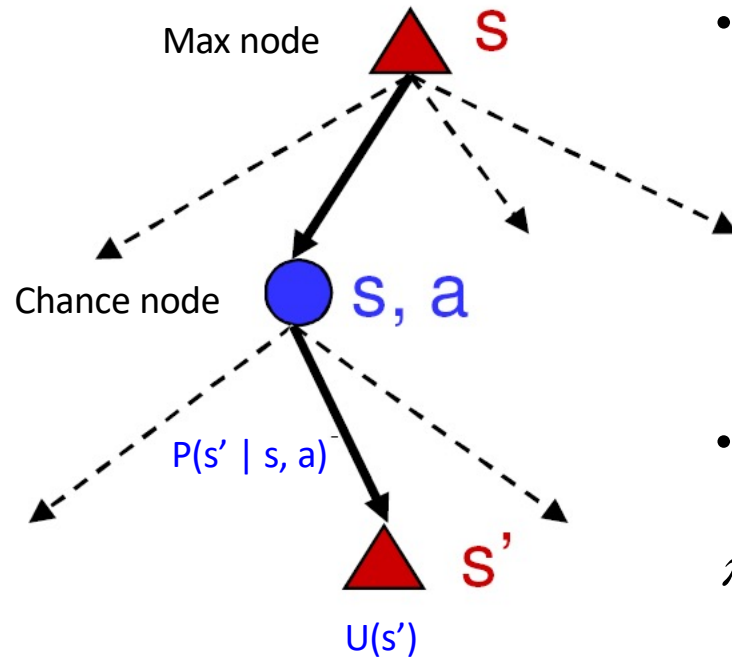
$$U(s_o) = \max_{\pi} U^{\pi}(s_o)$$

Thus utility is the:

- maximum, over all possible policies, of the
- expected value, over all possible state sequences, of the
- sum of all future rewards.

It's like the “expectimax” part of expectiminimax.

Finding the utilities of states



- If state s' has utility $U(s')$, then what is the expected utility of taking action a in state s ?

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

- How do we choose the optimal action?

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

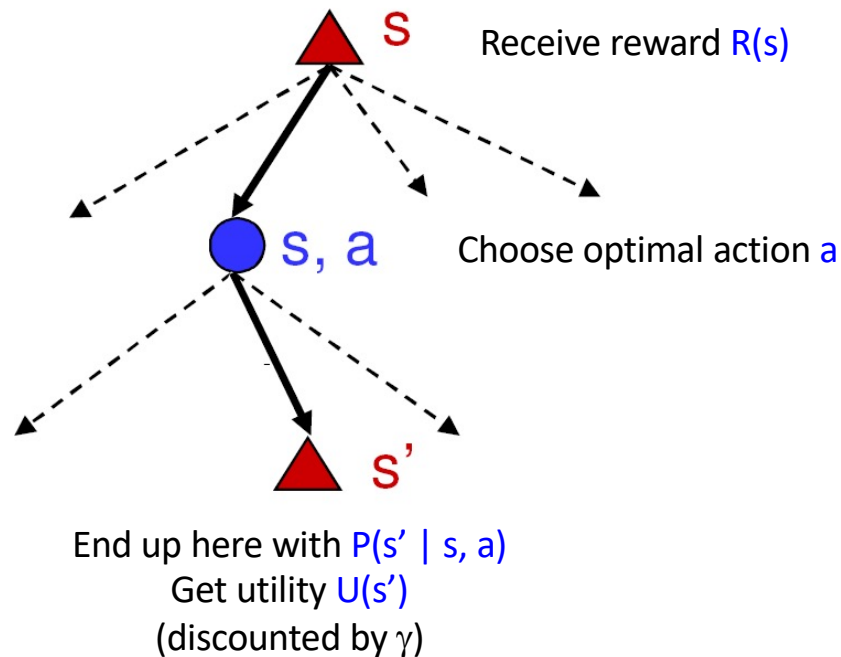
- What is the recursive expression for $U(s)$ in terms of the utilities of its successor states?

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

The Bellman equation

- Recursive relationship between the utilities of successive states:

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



The Bellman equation

- Recursive relationship between the utilities of successive states:

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

- For N states, we get N **nonlinear** equations in N unknowns
 - Known quantities: $P(s'|s, a)$, $R(s)$, and γ . Unknowns: $U(s)$.
 - Solving these N equations solves the MDP.
 - Nonlinear \rightarrow no closed-form solution.
 - If it weren't for the "max," this would be N linear equations in N unknowns. We could solve it by just inverting an $N \times N$ matrix.
 - The "max" means that there is no closed-form solution. Need to use an iterative solution method, which might not converge to the globally optimum solution.
 - Two solution methods: **value iteration** and **policy iteration**

Outline

1. What we need to know: transition model, reward function.
2. The closed-form optimum solution: The Bellman equation.
3. Iterative solutions that approximate the Bellman equation in polynomial time
 1. Value Iteration
 2. Policy Iteration

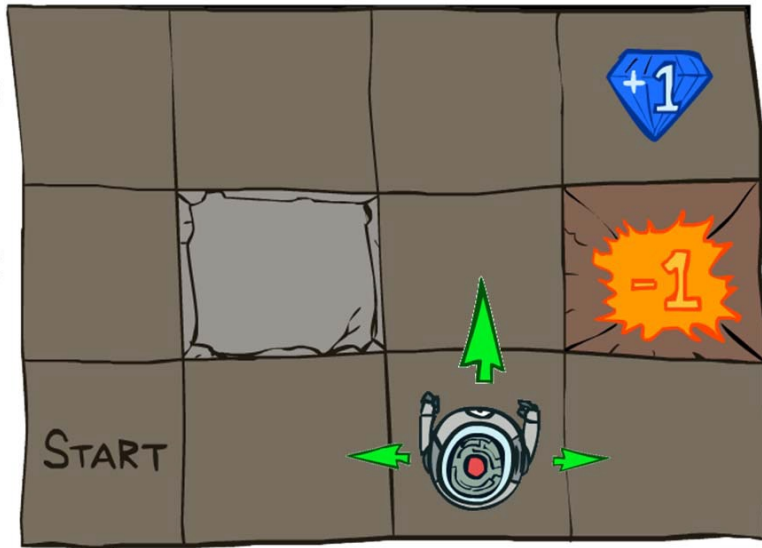
Method 1: Value iteration

- Start out with iteration $i = 0$, every $U_i(s) = 0$
- Iterate until convergence
 - During the i^{th} iteration, update the utility of each state according to this rule:

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

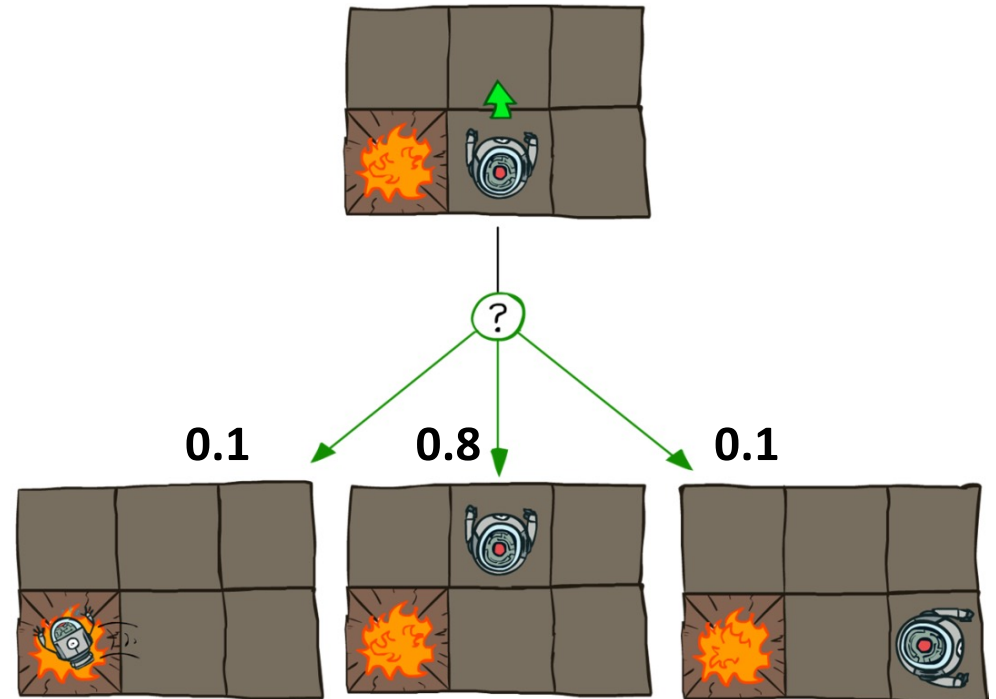
- In the limit of infinitely many iterations, guaranteed to find the correct utility values.
 - Error decreases exponentially, so in practice, don't need an infinite number of iterations...

Example: Grid world



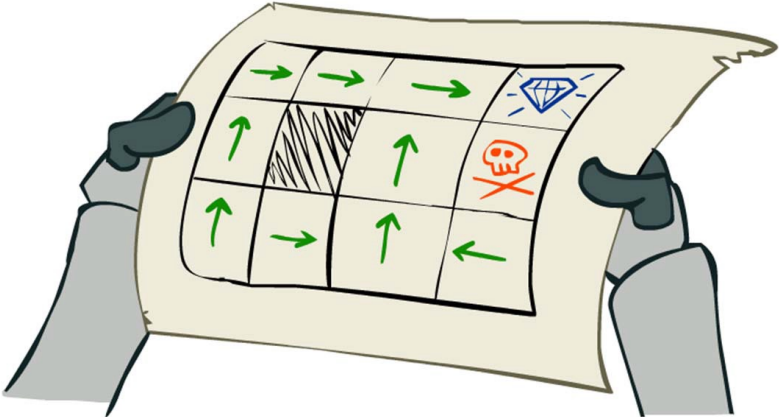
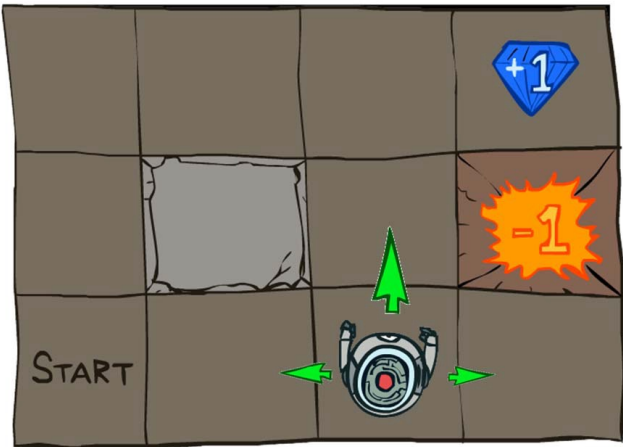
$R(s) = -0.04$ for every non-terminal state

Transition model:



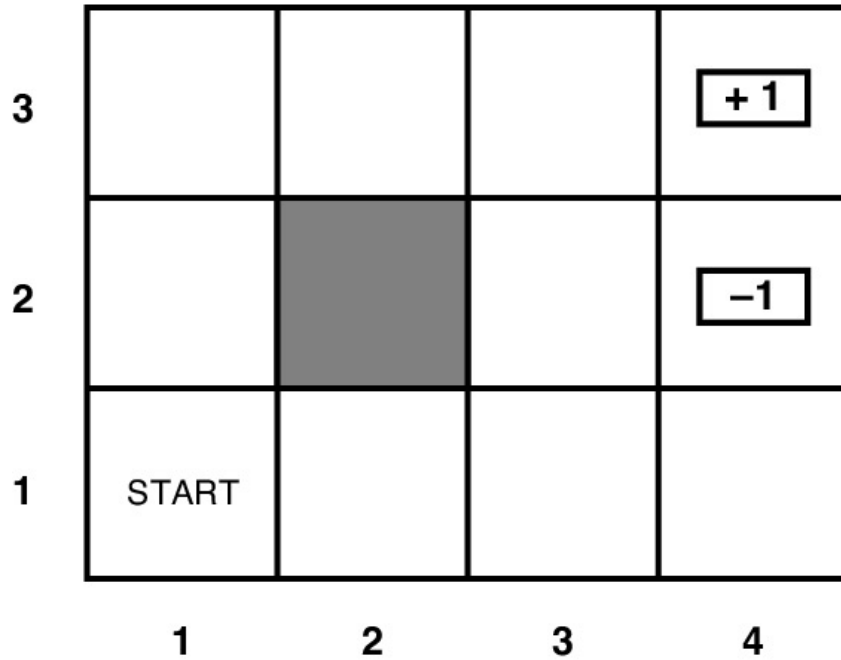
Source: P. Abbeel and D. Klein

Goal: Policy

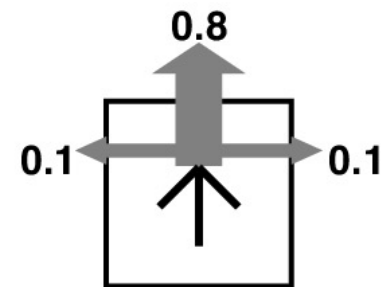


Source: P. Abbeel and D. Klein

Grid world



Transition model:





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



Value Iteration: Iteration 1

$$U_1(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a) U_0(s')$$



$U_1(s)$

-0.04	-0.04	-0.04	
-0.04		-0.04	
-0.04	-0.04	-0.04	-0.04

$R(s)$


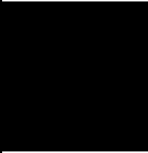

-0.04	-0.04	-0.04	
-0.04		-0.04	
-0.04	-0.04	-0.04	-0.04

$U_0(s)$

0	0	0	
0		0	
0	0	0	0

Value Iteration: Iteration 2

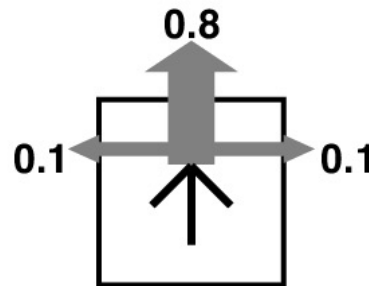
$U_1(s)$

-0.04	-0.04	-0.04	
-0.04		-0.04	
-0.04	-0.04	-0.04	-0.04




Value Iteration: Iteration 2

$$U_2(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a) U_1(s')$$

Transition model:



$U_1(s)$



-0.04	-0.04	-0.04	
-0.04		-0.04	
-0.04	-0.04	-0.04	-0.04

$$P(s'|s, \text{up}) = \begin{cases} 0.8 & s' = \text{up from } s \text{ (if no wall)} \\ 0.1 & s' = \text{left from } s \text{ (if no wall)} \\ 0.1 & s' = \text{right from } s \text{ (if no wall)} \end{cases}$$



Value Iteration: Iteration 2

$$U_2(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a) U_1(s')$$



$U_2(s) (\gamma = 1)$

-0.08	-0.08	+0.75	
-0.08		-0.08	
-0.08	-0.08	-0.08	-0.08



$$\sum_{s'} P(s'|s, \text{down}) U_1(s')$$

s'	-0.04	+0.06	
-0.04		-0.14	
-0.04	-0.04	-0.04	-0.04



$$\sum_{s'} P(s'|s, \text{up}) U_1(s')$$

s'	-0.04	+0.06	
-0.04		-0.14	
-0.04	-0.04	-0.04	-0.81



$U_1(s)$

-0.04	-0.04	-0.04	
-0.04		-0.04	
-0.04	-0.04	-0.04	-0.04

$$\sum_{s'} P(s'|s, \text{left}) U_1(s')$$

s'	-0.04	-0.04	
-0.04		-0.04	
-0.04	-0.04	-0.04	-0.14

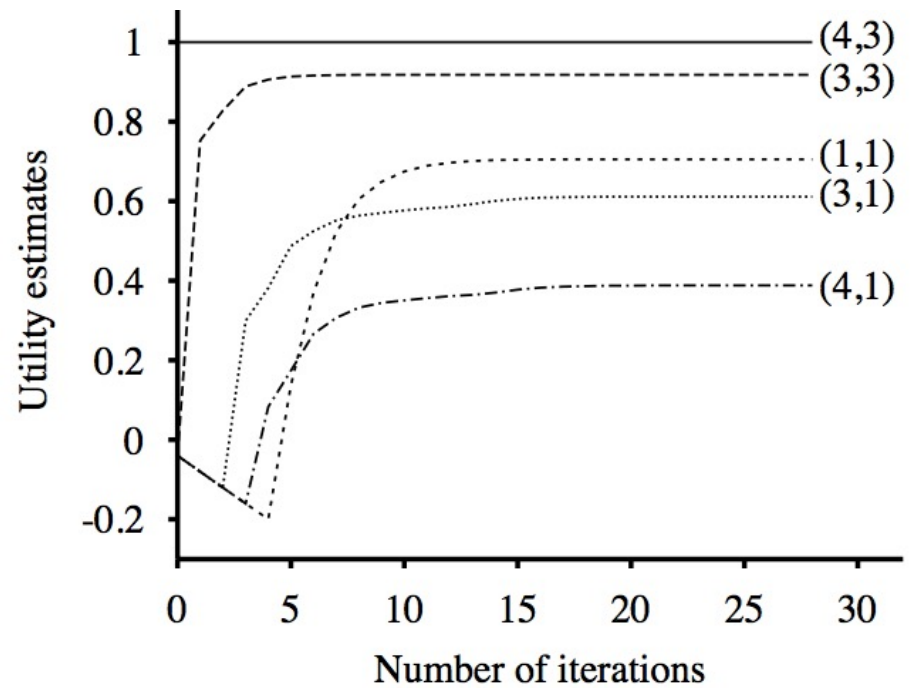
$$\sum_{s'} P(s'|s, \text{right}) U_1(s')$$

s'	-0.04	+0.79	
-0.04		-0.81	
-0.04	-0.04	-0.04	-0.14

Value iteration

Optimal utilities with discount factor 1
(Result of value iteration)

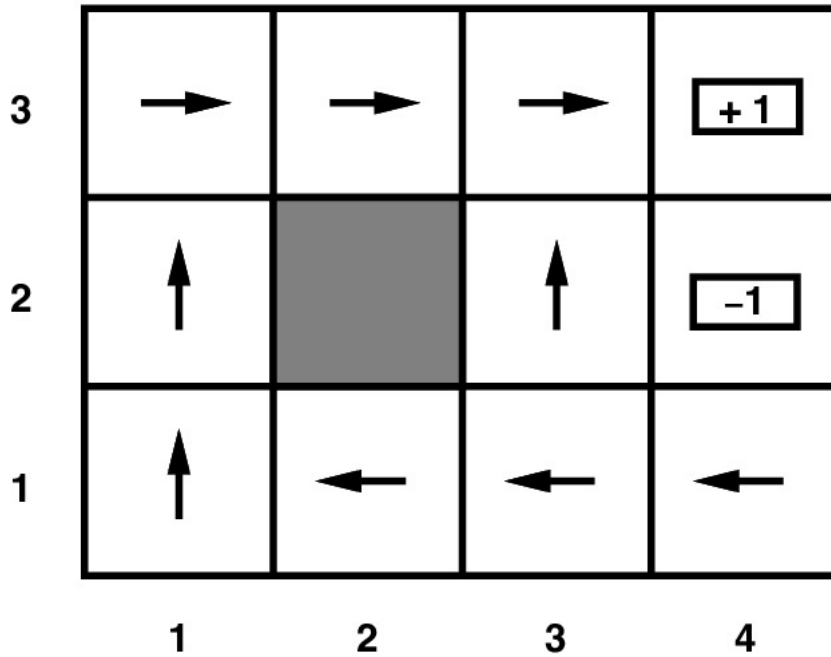
3	0.812	0.868	0.918	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388
	1	2	3	4



Final policy

3	→	→	→	+1
2	↑		↑	-1
1	↑	←	←	←
	1	2	3	4

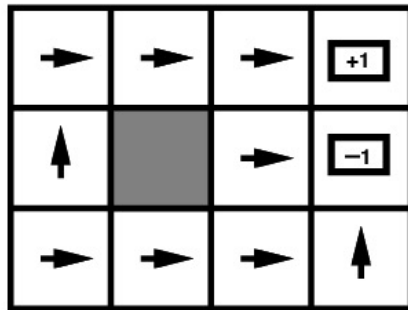
Grid world



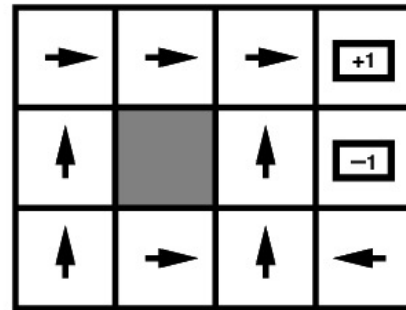
Optimal policy when
 $R(s) = -0.04$ for every
non-terminal state

Grid world

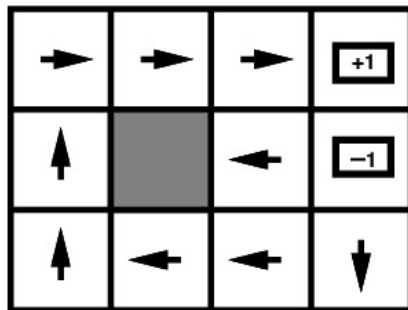
- Optimal policies for other values of $R(s)$:



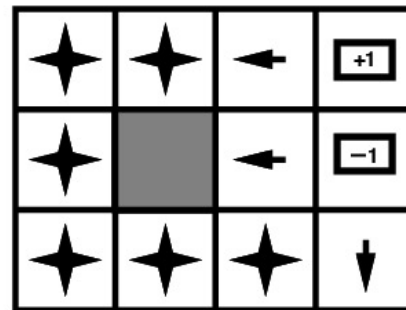
$$R(s) < -1.6284$$



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



$$-0.0221 < R(s) < 0$$



$$R(s) > 0$$

Outline

1. What we need to know: transition model, reward function.
2. The closed-form optimum solution: The Bellman equation.
3. Iterative solutions that approximate the Bellman equation in polynomial time
 1. Value Iteration
 2. Policy Iteration

Method 2: Policy Iteration

- Start with some initial policy π_0 and alternate between the following steps:
 - **Policy Evaluation:** calculate the utility of every state under the assumption that the given policy is fixed and unchanging, i.e, $U^\pi(s)$
 - **Policy Improvement:** calculate a new policy π_{i+1} based on the updated utilities.
- Notice it's kind of like gradient descent in neural networks:
 - 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.

Step 1: Policy Evaluation

Policy Evaluation: Given a fixed policy π , calculate the policy-dependent utility, $U^\pi(s)$, for every state s

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

Notice how this differs from the Bellman equation:


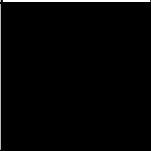

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a) U(s')$$

The difference is that policy evaluation is N_linear_ equations in N unknowns, whereas the Bellman equation is N_nonlinear_ equations in N unknowns (N=# states).




Policy Evaluation: Iteration 1

Policy Evaluation:
$$U^{\pi^0}(s) = R(s) + \gamma \sum_{s'} P(s'|s, a) U^{\pi^0}(s')$$

$U^{\pi^0}(s)$

+0.50	+0.69	+0.74	
-0.65		-0.90	
-1.40	-1.44	-1.39	-1.40

$\pi^0(s)$

→	→	→	
→		→	
→	→	→	→

Step 2: Policy Improvement

- **Policy Evaluation:** Given a fixed policy π , calculate the policy-dependent utility, $U^\pi(s)$, for every state s

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

- **Policy Improvement:** Given $U^\pi(s)$ for every state s , find an improved $\pi(s)$



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

Policy Improvement: Iteration 1



Policy Evaluation: $U^{\pi^0}(s) = R(s) + \gamma \sum_{s'} P(s'|s, a) U^{\pi^0}(s')$

Policy Improvement: $\pi^1(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s, a) U^{\pi^0}(s')$



$\pi^1(s)$

→	→	→	
↑		↑	
↑	→	↑	↑

$U^{\pi^0}(s)$

+0.50	+0.69	+0.74	
-0.65		-0.90	
-1.40	-1.44	-1.39	-1.40

$\pi^0(s)$

→	→	→	
→		→	
→	→	→	→

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