

# Markov Decision Processes

# Decision Theory

- **Decision Theory** deals with choosing among actions based on the desirability of their immediate outcomes.
- **Probability Theory**: Reasoning about the likelihood of situations or events
- **Utility Theory**: Reasoning about preferences between possible outcomes
- **Decision Theory = Probability Theory + Utility Theory**
  - An agent is “**rational**” if it chooses the action that yields the highest expected utility, averaged over all possible outcomes of the action.
  - If an agent acts so as to maximize a *utility function* that correctly reflects the performance measure, then the agent will achieve the highest possible performance score (averaged over all the possible environments).

# Maximum Expected Utility (MEU)

- Actions are *stochastic*, so the result of an action can be any of several states.
- A **transition model** describes the probabilities of the possible successor states given an action  $a$  and evidence  $e$  about the world state.
- The **probability of outcome  $s'$** , given evidence observations  $e$  and the event that action  $a$  is executed, is written as

$$P(\text{RESULT}(a)=s' \mid a, e)$$

- The *agent's preferences* are captured by a **utility function**,  $U(s)$ , which assigns a single number to express the desirability of a state.

# Maximum Expected Utility (MEU)

- The **expected utility of an action** given the evidence  $e$ ,  $EU(a|e)$ , is the average utility value of the outcomes, weighted by the probability that the outcome occurs:

$$EU(a|e) = \sum_{s'} P(\text{RESULT}(a) = s' | a, e) U(s')$$

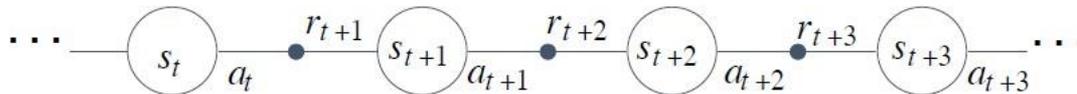
- The principle of **maximum expected utility (MEU)** says that a *rational agent* should choose the action that maximizes the agent's expected utility:

$$action = \underset{a}{\operatorname{argmax}} EU(a | e)$$

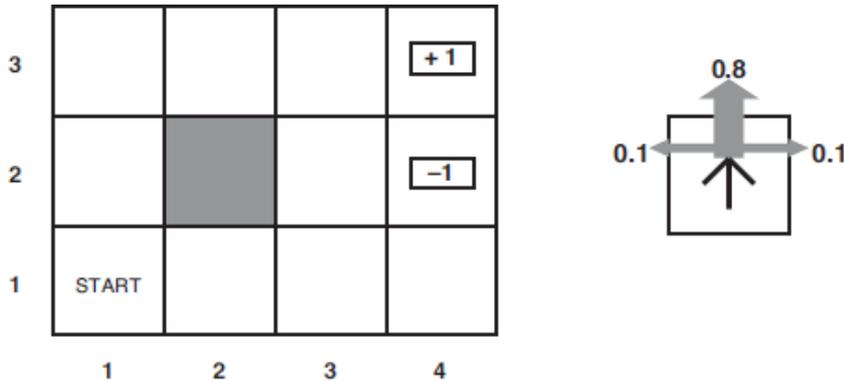
- The MEU principle formalizes the general notion that the agent should “do the right thing,” but it does not solve the whole problem.
  - Computing  $P(\text{RESULT}(a) | a, e)$  requires a complete causal model of the world
  - Computing the outcome utilities  $U(s')$  often requires searching or planning, because an agent may not know how good a state is until it knows where it can get to from that state.
  - So, decision theory only provides a useful framework.

# Sequential Decision Problems

- In sequential decision problems:
  - The agent moves from state to state.
  - At each time step, the agent chooses an action.
  - Actions move the agent to a successor state.
  - States have utilities.
  - A rational agent acts so as to maximize the expected utility, i.e., maximize the reward.
- System advances in discrete time steps  $t = 0, 1, 2, \dots$ 
  - Agent observes the state  $s_t$  at time  $t$ :  $s_t \in \mathcal{S}$  (fully observable environment)
  - Agent takes an action at time  $t$ :  $a_t \in \text{Actions}(s_t)$
  - Resulting in some next state at time  $t+1$ :  $s_{t+1} \in \mathcal{S}$
  - Acquiring some immediate “reward”  $r_{t+1} = R(s_{t+1})$



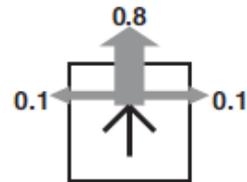
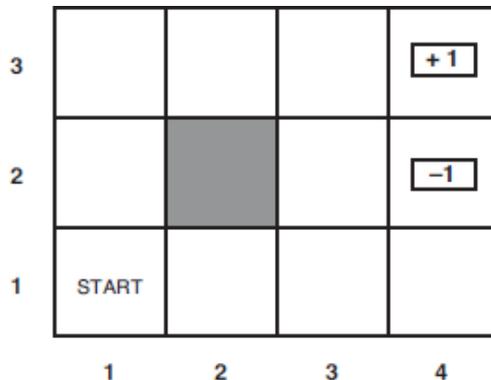
# Sequential Decision Problems: Example



- A simple 4×3 environment that presents the agent with a sequential decision problem.
- The transition model of the environment: the “intended” outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement.
- The two terminal states have reward +1 and −1, respectively, and all other states have a reward of −0.04.

# Sequential Decision Problems: Example

- If the environment were *deterministic*, a solution would be easy: [Up, Up, Right, Right, Right].
- Our environment is *stochastic*, the sequence [Up, Up, Right, Right, Right ] goes up around the barrier and reaches the goal state at (4,3) with probability  $0.8^5 = 0.32768$ .
  - There is also a small chance of accidentally reaching the goal by going the other way around with probability  $0.1^4 \times 0.8$ , for a grand total of 0.32776.



# Markov Decision Process

- A sequential decision problem for a *fully observable, stochastic environment with a Markovian transition model* and *additive rewards* is called a **Markov Decision Process (MDP)**.
- **An MDP is defined by:**
  - A **set of states**  $s \in S$  (initial state  $s_0$ )
  - A **set of actions**  $a \in A$  in each state
  - A **transition function**  $T(s, a, s')$ 
    - Probability that  $a$  from  $s$  leads to  $s'$ , i.e.,  $P(s' | s, a)$
    - Also called the model or the dynamics
  - A **reward function**  $R(s, a, s')$ 
    - Sometimes just  $R(s)$  or  $R(s')$
  - A **start state**  $s_0$
  - Maybe a **terminal state**

# Markov Decision Process:

*What is Markov about MDPs?*

- “Markov” generally means that given the present state, the future and the past are independent.
- For Markov decision processes, “Markov” means action outcomes depend only on the current state.

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \dots, S_0 = s_0)$$

=

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

- This is just like search, where the successor function could only depend on the current state (not the history)

# Policy

- In *deterministic single-agent search problems*, we want an optimal plan, or sequence of actions, from start to a goal.
  - Any fixed action sequence won't solve the problem in MDPs, because the agent might end up in a state other than the goal.
- For MDPs, we want an **optimal policy  $\pi^*: \mathbf{S} \rightarrow \mathbf{A}$**  as a solution.
  - A **policy  $\pi$**  gives an action for each state
    - Policy dictates what action to take in each possible state
    - One-to-one mapping from states to actions
    - A policy is denoted by  $\pi$ , and the action recommended at state  $s$  is  $\pi(s)$
  - An **optimal policy** is one that maximizes *expected utility* if followed
  - An explicit policy defines a reflex agent

# Markov Decision Process: Utility

- Objective in MDPs is to maximize **total reward** over an episode: **Total Reward = Utility**

**Additive Rewards:** The utility of a state sequence is

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$$

**Discounted Rewards:** The utility of a state sequence is

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

where the **discount factor**  $\gamma$  is a number between 0 and 1.

- The discount factor describes the preference of an agent for current rewards over future rewards.
- When  $\gamma$  is close to 0, rewards in the distant future are viewed as insignificant.
- When  $\gamma$  is 1, discounted rewards are exactly equivalent to additive rewards

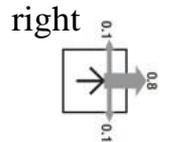
# Markov Decision Process: Utility

- The utilities of the states in the 4×3 world, calculated with  $\gamma = 1$  and  $R(s) = -0.04$  for nonterminal states.
  - These values are obtained solving utility equations.

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

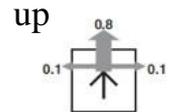
- $$U(s_{3,3}) = R(s_{3,3}) + 0.8*U(s_{4,3}) + 0.1*U(s_{3,2}) + 0.1*U(s_{3,3})$$

$$= -0.04 + 0.8*1 + 0.1*0.660 + 0.1*0.918 = 0.9178$$



- $$U(s_{3,2}) = R(s_{3,2}) + 0.8*U(s_{3,3}) + 0.1*U(s_{4,2}) + 0.1*U(s_{3,2})$$

$$= -0.04 + 0.8*0.918 + 0.1*-1 + 0.1*0.660 = 0.6604$$



# Markov Decision Process: Optimal Policies

- The **optimal policy** is independent of the starting state using discounted utilities with infinite horizons.
  - Thus, the **true utility of a state** is just  $U^{\pi^*}(s)$  that is, the expected sum of discounted rewards if the agent executes an **optimal policy**.
  - We write this as  $U(s)$ .
  - Note that,  $R(s)$  is the “short term” reward for being in  $s$ , whereas  $U(s)$  is the “long term” total reward from  $s$  onward.
- **Optimal policy** is one which maximizes the expected utility of all states.
  - The utility function  $U(s)$  allows the agent to select actions by using the principle of maximum expected utility from that is, choose the action that maximizes the expected utility of the subsequent state.

$$\pi_s^* = \operatorname{argmax}_{a \in \text{Actions}(s)} \sum_{s'} P(s' | s, a) U(s')$$

# Markov Decision Process: Optimal Policies

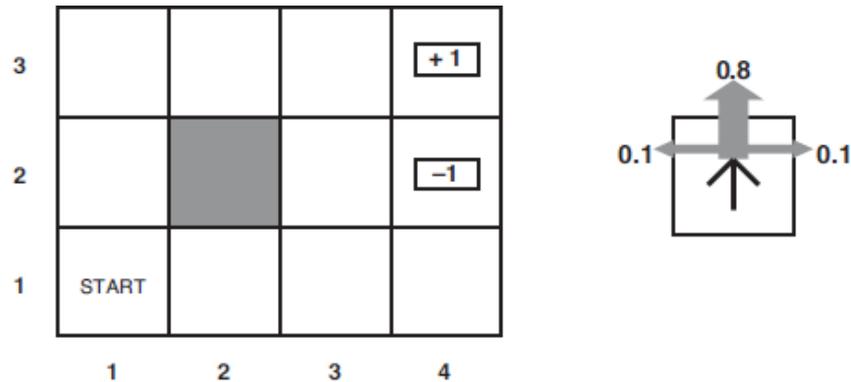
- A **policy**  $\pi(\mathbf{s})$  maps states to actions and fully defines an agent's behavior.
- The *utility of a given state sequence* is the sum of discounted rewards obtained during the sequence, we can compare policies by comparing the expected utilities obtained when executing them.
- The **expected utility** of a state under policy  $\pi$  is the expected total discounted reward:

$$U^{\pi}(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(S_t) \right] \quad \text{where } S_t \text{ is the state } s.$$

- The **optimal policy**  $\pi_s^*$  **starting from**  $\mathbf{s}$  is the one which maximizes expected future utility:

$$\pi_s^* = \operatorname{argmax}_{\pi} U^{\pi}(s)$$

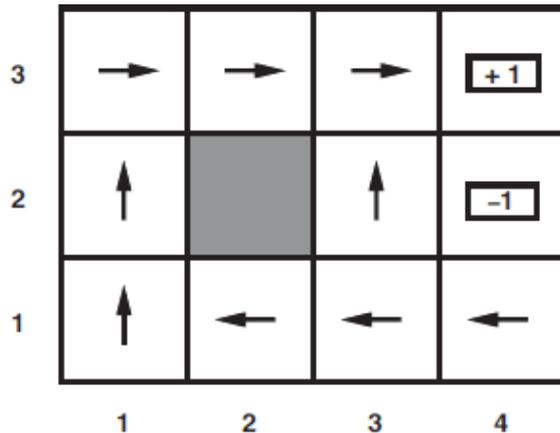
# Optimal Policies: Example



- Optimal policy depends on reward function  $R$ .
- We can get different policies for different reward functions.
- The balance of risk and reward changes depending on the value of  $R(s)$  for the nonterminal states.

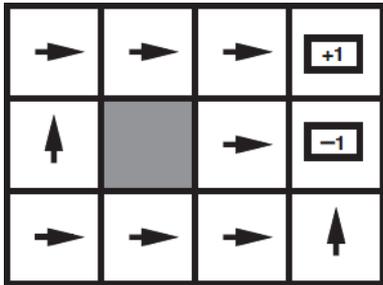
# Optimal Policies: Example

- An optimal policy for the stochastic environment with  $R(s) = -0.04$  in the nonterminal states.



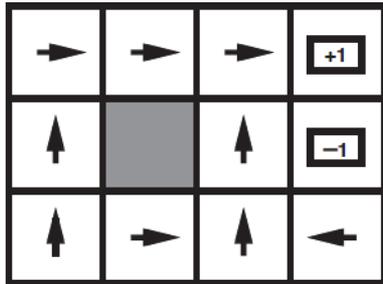
- Because the cost of taking a step is fairly small compared with the penalty for ending up in (4,2) by accident, the optimal policy for the state (3,1) is conservative.
- The policy recommends taking the long way round, rather than taking the shortcut and thereby risking entering (4,2).

# Optimal Policies: Example



$R(s) < -1.6284$

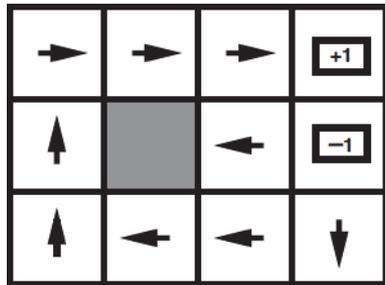
- When  $R(s) \leq -1.6284$ , life is so painful that the agent heads straight for the nearest exit, even if the exit is worth  $-1$ .



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

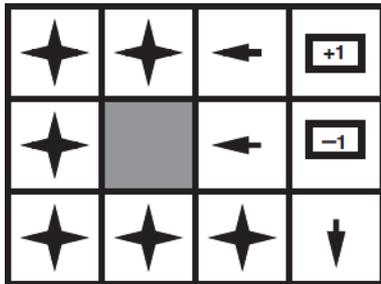
- When  $-0.4278 \leq R(s) \leq -0.0850$ , life is quite unpleasant; the agent takes the shortest route to the  $+1$  state and is willing to risk falling into the  $-1$  state by accident.
- In particular, the agent takes the shortcut from  $(3,1)$ .

# Optimal Policies: Example



$-0.0221 < R(s) < 0$

- When life is only slightly dreary ( $-0.0221 < R(s) < 0$ ), the optimal policy takes no risks at all.
- In (4,1) and (3,2), the agent heads directly away from the  $-1$  state so that it cannot fall in by accident, even though this means banging its head against the wall quite a few times.



$R(s) > 0$

- If  $R(s) > 0$ , then life is positively enjoyable and the agent avoids both exits.
- As long as the actions in (4,1), (3,2), and (3,3) are as shown, every policy is optimal, and the agent obtains infinite total reward because it never enters a terminal state.

# How to Find Optimal Policies?: Solving MDPs

- Two algorithms to calculate optimal policies.

## **Value Iteration:**

- The basic idea is to calculate the utility of each state and then use the state utilities to select an optimal action in each state.

## **Policy Iteration:**

- Alternative approach to obtain optimal values.
- Uses policy evaluation and policy improvement steps.

# Solving MDPs: Value Iteration

- There is a **direct relationship** between the **utility of a state** and the **utility of its neighbors**:
  - *The utility of a state is the **immediate reward** for that state plus the **expected discounted utility of the next state**, assuming that the agent chooses the optimal action.*
- The **optimal utility values**,  $U(s)$ , are linked to the **optimal policy**  $\pi^*$
- The **optimal policy** picks the action which **maximizes**  $U(s')$ , the expected utility of the successor state  $s'$ .
- Thus, the utility of a state  $s$  is linked to the utility of its neighbors (and only the utility of its neighbors, given the Markov property)



# Value Iteration

- The Bellman equation is the basis of the value iteration algorithm for solving MDPs.
  - If there are  $n$  possible states, then there are  $n$  Bellman equations, one for each state.
  - The  $n$  equations contain  $n$  unknowns — the utilities of the states.
  - So we would like to solve these simultaneous equations to find the utilities.
  - There is one problem: the equations are nonlinear, because the “max” operator is not a linear operator.
  - Whereas systems of linear equations can be solved quickly using linear algebra techniques, systems of nonlinear equations are more problematic.

# Value Iteration: Bellman Update

## Iterative Approach.

- Start with arbitrary initial values for the utilities,
  - Calculate the right-hand side of the equation, and plug it into the left-hand side — thereby updating the utility of each state from the utilities of its neighbors.
  - Repeat this until we reach an equilibrium.
- 
- Let  $U_i(s)$  be the utility value for state  $s$  at the  $i^{\text{th}}$  iteration.
  - The iteration step, called a **Bellman update**, is as follows:

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

# Value Iteration: Algorithm

**function** VALUE-ITERATION(*mdp*,  $\epsilon$ ) **returns** a utility function  
**inputs:** *mdp*, an MDP with states  $S$ , actions  $A(s)$ , transition model  $P(s' | s, a)$ ,  
rewards  $R(s)$ , discount  $\gamma$   
 $\epsilon$ , the maximum error allowed in the utility of any state  
**local variables:**  $U, U'$ , vectors of utilities for states in  $S$ , initially zero  
 $\delta$ , the maximum change in the utility of any state in an iteration

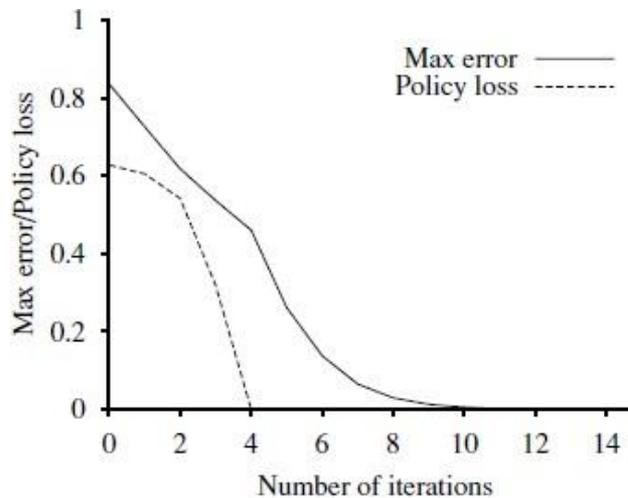
**repeat**  
     $U \leftarrow U'; \delta \leftarrow 0$   
    **for each state**  $s$  **in**  $S$  **do**  
         $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s']$   
        **if**  $|U'[s] - U[s]| > \delta$  **then**  $\delta \leftarrow |U'[s] - U[s]|$   
**until**  $\delta < \epsilon(1 - \gamma)/\gamma$   
**return**  $U$

Bellman update

Termination condition

# Value Iteration: Policy Convergence

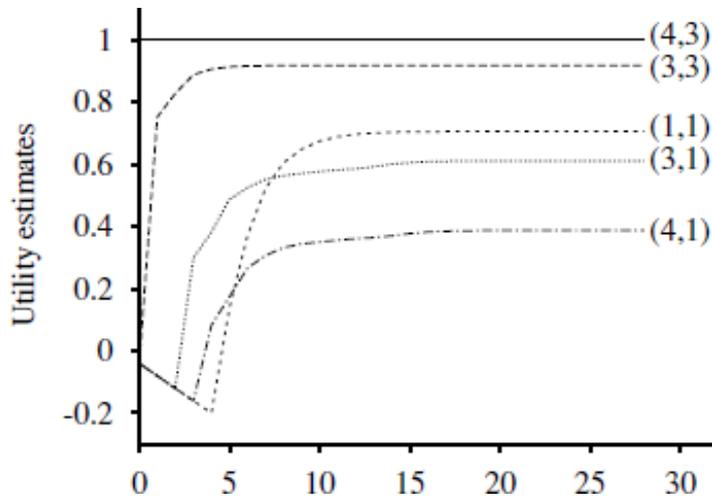
- Policy converges before utility.



- The maximum error  $\|U_i - U\|$  of the utility estimates and the policy loss  $\|U\pi_i - U\|$ , as a function of the number of iterations of value iteration.

# Value Iteration: Evolution of Utilities

- Apply value iteration to the  $4 \times 3$  world starting with initial values of zero
- The states at different distances from  $(4,3)$  accumulate negative reward until a path is found to  $(4,3)$ , whereupon the utilities start to increase.



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

# Solving MDPs: Policy Iteration

- It is possible to get an **optimal policy** even when the utility function estimate is inaccurate.
  - If one action is clearly better than all others, then the exact magnitude of the utilities on the states involved need not be precise.
- This insight suggests an alternative way to find optimal policies. → **Policy Iteration**
- What we really want to do is find the optimal policy  $\pi^*$ , which can be done even if the utility function  $U(s)$  is only approximate.
- We can search the space of possible policies  $\pi$ . This space is finite.
  - With  $n$  states and  $k$  actions per state, there are  $k^n$  policies.

## Policy Iteration steps:

1. Start with an arbitrary policy  $\pi_0$ .
2. With each iteration  $i$ , evaluate the policy  $\pi_i$  to obtain  $U_i$ .
3. Formulate an improved policy  $\pi_{i+1}$  by selecting the best action for each state  $s$  according to  $U_i(s)$ .

# Policy Iteration: Algorithm

**function** POLICY-ITERATION( $mdp$ ) **returns** a policy

**inputs:**  $mdp$ , an MDP with states  $S$ , actions  $A(s)$ , transition model  $P(s' | s, a)$

**local variables:**  $U$ , a vector of utilities for states in  $S$ , initially zero

$\pi$ , a policy vector indexed by state, initially random

**repeat**

$U \leftarrow$  POLICY-EVALUATION( $\pi, U, mdp$ )

$unchanged? \leftarrow$  true

**for each** state  $s$  **in**  $S$  **do**

**if**  $\max_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s'] > \sum_{s'} P(s' | s, \pi[s]) U[s']$  **then do**

$\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s']$

$unchanged? \leftarrow$  false

**until**  $unchanged?$

**return**  $\pi$

# Policy Iteration: Policy Evaluation

- At the  $i^{\text{th}}$  iteration, the policy  $\pi_i$  specifies the action  $\pi_i(s)$  in state  $s$ .
- So, we have a simplified version of the Bellman equation relating the utility of  $s$  (under  $\pi_i$ ) to the utilities of its neighbors:

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi_i(s)) U_i(s')$$

- These equations are linear, because the “max” operator has been removed.
- For  $n$  states, we have  $n$  linear equations with  $n$  unknowns, which can be solved exactly in time  $O(n^3)$  by standard linear algebra methods.

# Policy Iteration: Policy Evaluation

## *Simplified Version of Bellman Equation*

- Simplified version of the Bellman equation relating the utility of  $s$  (under  $\pi_i$ ) to the utilities of its neighbors:

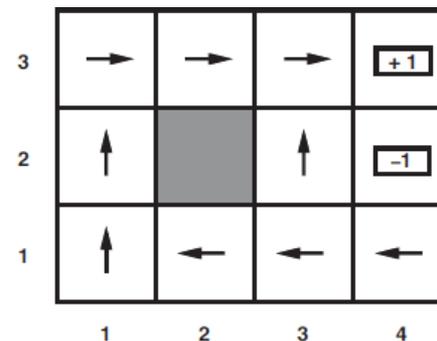
$$U_i(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi_i(s)) U_i(s')$$

**Example:** we have  $\pi_i(1, 1)=\text{Up}$ ,  $\pi_i(1, 2)=\text{Up}$ , and so on, and the simplified Bellman equations are:

$$U_i(1, 1) = -0.04 + 0.8U_i(1, 2) + 0.1U_i(1, 1) + 0.1U_i(2, 1) ,$$

$$U_i(1, 2) = -0.04 + 0.8U_i(1, 3) + 0.2U_i(1, 2) ,$$

$\vdots$



# Policy Iteration: Cheaper Policy Evaluation

- For small state spaces, policy evaluation using exact solution methods is often the most efficient approach.
- For large state spaces,  $O(n^3)$  time might be expensive.
- Fortunately, it is not necessary to do exact policy evaluation and we can perform *some number of simplified value iteration steps* (simplified Bellman updates) to give a reasonably good approximation of the utilities.
- The simplified Bellman update is

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

# Policy Iteration - Example

## A Markov Decision Process:

- **States:** {1,2}                      **Rewards:**  $R(1)=3$   $R(2)=2$
- **Actions:** {a, b}                      **Discount:**  $\gamma = 0.5$
- **Transition Model:**  $P(1|1,a) = 1.0$      $P(2|2,a) = 1.0$      $P(2|1,b) = 1.0$      $P(1|2,b) = 1.0$

Assume that

- **Initial policy  $\pi_0$ :**  $\pi_0(1)=a$   $\pi_0(2)=a$
- Find the **optimum policy** using *policy iteration algorithm*.
  - Give utility values at each iteration

# Policy Iteration - Example

**Rewards:**  $R(1)=3$   $R(2)=2$     **Discount:**  $\gamma = 0.5$

**Transition Model:**  $P(1|1,a) = 1.0$   $P(2|2,a) = 1.0$      $P(2|1,b) = 1.0$   $P(1|2,b) = 1.0$   
 =

$\pi_0 = \langle a, a \rangle$

$$\begin{array}{l}
 \mathbf{U}_0(1) = \mathbf{R}(1) + \gamma \mathbf{P}(1|1,a) \mathbf{U}_0(1) = + 0.5 * 1 * \mathbf{U}_0(1) \quad \rightarrow \quad \mathbf{U}_0(1) = 6 \\
 \mathbf{3} \qquad \qquad \qquad \qquad \qquad \qquad \mathbf{U}_0(1) \qquad \qquad \qquad \rightarrow \quad \mathbf{U}_0(2) = 4
 \end{array}$$

$$\begin{array}{l}
 \mathbf{U}_0(2) = \mathbf{R}(2) + \gamma \mathbf{P}(2|2,a) \mathbf{U}_0(2) = + 0.5 * 1 * \mathbf{U}_0(2) \\
 \mathbf{2} \qquad \qquad \qquad \qquad \qquad \qquad \mathbf{U}_0(2)
 \end{array}$$

$\mathbf{1:} \quad \mathbf{P}(1|1,a) \mathbf{U}_0(1) = 6 \qquad \mathbf{P}(2|1,b) \mathbf{U}_0(2) = 4 \qquad 6 > 4 \quad \rightarrow \quad \mathbf{a}$   
 $\mathbf{2:} \quad \mathbf{P}(2|2,a) \mathbf{U}_0(2) = 4 \qquad \mathbf{P}(1|2,b) \mathbf{U}_0(1) = 6 \qquad 6 > 4 \quad \rightarrow \quad \mathbf{b}$

Thus,  $\pi_1 = \langle a, b \rangle$ . Policy is changed, a new iteration is required.

# Policy Iteration - Example

**Rewards:**  $R(1)=3$   $R(2)=2$     **Discount:**  $\gamma = 0.5$

**Transition Model:**  $P(1|1,a) = 1.0$     $P(2|2,a) = 1.0$      $P(2|1,b) = 1.0$     $P(1|2,b) = 1.0$

$\pi_1 = \langle a,b \rangle$

$$U_1(1) = R(1) + \gamma P(1|1,a) U_1(1) = 3 + 0.5 * 1 * U_1(1) \quad \rightarrow \quad U_1(1) = 6$$

$$U_1(2) = R(2) + \gamma P(1|2,b) U_1(1) = 2 + 0.5 * 1 * 6 \quad \rightarrow \quad U_1(2) = 5$$

$$1: \quad P(1|1,a) U_1(1) = 6 \quad \quad P(2|1,b) U_1(2) = 5 \quad \quad 6 > 5 \quad \rightarrow \quad a$$

$$2: \quad P(2|2,a) U_1(2) = 5 \quad \quad P(1|2,b) U_1(1) = 6 \quad \quad 6 > 5 \quad \rightarrow \quad b$$

Thus,  $\pi_2 = \langle a,b \rangle$  policy is not changed.

$\pi_2$  is OPTIMUM POLICY.