

Chapter 17 (AIAMA)

Making Complex Decisions

Sukarna Barua
Associate Professor, CSE, BUET

Sequential Decision Problems

In a sequential decision problem:

✓ ■ Utility do not depend on a single decision/action.

- Example: You have two choices in a game: First choice gives you 10k with probability $\frac{1}{2}$ and 0k otherwise. Second choice gives you 30k with probability $\frac{1}{4}$ and 5k otherwise. Which action should a rational agent choose? It is simple to choose this single action!

✓ ■ Utility depends on a sequence of agent's decisions/actions.

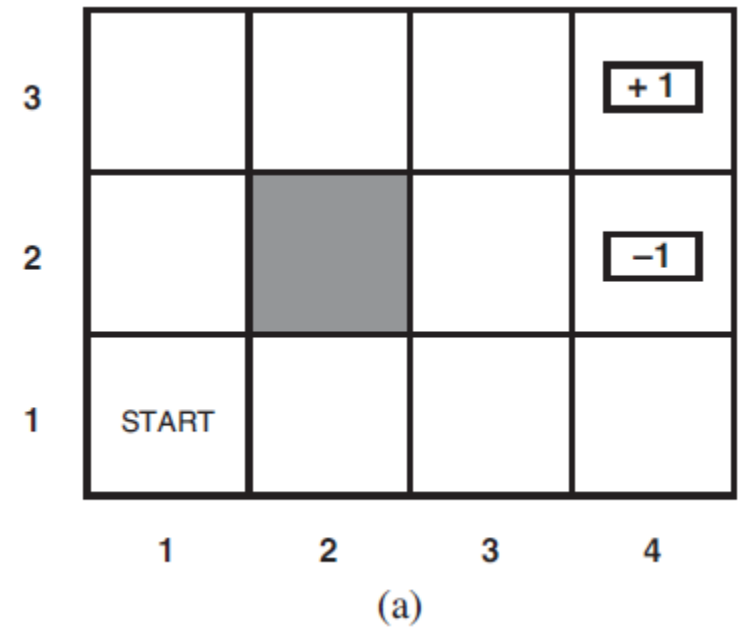
✓ - What an agent should do at any time depends on what it will do in the future.

✓ - What an agent does in the future depends on what it did before.

Sequential Decision Problems

Consider an agent's exploration in a 4×3 grid environment:

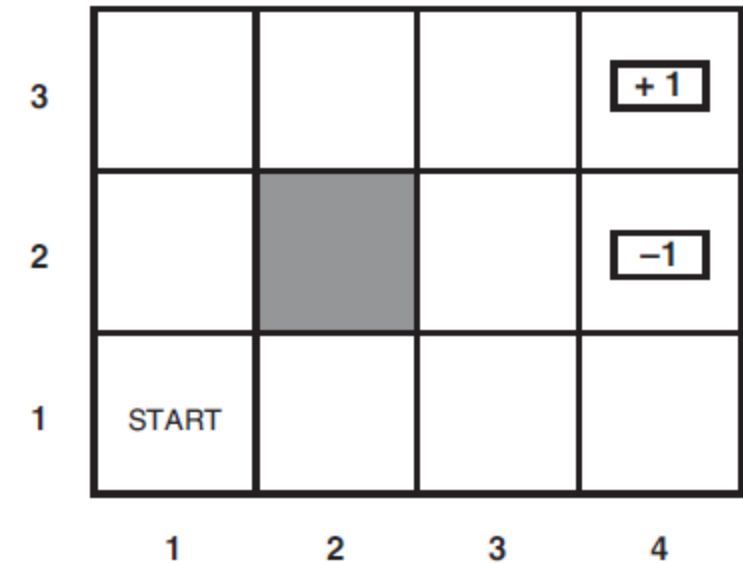
- ✓ Agent begins in the START state
- ✓ Agent chooses an action in each state
 - **Actions:** UP, LEFT, DOWN, RIGHT
- ✓ Exploration ends when agent reaches one of the goal states (+1 or -1)



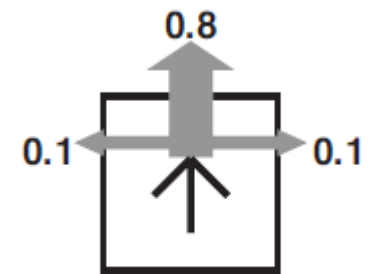
Sequential Decision Problems

Actions of agent:

- ✓ Agent chooses an action in each state
 - Actions: UP, LEFT, DOWN, RIGHT
- ✓ Action effect is not deterministic rather stochastic
 - ✓ Each action takes agent in the right direction with probability 0.8
 - ✓ With 0.1 probability, agent can move to the right angles of the intended direction
 - ✓ If agent hits a wall, it bumps and stays in the same location



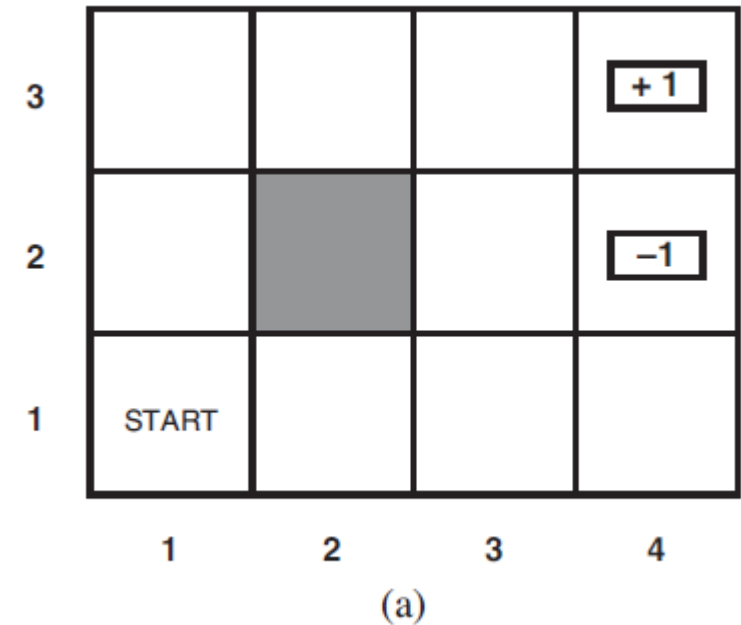
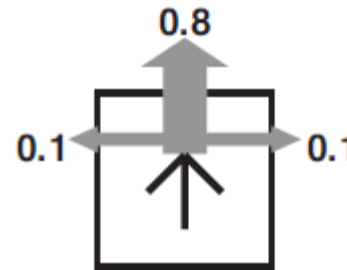
(a)



Sequential Decision Problems

Actions of agent: Example

- Agent chooses action **UP** in (1,1)
 - ✓ With 0.8 prob, agent moves to (1,2)
 - ✓ With 0.1 prob, agent moves to (2,1)
 - ✓ With 0.1 probability, moves to left, bumps to wall and stays at (1,1)

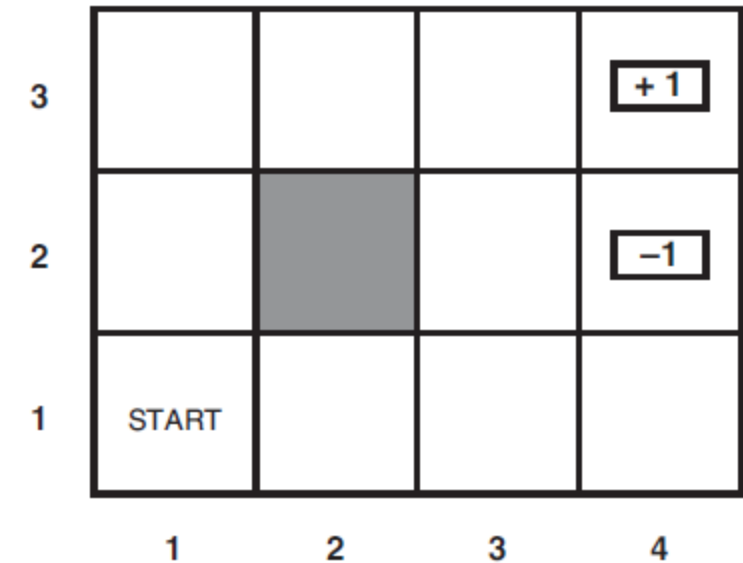


Sequential Decision Problems

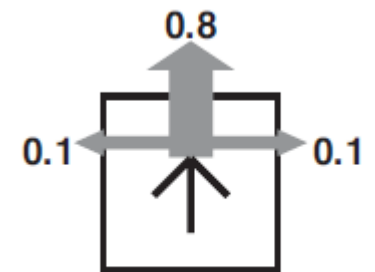
What if action effect was deterministic?

- The problem turns into a simple search problem:
 - Find the sequence of moves leading to goal.
 - DFS/BFS can easily find this.

- In sequential decision problem:
 - Action effect is not deterministic.
 - Simple search do not work [Why? We need action for
for every state instead of a shortest path to goal]

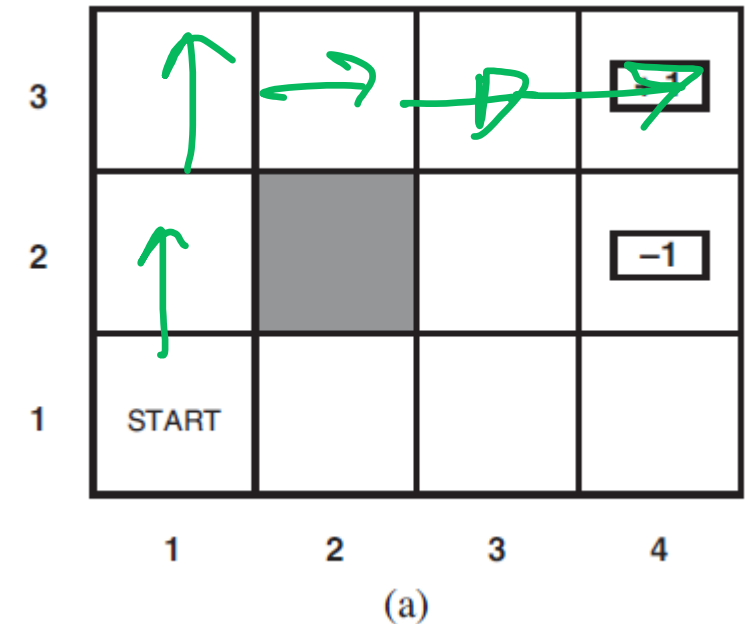


(a)



Sequential Decision Problems

- ✓ Agent cannot reach goal by following the action sequence: UP, UP, RIGHT, RIGHT, RIGHT. [Why?]
 - Because action effects are not deterministic
- ✓ What is the probability that agent reaches +1 by following the path given above as action sequence?
 - Answer: $0.8^5 = 0.32768$



Sequential Decision Problems

Rewards from the environment:

Agent receives a reward from the environment

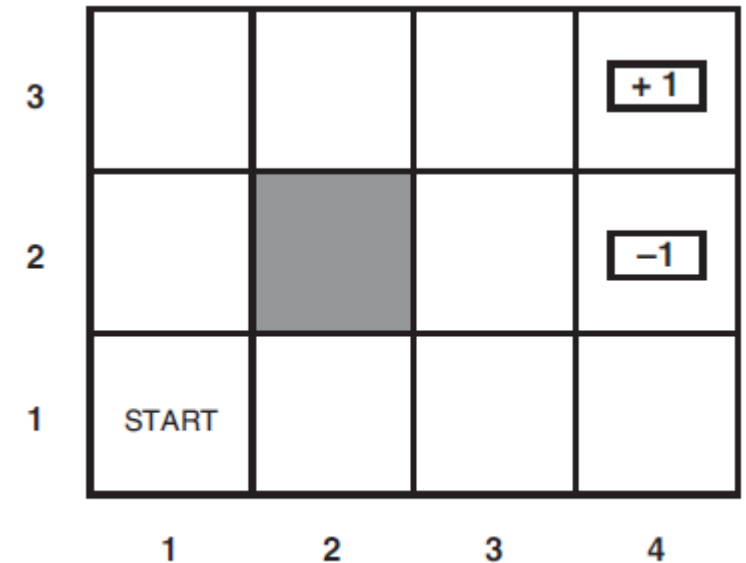
- For state s , reward is $R(s)$
- ✓ - $R(s) = -1.0$ and $+1.0$ for terminal states
- ✓ - $R(s) = -0.04$ in all other states

- The utility of a sequence of steps is the sum of all rewards.

- Say, agent reaches $+1$ state after 10 steps

✓ total utility = $10 \times -0.04 + 1.0 = 0.6$

Reward \rightarrow gain/loss
for per action



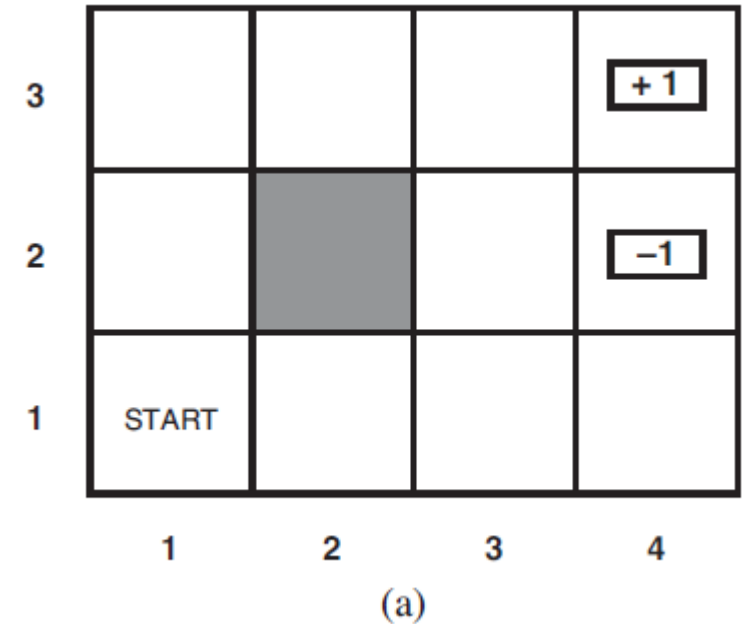
(a)

Utility \rightarrow sum of
rewards

Sequential Decision Problems

Utility from the environment:

- Agent tries to reach +1 as quickly as possible
- ✓ Otherwise, agent gets negative reward and utility decrease
- ✓ Consider, agent reaches +1 with 10 steps vs 5 steps.
 - ✓ For 10 steps, total utility received = 0.6
 - ✓ For 5 steps, total utility received = 0.8
- Real utility is gained only from terminal states.
 - ✓ However, utility can be decreased/increased by the cost/penalty incurred from visiting each state s to reach the terminal state.

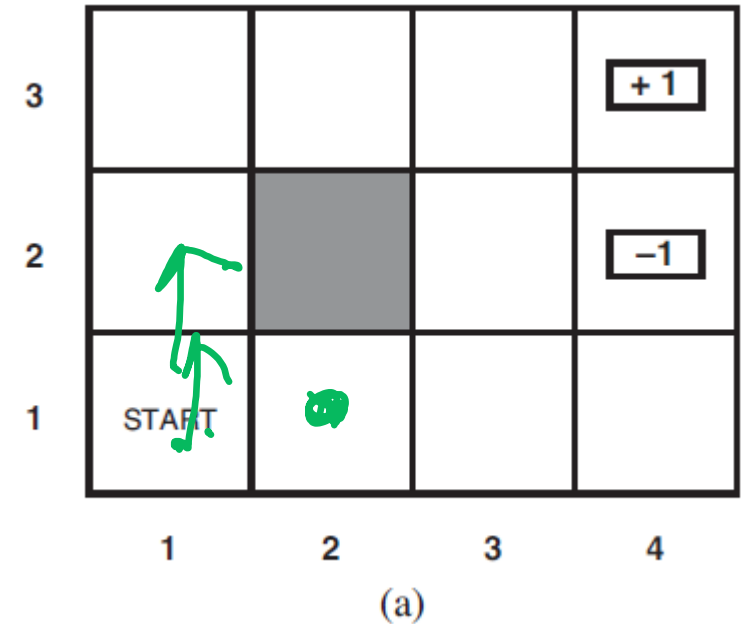


Sequential Decision Problems

$$P(s'|s, a)$$

Transition model:

- Agent moves to a new state by following an action in current state.
 - Modeled by a transition probability $P(s'|s, a)$
 - Can be described by a three-dimensional table specifying all probability values
- The transition model is Markovian
 - $P(s'|s, a)$ depends only on current state s and not on any previous states visited.
 - Example: $P((1,2)|(1,1), UP) = 0.8$, $P((2,1)|(1,1), UP) = 0.1$, etc.



Markov Decision Process

■ A Markov Decision Process (MDP) is a sequential decision problem where:

- Environment is fully observable [*Agent knows where it is*]
- Environment is stochastic [*Actions do not always leads to the intended state*]
- Transition model is Markovian
- Rewards are additive

■ A MDP consists of

- A set of states (with initial state s_0)
- A set of actions $ACTIONS(s)$ in each state s
- A transition model $P(s'|s, a)$
- A reward model $R(s)$

Markov Decision Process

- ✓ A Partially Observable Markov Decision Process (POMDP) is a sequential decision problem where:
 - Environment is partially observable
 - Agent does not necessarily know where it is in
 - Reward and action do not just depend on s but also on how much the agent knows when it is in s
- ✓ More complex than ordinary MDPs. [*We won't study in this course*]

Markov Decision Process: Solution

✓ ■ **A solution to an MDP is called a policy that defines**

■ Which action agent should follow for every state

■ Usually denoted by π or $\pi(s)$.

$\pi \rightarrow$ policy

✓ ■ If agent is given to follow a policy, the agent always takes the action given in the policy, no matter what the outcome is.

○ ■ **A policy π**

✓ ■ Is not an action-planning [*which actions leads to goal, simple search problem*]

- In fact, an action-sequence is not guaranteed to take to goal.

✓ ■ Is a strategy planning [*which action to take at each state, MDP*]

Markov Decision Process: Optimal Policy

- A policy π

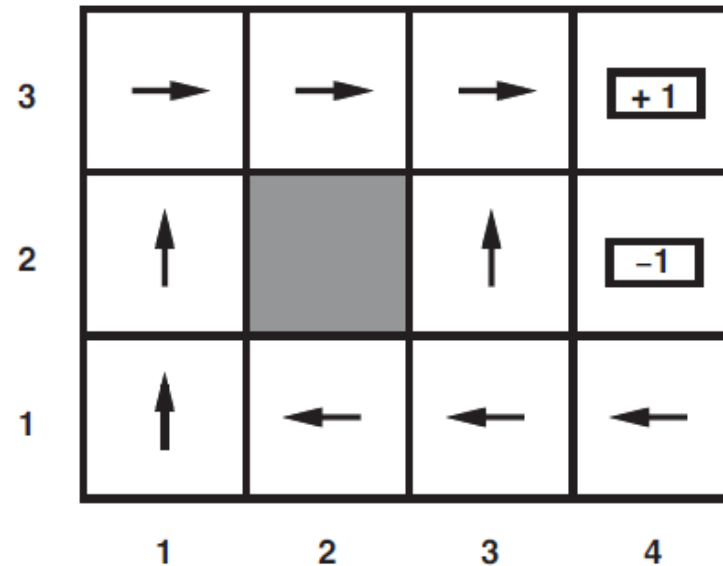
- Does not always takes an agent to the same state [*stochastic action effects*]
- Each execution of a policy may result in a different state sequence followed by the agent [*due to the stochastic nature of the action effect*].

- What makes a policy π **optimal**?

- A policy is optimal if it yields the highest expected utility [*not a reward*]
- Usually denoted by π^*

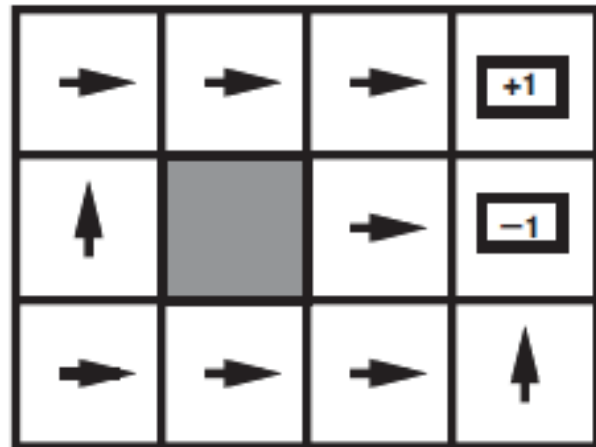
Markov Decision Process: Optimal Policy

- ✓ **Example:** (An optimal policy for the grid world problem with $R(s) = -0.04$ for nonterminal states.)

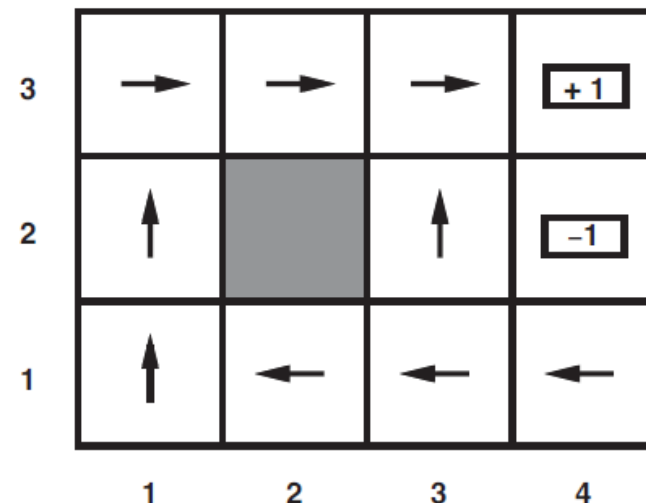


Markov Decision Process: Optimal Policy

- What happens to optimal policy 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 .
 - Note the action at $(3,2)$ [Agent chooses -1 to quickly exit]

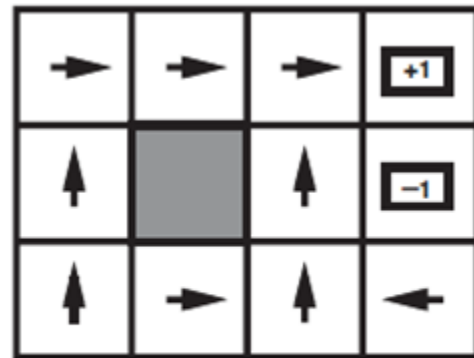


$R(s) < -1.6284$

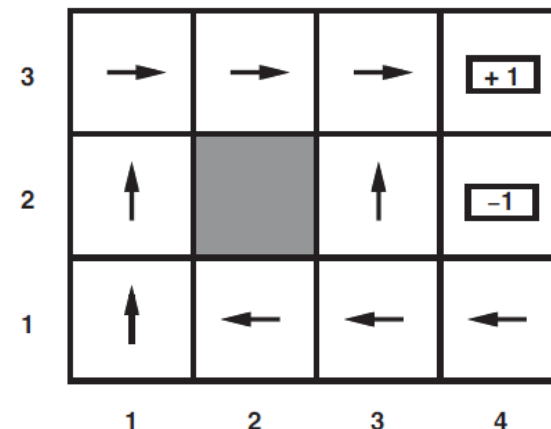


Markov Decision Process: Optimal Policy

- What happens to optimal policy when $-0.4278 < R(s) < -0.0850$?
- Life is not so painful but unpleasant; the agent takes the shortest route to +1, and is willing to risk falling to -1 by accident
- Note the action at (3,1) [Agent chooses UP to quickly exit]

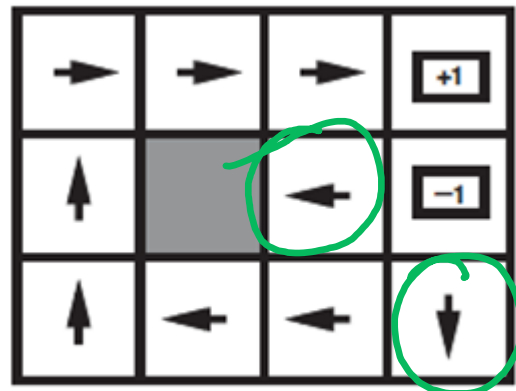


$$-0.4278 < R(s) < -0.0850$$

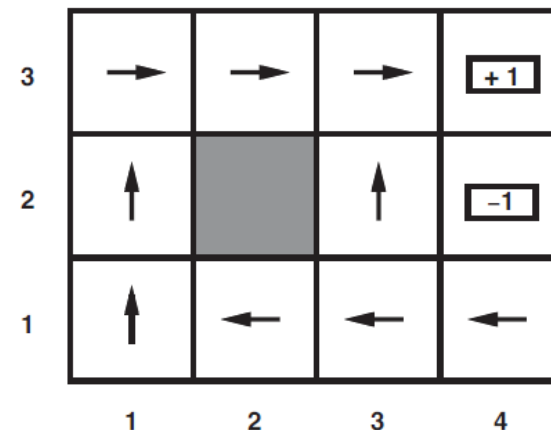


Markov Decision Process: Optimal Policy

- What happens to optimal policy when $-0.0221 < R(s) < 0$?
 - 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.



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



Markov Decision Process: Reward Types

- ✓ Assume an agent visits an state sequence $[s_0, s_1, \dots]$. The utility obtained is $U_h([s_0, s_1, \dots])$ can be considered in two ways:
 - ✓ **Additive rewards:** $U_h([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$
 - ✓ Already considered in our previous grid world example
 - ✓ **Discounted rewards:** $U_h([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$
 - ✓ Future rewards are discounted by a factor γ where $0 < \gamma \leq 1$.
 - ✓ γ close 1: Agent considers future rewards as highly as current rewards
 - ✓ γ close 0: Agent considers distant rewards insignificant

Markov Decision Process: Expected Utility is Bounded

- ✓ With discounted rewards, the utility obtained from visiting an infinite sequence of states is still finite as shown below:
 - ✓ - Assume rewards are bounded by $\pm R_{max}$ and $\gamma < 1$
 - ✓ - The total utility after infinite number of steps is:

$$U_h = \sum_{i=0}^{\infty} \gamma^i R(s_i) \leq \sum_{i=0}^{\infty} \gamma^i R_{max} = R_{max} \left(\frac{1}{1-\gamma} \right)$$

$$\boxed{U_h = R_{max} \frac{1}{1-\gamma}}$$

Markov Decision Process: Expected Utility of a Policy

- Assume agent starts at initial state s .
- Agent reaches state S_t at time t when executing a policy π [Note S_t is not a deterministic state; it is a random variable. Why?]
- Agent acts in an infinite-horizon [no limit on the # of steps, but exploration ends as soon as terminal state is reached]
- The probability distribution over the state sequences S_1, S_2, \dots visited by the agent is determined by:
 - The initial state s
 - The policy π
 - The transition model $P(s' | s, a)$

Markov Decision Process: Expected Utility for a Policy

- ✓ The expected utility obtained by executing a given policy is computed as:

$$✓ \boxed{U^\pi(s)} = E[\sum_{t=0}^{\infty} \gamma^t R(S_t)] \text{ where } S_0 = s$$

[We can't just sum $R(S_t)$ as $R(S_t)$ is a random variable having a probability distribution over states, so we need to take the expected value]

- ✓ Can we compute $U^\pi(s)$ using the above equation?

- ✓ Not easy!

- ✓ As t increases, computing $R(S_t)$ becomes extremely difficult [*with exponential number of values for the random variable S_t , Why?*]

Markov Decision Process: Optimal Policy

- **Optimal Policy:** The optimal policy is the one that gives the maximum expected utility:

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

- π_s^* is the optimal policy starting with state s .
- π_s^* is a policy; hence it recommends an action for every state.

- Optimal policy is independent of start state s . [Why?]

Remember policy is like a function specifying an action for every state; hence it does not matter where you start; you always have an action-sequence as per the optimal policy.

- We can simply write the optimal policy as π^* .

Markov Decision Process: Utility Function

- The maximum achievable utility starting from any state s is given by: $U^{\pi^*}(s)$
- The sum of the discounted rewards if agent executed optimal policy starting from s .
- We simply express it $U(s) [= U^{\pi^*}(s)]$
 - Note $U(s)$ and $R(s)$ are quite different quantities!
 - Note $U(s)$ and $U^{\pi}(s)$ are two different quantities!
 - $U(s)$ depends on s [Why?]

Markov Decision Process: Utility Function

- Following figure shows the maximum utility $U(s)$ for different states in the grid world problem for $\gamma = 1$ and $R(s) = -0.04$
- Note that states closer to $+1$ have higher utilities compared to states far from it.
 - Why? [Because starts far away needs more steps to reach $+1$, thus additive rewards (-0.04 per state) reduces the utility]

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

Markov Decision Process: Optimal Action

- ✓ Agent can use the utility function $U(s)$ to choose the best action at each state:
 - ✓ Choose the action a in state s which gives you maximum utility from the next possible states; such action-sequence defines an optimal policy.

$$\checkmark \pi^*(s) = \operatorname{argmax}_{a \in A(s)} \sum_{s'} U(s') P(s'|s, a)$$

- ✓ The above is the optimal policy equation.

Markov Decision Process: Value Iteration

- ✓ **Value iteration:** An algorithm to calculate the optimal policy
- ✓ Basic idea of the algorithm:
 - ✓ Step 1: Calculate utility $U(s)$ for every state s [Remember $U(s)$ is the maximum utility achievable from state s by following optimal policy]
 - ✓ Step 2: Calculate optimal action for each state based on calculated $U(s)$ in Step 1 using optimal policy equation [*see previous slide*].
 - ✓ Seems chicken and egg problem? [*As in Step 1, we don't have an optimal policy!*]
 - No! $U(s)$ can be calculated without knowing optimal policy!

Markov Decision Process: Bellman Equation

- **Utility function:** Let's define utility function in a different way.

- The optimal utility $U(s)$ of a state s satisfies the following equation:

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

- $U(s)$ = immediate reward + expected discounted utility of next state [*assume agent follows optimal action as per optimal policy*]

- This is optimal substructure property for the optimal utility!

- This is called **Bellman equation**.

Markov Decision Process: Bellman Equation Example

Let's verify Bellman Equation for grid world problem.

Bellman equation for state (1,1) is:

$$U(1,1) = -0.04 + \gamma \max \begin{cases} 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), & (Up) \\ 0.9U(1,1) + 0.1U(1,2), & (Left) \\ 0.9U(1,1) + 0.1U(2,1), & (Down) \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) \end{cases} \quad (Right)$$

Plugging in numbers from the figure:

We can show UP is the best action

Choose any other action (i.e., RIGHT) and verify Bellman equation won't be satisfied

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

Markov Decision Process: Value Iteration Algorithm

- ✓ Based on Bellman equations
- ✓ Number of states = n , Number of Bellman equations = n
- ✓ Number of unknowns = n [$U(s)$ for every state s]
- ✓ If equations were linear, we could easily use Gauss Elimination to solve.
 - ✓ Bellman equations are non-linear due to the max operation!
 - ✓ Hence, easier approaches won't help
- ✓ Solution: iterative optimization known as **Value Iteration** algorithm.

Markov Decision Process: Value Iteration Algorithm

- Iterative approach to solve Bellman equations [Value Iteration Algorithm]
 - Start with random values for U s [*usual practice: start with all zeros*]
 - Repeat until equilibrium:
 - Calculate right hand side of Bellman equations using current values of U 's and use it to update left-hand side (new updated values for U 's):

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

[The above step is known as Bellman update]

Markov Decision Process: Value Iteration Algorithm

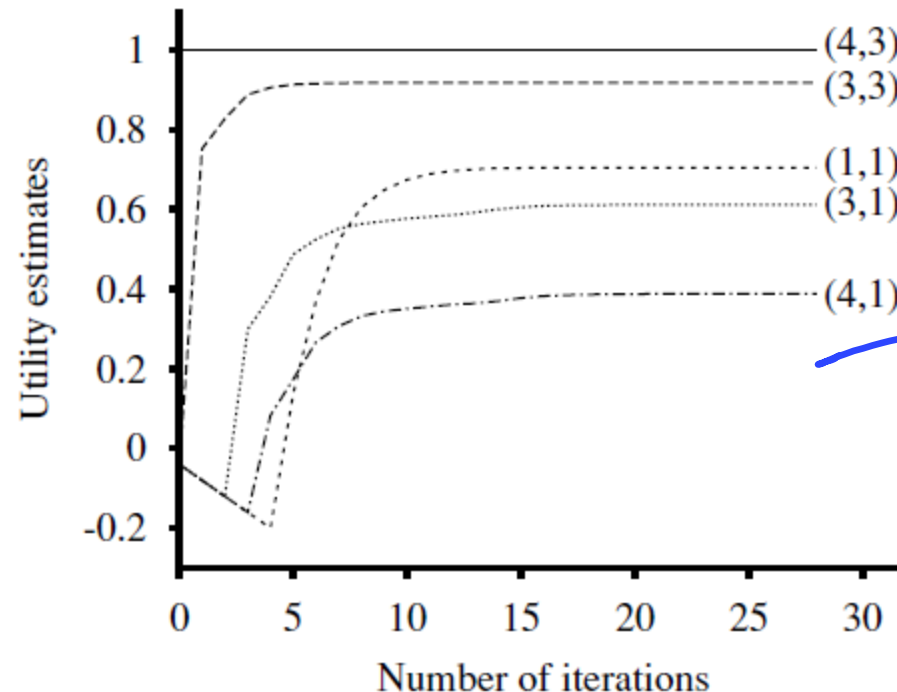
■ Pseudocode of value iteration algorithm:

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


Markov Decision Process: Value Iteration Example

- Value iteration applied to grid world problem with initial utilities as 0s.
- Note the number of iterations required for convergence of utilities for different states.



each line is
for each cell
(state).

Markov Decision Process: Value Iteration Algorithm

- ✓ Does the value iteration converge to a solution?
- ✓ If apply Bellman update infinitely often, the equilibrium is guaranteed.
 - ✓ Final utility values must be the solutions to Bellman equations.
 - ✓ The solution is also unique.
 - ✓ *Proof of convergence*: We won't study in this course. 
- ✓ Once utilities are obtained, we can also find the optimal policy! [*best action from each state satisfying Bellman equation*]

Markov Decision Process: Policy Iteration Algorithm

- Is it possible to get an optimal policy even when the utility function estimate is inaccurate?
 - *Answer: Yes.* If one action is clearly better than all others, then the exact magnitude of the utilities on the states involved need not be precise.
- An alternative approach can be designed using the above idea. This is known as **Policy Iteration.**

Markov Decision Process: Policy Iteration Algorithm

Steps of Policy Iteration Algorithm

- Start with a random policy π_0

- Repeat until convergence:

- Policy Evaluation: Using the current policy π_i , calculate utilities U_i s as:

$$U_i(s) = U^{\pi_i}(s)$$

- Policy improvement: Using the utilities U_i 's, calculate a new updated policy π_{i+1} as:

$$\pi_{i+1}(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} U_i(s') P(s'|s, a)$$

- Termination criteria: The above algorithm terminates when policy improvement yields no change.

Markov Decision Process: Policy Iteration Algorithm

- How to evaluate the two steps?
 - ✓ Policy improvement: Pretty straightforward.
 - ✓ Policy evaluation: Not easy thought! [we discussed it previously]
 - *Solution?* Use an approach similar to the value iteration algorithm.
 - *One important difference*: Bellman equations for unities will be linear now! [why?]

Markov Decision Process: Policy Iteration Algorithm

■ Policy evaluation:

- ✓ Utilities $U_i(s)$ of each state will satisfy the following simpler version of Bellman equation:

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

- ✓ The equations are linear [no max operation over actions as the policy is given and agent just follows the action given in the policy]
- ✓ We have n equations with n unknowns; all equations are linear.
- ✓ Gaussian Elimination (or similar methods) will solve this.

Markov Decision Process: Policy Iteration Algorithm

- *Policy evaluation:*

- Small state space: (Gaussian elimination with $O(n^3)$ run-time.)
- For large space: Use modified policy iteration to find U 's
 - Same as policy iteration algorithm
 - Use small number of iterations as exact solution is not needed

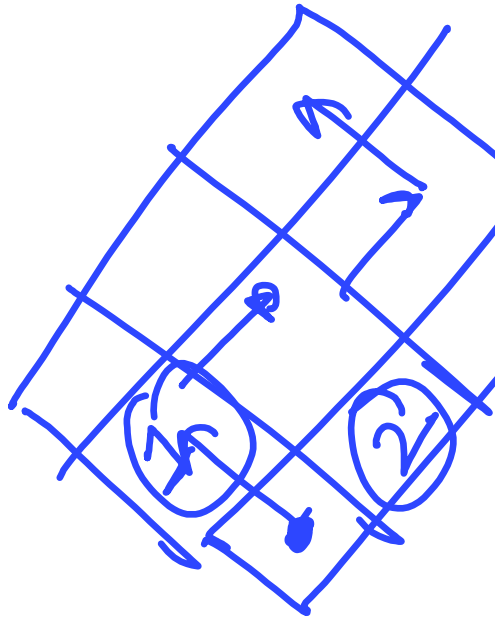
Markov Decision Process: Policy Iteration Algorithm

▪ Pseudocode:

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
 π , 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  $\pi$ 
  
```



SP