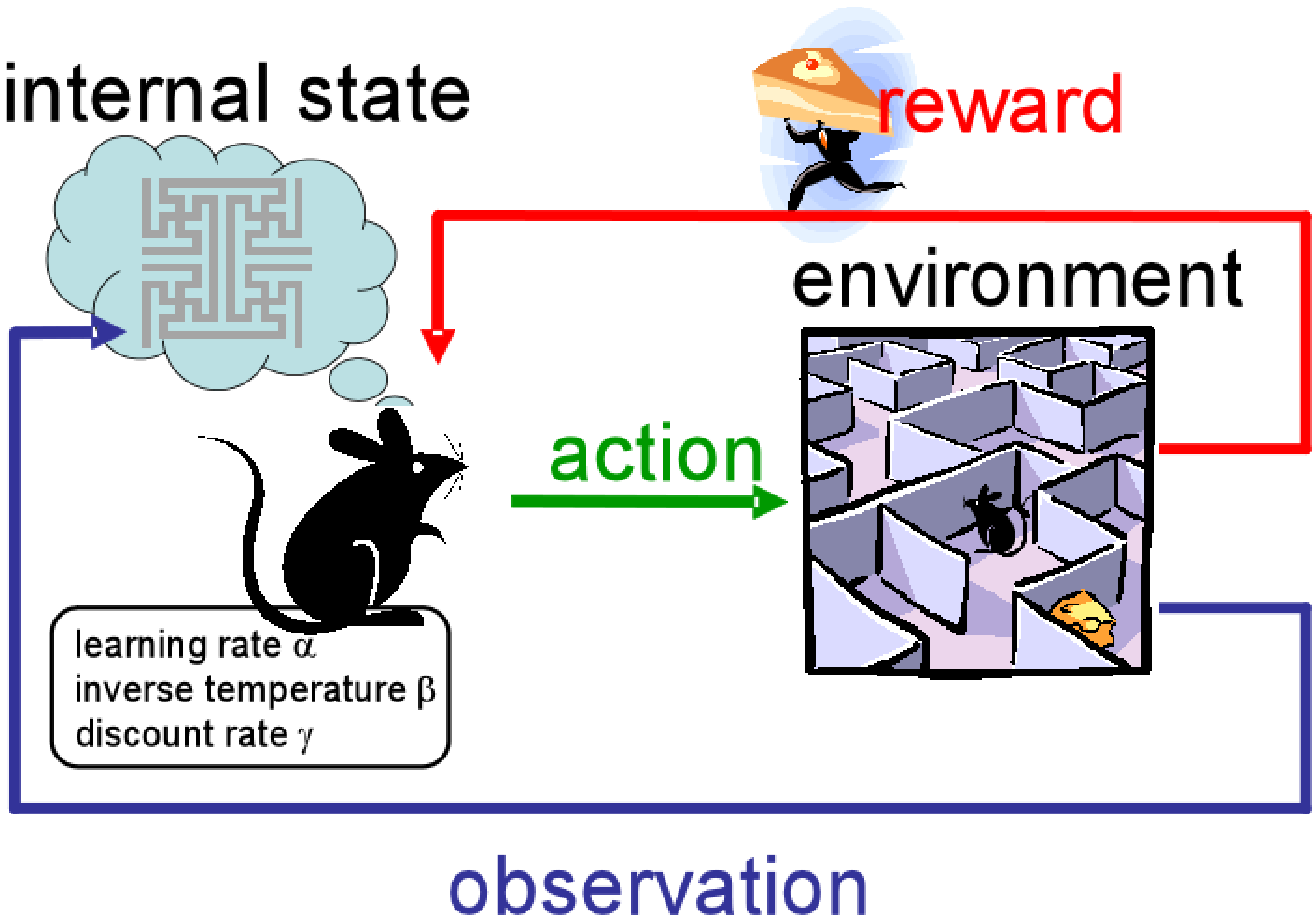
Reinforcement learning: Markov Decision Processes

CS434

Reinforcement learning



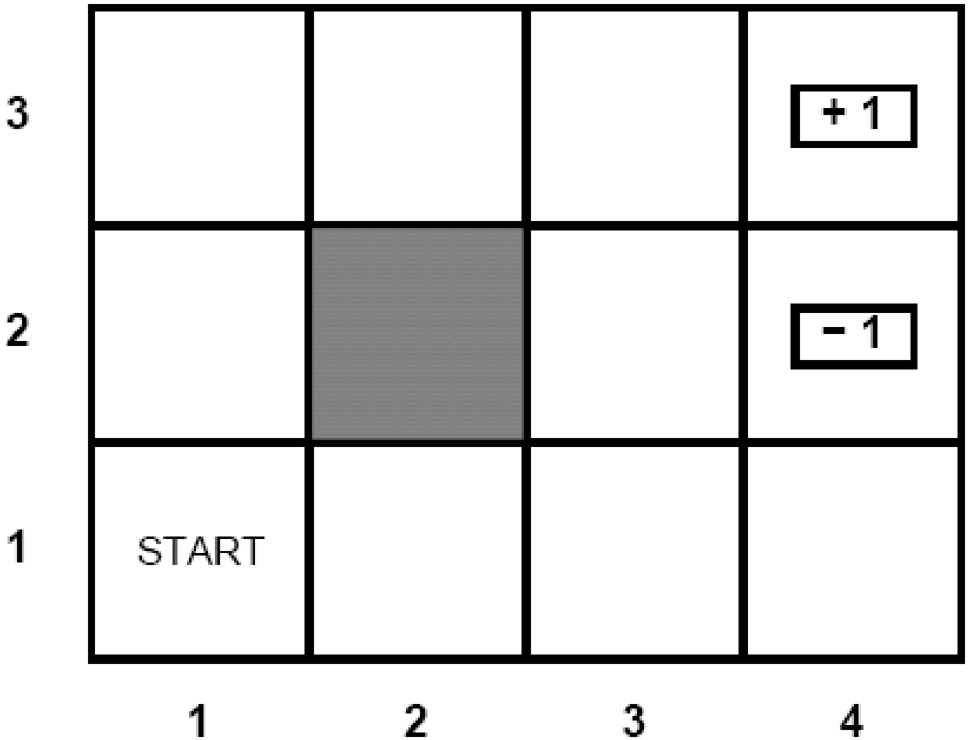
Reinforcement Learning

- You can think of supervised learning as the teacher providing answers (the class labels)
- In reinforcement learning, the agent learns based on a punishment/reward scheme
- Before we can talk about reinforcement learning, we need to introduce Markov Decision Processes

Decision Processes: General Description

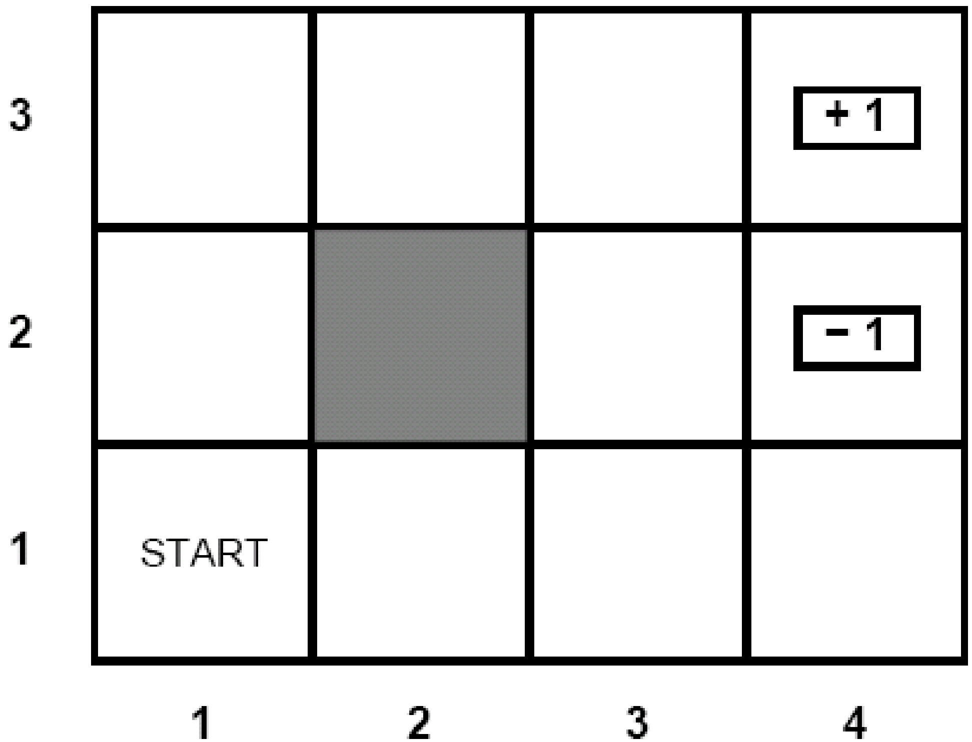
- Decide what action to take next given that your action will affect what happens in the future
- Real world examples:
  - Robot path planning
  - Elevator scheduling
  - Travel route planning
  - Aircraft navigation
  - Manufacturing processes
  - Network switching and routing

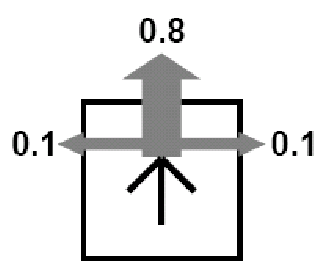
Sequential Decisions



- Assume a fully observable, deterministic environment, like the example shown here
- Each grid cell is a state
- The goal state is marked +1
- At each time step, agent must move Up, Right, Down, or Left
- How do you get from start to the goal state?

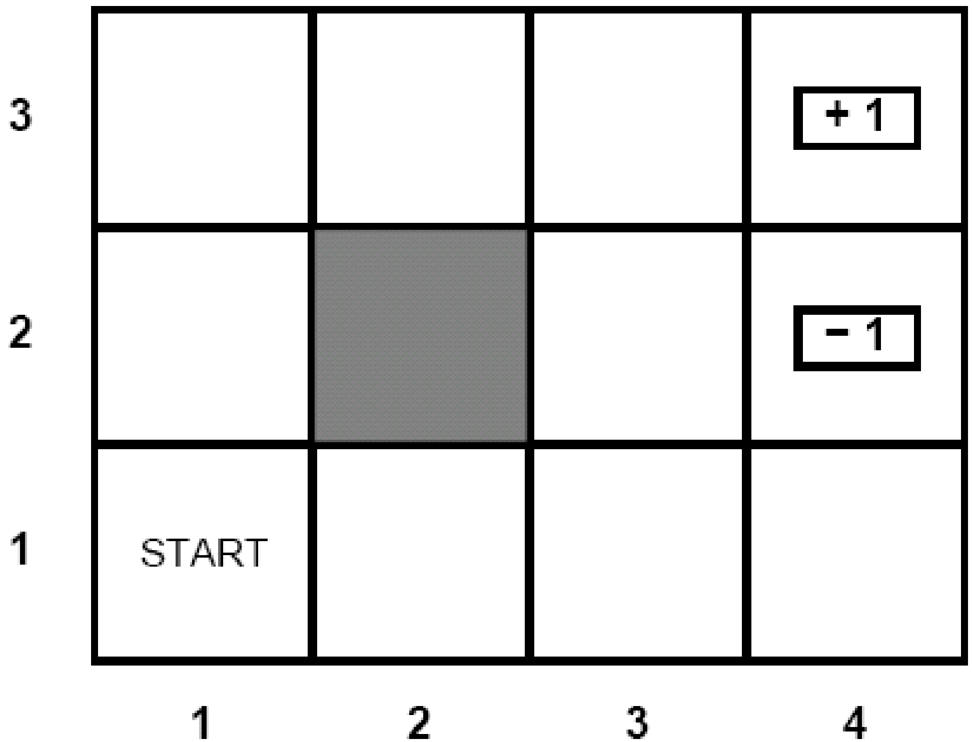
Sequential Decisions





- Suppose the environment is now stochastic
- With 0.8 probability you go in the direction you intend
- With 0.2 probability you move at right angles to the intended direction (0.1 in either direction – if you hit the wall you stay put)
- What is the optimal solution now?

Sequential Decisions



- *Up*, *Up*, *Right*, *Right*, *Right* reaches the goal state with probability 0.8^5=032768
- But in this stochastic world, going *Up*, *Up*, *Right*, *Right*, *Right* might end up with you actually going *Right*, *Right*, *Up*, *Up*, *Right* with probability $(0.1^4)(0.8)$=0.00008
- However, you might end up in the -1 state accidentally

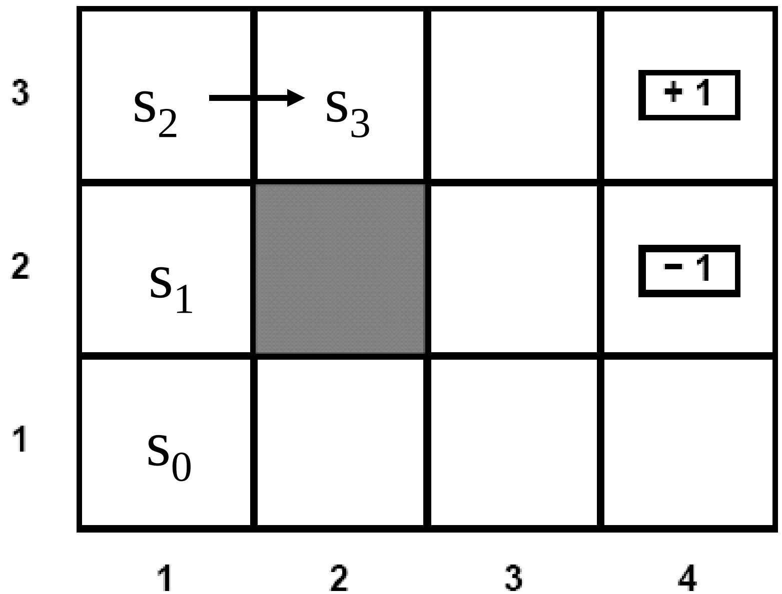
The Effect of Action: Transition Model

- Transition model: specifies the **outcome probabilities for each action in each possible state**
- $T(s, a, s')$ $=$ probability of going to state s' if you are in state s and do action a
- The transitions are **Markovian**, ie. the probability of reaching state s' from s depends only on s and not on the history of earlier states (aka The Markov Property)
- Mathematically:

  Suppose you visited the following states in chronological order: $s_0, \dots, s_t$

$$P(s_{t+1} \mid a, s_0, \dots, s_t) = P(s_{t+1} \mid a, s_t)$$

Markov Property Example



Suppose
$s_0 = (1,1), s_1 = (1,2), s_2 = (1,3)$

If I go *Right* from state s_2, the probability of going to s_3 only depends on the fact that I am at state s_2 and not the entire state history $\{s_0, s_1, s_2\}$

The Reward Function

- At each state s, the agent receives a reward $R(s)$ which may be positive or negative (but must be bounded)
- Based on reward, we can define the utility of a sequence of state
- For now, we'll define **the utility of a sequence of states** as the sum of the rewards received

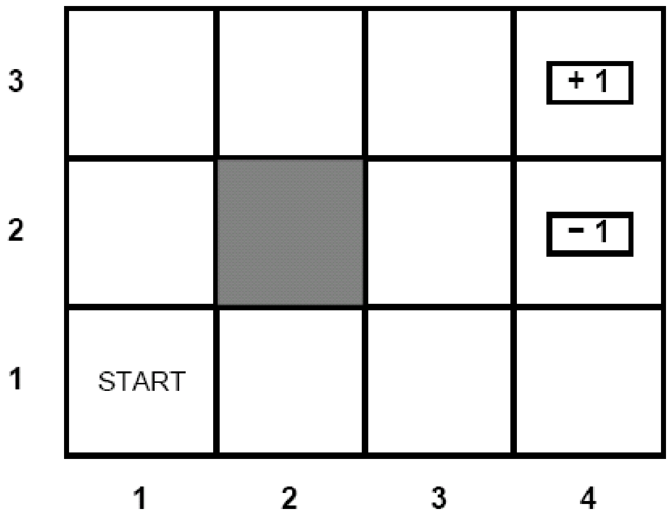
Reward Function Example

$R(4,3) = +1$ (Agent wants to get here)

$R(4,2) = -1$ (Agent wants to avoid this)

$R(s) = -0.04$ (for other states as it takes one step)

$U(s_1, \ldots, s_n) = R(s_1) + \ldots + R(s_n)$

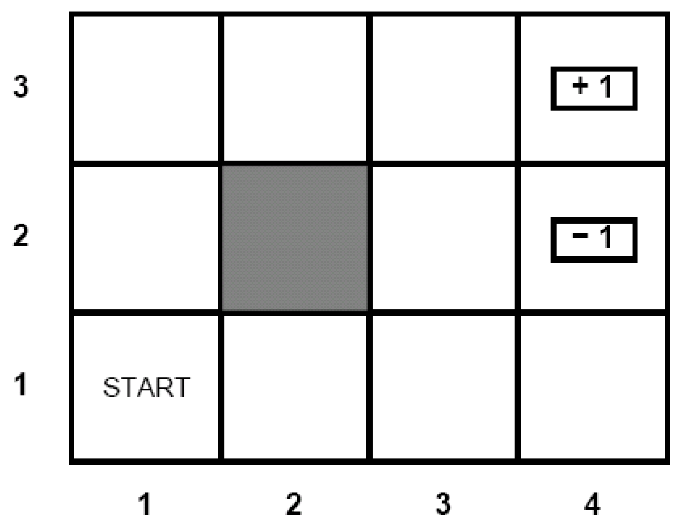


If the states an agent goes through are Up, Up, Right, Right, Right, the utility of this state sequence is:

-0.04-0.04-0.04-0.04-0.04+1

Reward Function Example

If there's no uncertainty, then the agent would find the sequence of actions that maximizes the sum of the rewards of the visited states



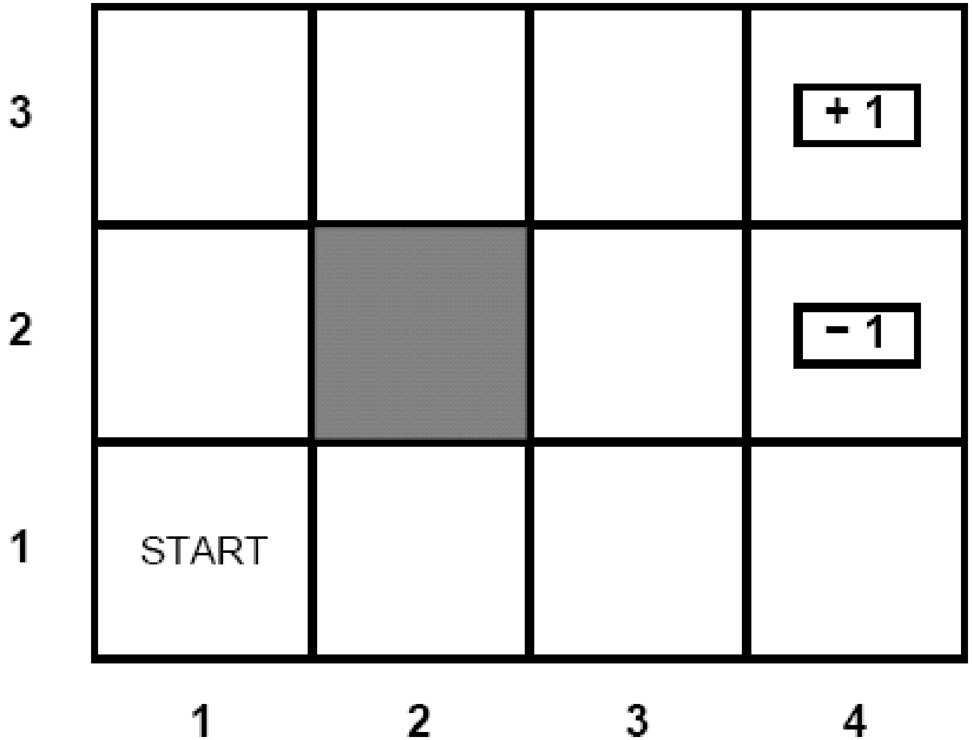
Markov Decision Process

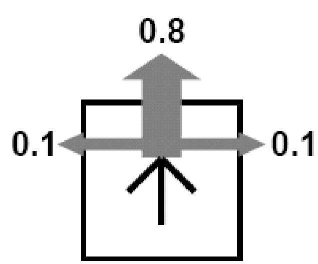
A sequential decision problem with a fully observable environment with a *Markovian transition model* and *additive rewards* is modeled by a Markov Decision Process (MDP)

An MDP has the following components:

1. A (finite) set of states S
2. A (finite) set of actions A
3. Transition Model: $T(s, a, s') = P(s' \mid a, s)$
4. Reward Function: $R(s)$

Example





- State space:

(1,1), (1,2) ……(4,3)

- Action space:

up, down, right, left

- Transition probability:

With 0.2 probability you move at right angles to the intended direction (0.1 in either direction – if you hit the wall you stay put)

- Reward:

R(4,3)=1, R(4,2)=-1, everywhere else: -0.04

Solutions to an MDP

- Why is the following not a satisfactory solution to the MDP?

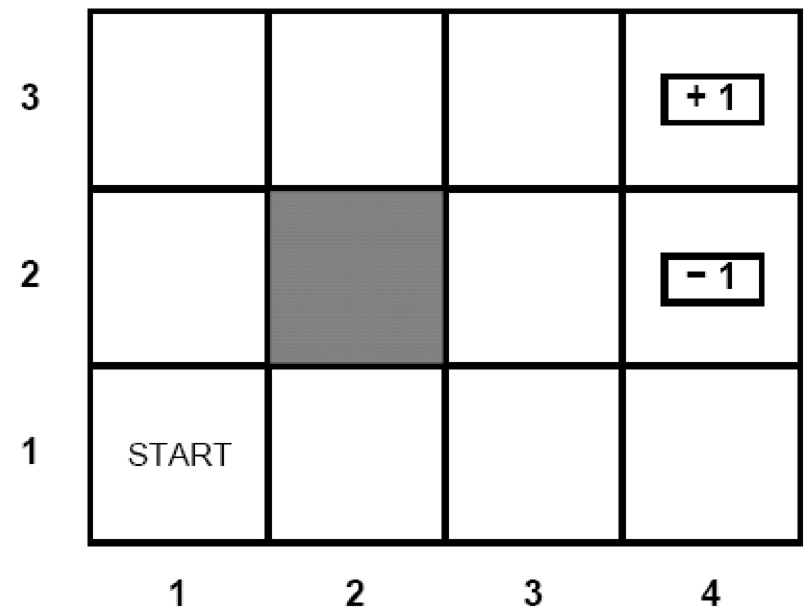
  [1,1]-Up

  [1,2]-Up

  [1,3]-Right

  [2,3]-Right

  [3,3]-Right



Solution to an MDP: Policy

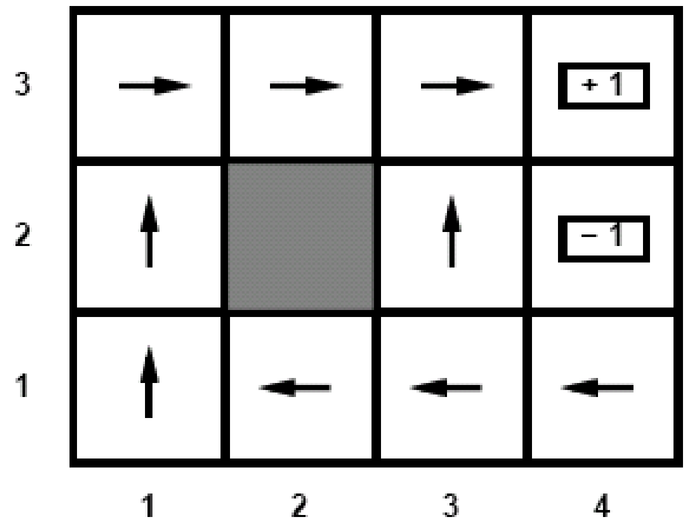
- **Policy**: **mapping from a state to an action**
- Need to be defined for all states so that the agent will always know what to do
- Notation:
  - π denotes a policy
  - $\pi(s)$ denotes the action recommended by the policy π for state s

Optimal Policy

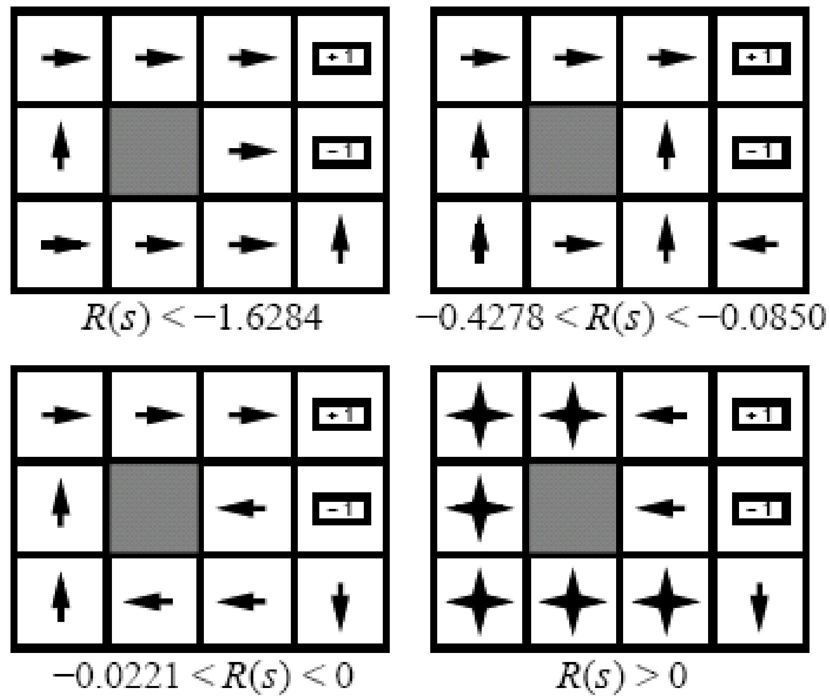
- Many different policies exist for an MDP
- Some are better than others. The “best” one is called the optimal policy π^* (we will define best more precisely in later slides)
- Note: every time we start at the initial state and execute a policy, we get a different state sequence (because the environment is stochastic)
- We get a different utility each time we execute a policy
- Need to measure the expected utility, i.e. the average utility of possible state sequences generated by the policy

Optimal Policy Example

R(s)=-0.04



(a)



(b)

Notice the tradeoff between risk and reward!

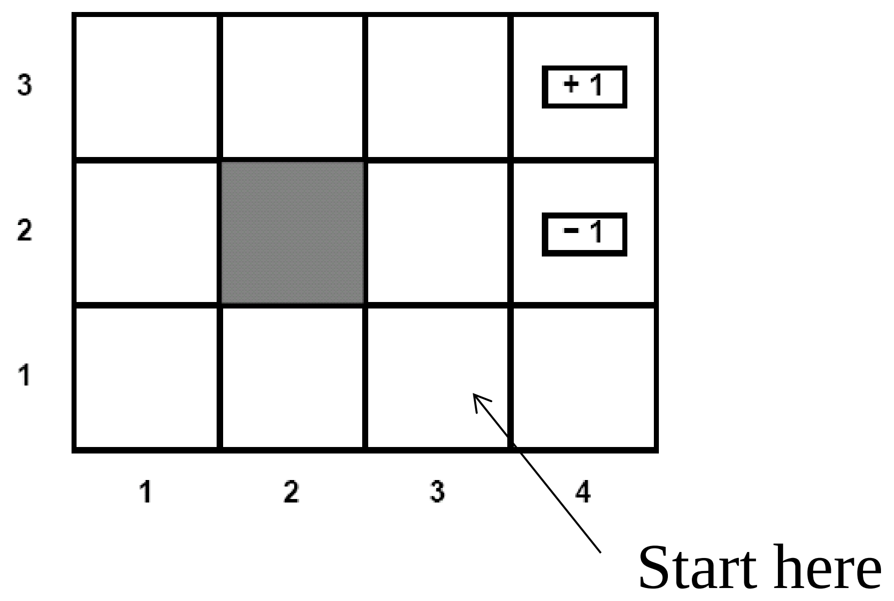
Roadmap for the Next Few Slides

We need to describe how to compute optimal policies

1. Before we can do that, we need to define the *utility* of a state sequence
2. Before we can do (1), we need to explain the *stationarity* assumption
3. Before we can do (2), we need to explain *finite/infinite horizons*

Finite/Infinite Horizons

- Finite horizon: fixed time N after which nothing matters (think of this as a deadline)
- Suppose our agent starts at (3,1), R(s)=-0.04, and N=3. Then to get to the +1 state, agent must go up.
- If N=100, agent can take the safe route around



Nonstationary Policies

- Nonstationary policy: the optimal action in a given state changes over time
- With a finite horizon, the optimal policy is nonstationary
- With an infinite horizon, there is no incentive to behave differently in the same state at different times
- With an infinite horizon, the optimal policy is stationary
- We will assume infinite horizons

Stationary Preference

- To calculate the utility of state sequences, you need a stationary preference assumption
- Suppose you have two state sequences:
  $[s_0, s_1, s_2, \ldots]$
  $[s_0', s_1', s_2', \ldots]$
- Suppose they begin with the same state ($s_0 = s_0'$)
- Then the two sequences should be preferred in the same order as $[s_1, s_2, \ldots]$ and $[s_1', s_2', \ldots]$
- If you prefer one future to another starting tomorrow, you should still prefer that future if it started today

Utility of a State Sequence

Under stationarity, there are two ways to assign utilities to sequences:

1. Additive rewards: The utility of a state sequence is:

   $U(s_0, s_1, s_2, \ldots) = R(s_0) + R(s_1) + R(s_2) + \ldots$

2. Discounted rewards: The utility of a state sequence is:

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

   Where $0 \leq \gamma \leq 1$ is the discount factor

The Discount Factor

- Describes preference for current rewards over future rewards
- Compensates for uncertainty in available time (models mortality)
- Eg. Being promised $10000 next year is only worth 90% of being promised $10000 now
- γ near 0 means future rewards don't mean anything
- γ = 1 makes discounted rewards equivalent to additive rewards

Utilities

We assume infinite horizons. This means that if the agent doesn't get to a terminal state, then environmental histories are infinite, and utilities with additive rewards are infinite. How do we deal with this?

Discounted rewards makes utility finite.

Assuming largest possible reward is R_{max} and $\gamma < 1$,

$$U(s_0, s_1, s_2, \ldots) = \sum_{t=0}^{\infty} \gamma^t R(s_t)$$

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

Computing Optimal Policies

- A policy π generates a sequence of states
- But the world is stochastic, so a policy π has a range of possible state sequences, each of which has some probability of occurring
- The value of a policy is the *expected sum of discounted rewards* obtained by following this policy

The Optimal Policy

- Given a policy π, we write the expected sum of discounted rewards obtained as:

$$E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi\right]$$

- An optimal policy π* is the policy that maximizes the expected sum above

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi\right]$$

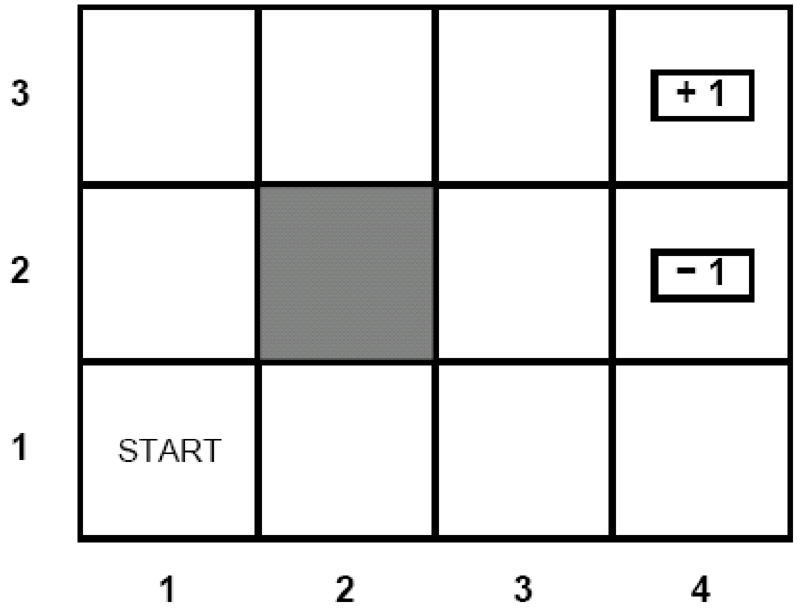
The Optimal Policy

- For every MDP, there exists an optimal policy
- There is no better option (in terms of expected sum of discounted rewards) than to follow this policy
- How do you calculate this optimal policy? Can't evaluate all policies…too many of them
- Instead, we calculate the ***utility of the states***
- Then use the state utilities to select an optimal action in each state

Rewards vs Utilities

- What's the difference between R(s) the reward for a state and U(s) the utility of a state?
  - R(s) – the short term reward for being in s
  - U(s) – The long-term total expected reward for the sequence of states starting at s (not just the reward for state s)

Utilities in the Maze Example

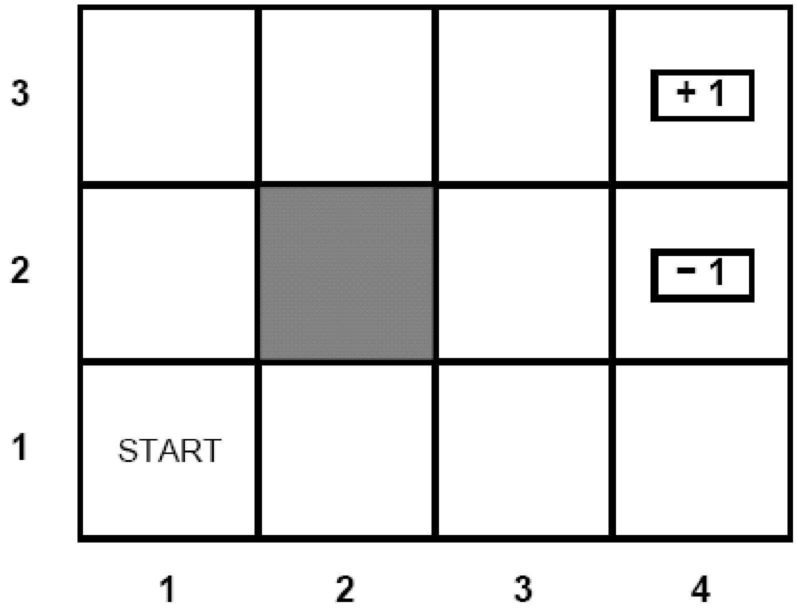


Start at state (1,1). Let's suppose we choose the action Up.

$$U(1,1) = R(1,1) + \ldots$$

Reward for current state

Utilities in the Maze Example



Start at state (1,1). Let's choose the action Up.

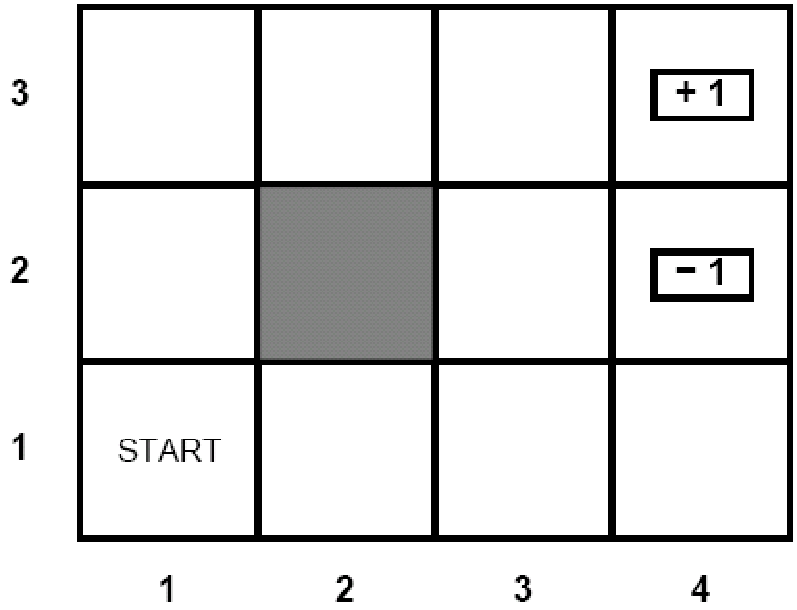
$$U(1,1) = R(1,1) + 0.8*U(1,2) + 0.1*U(2,1) + 0.1*U(1,1)$$

Prob of moving right

Prob of moving up

Prob of moving left (into the wall) and staying put

Utilities in the Maze Example



Now let's throw in the discounting factor

$U(1,1) = R(1,1) + \gamma*0.8*U(1,2) + \gamma*0.1*U(2,1) + \gamma*0.1*U(1,1)$

The Utility of a State

If we choose action *a* at state s, expected future rewards (discounted) are:

$$U_a(s) = R(s) + \gamma \sum_{s'} T(s,a,s')U(s')$$

Reward at current state s

Probability of moving from state s to state s' by doing action a

Expected sum of total discounted rewards starting at state s by taking action a

Expected sum of future discounted rewards starting at state s'

The Utility of a State

- In the previous example, we define the utility assuming that action *a* is taken at state s
- What we want is the utility of a state s assuming that we can choose the optimal action to take at state s.
- We modify the previous formula slightly by adding a max term over actions.

$$U_a(s) = R(s) + \gamma \sum_{s'} T(s,a,s')U(s')$$

$$\Downarrow$$

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$$

The Utility of a State

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming the agent chooses the optimal action

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

This is called the Bellman Equation

The Optimal Policy

- Selection of the action π*(s) = a which maximizes the expected utility U(s')

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')U(s')$$

- Intuitively, π* gives us the best action we can take from any state to maximize our future discounted rewards

What we have seen so far

- How to formulate a problem as an MDP
- What the Markov property is
- The definition of the state utility in an MDP
- The Bellman equation

Next, we will see how we can solve the bellman equation, which will give us the optimal policy

The Bellman Equation

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

- The Bellman equations define the utility of the states
- If there are n states, there are n Bellman Equations to solve
- This is a system of simultaneous equations
- But the equations are nonlinear because of the max operator

An Iterative Solution

Define $U_1(s)$ to be the utility if the agent is at state s and lives for 1 time step

$$U_1(s) = R(s)$$

Calculate this for all states *s*

Define $U_2(s)$ to be the utility if the agent is at state *s* and lives for 2 time steps

$$U_2(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U_1(s')$$

This has already been calculated above

The Bellman Update

More generally, we have:

$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')U_i(s')$$

- This is the maximum possible expected sum of discounted rewards (ie. the utility) if the agent is at state *s* and lives for i+1 time steps
- This equation is called the Bellman Update

The Bellman Update

- As the number of iterations goes to infinity, $U_{i+1}(s)$ converges to an equilibrium value $U^*(s)$.
- The final utility values $U^*(s)$ are the solutions to the Bellman equations. Even better, they are the unique solutions and the corresponding policy is optimal
- This algorithm is called ***Value-Iteration***
- The optimal policy is given by:

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')U^*(s')$$

The Value Iteration Algorithm

$$U_1(s) = R(s)$$

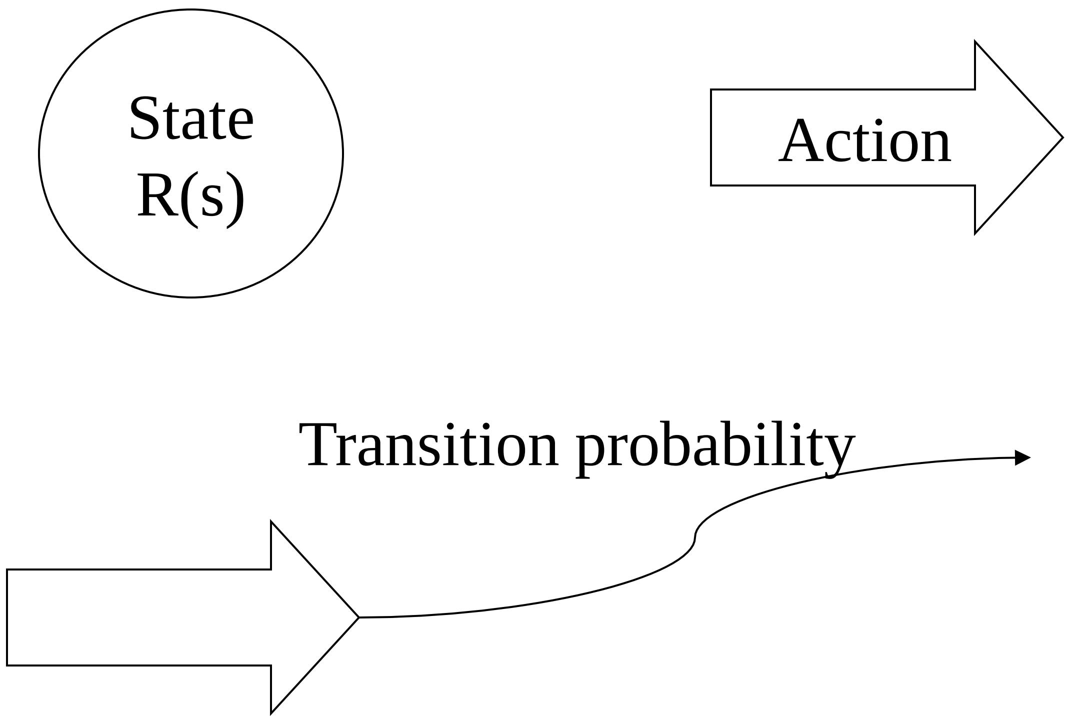
$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s')$$

1. Apply bellman update until it the utility function converges (to $U^*(s)$).
2. The optimal policy is given by:

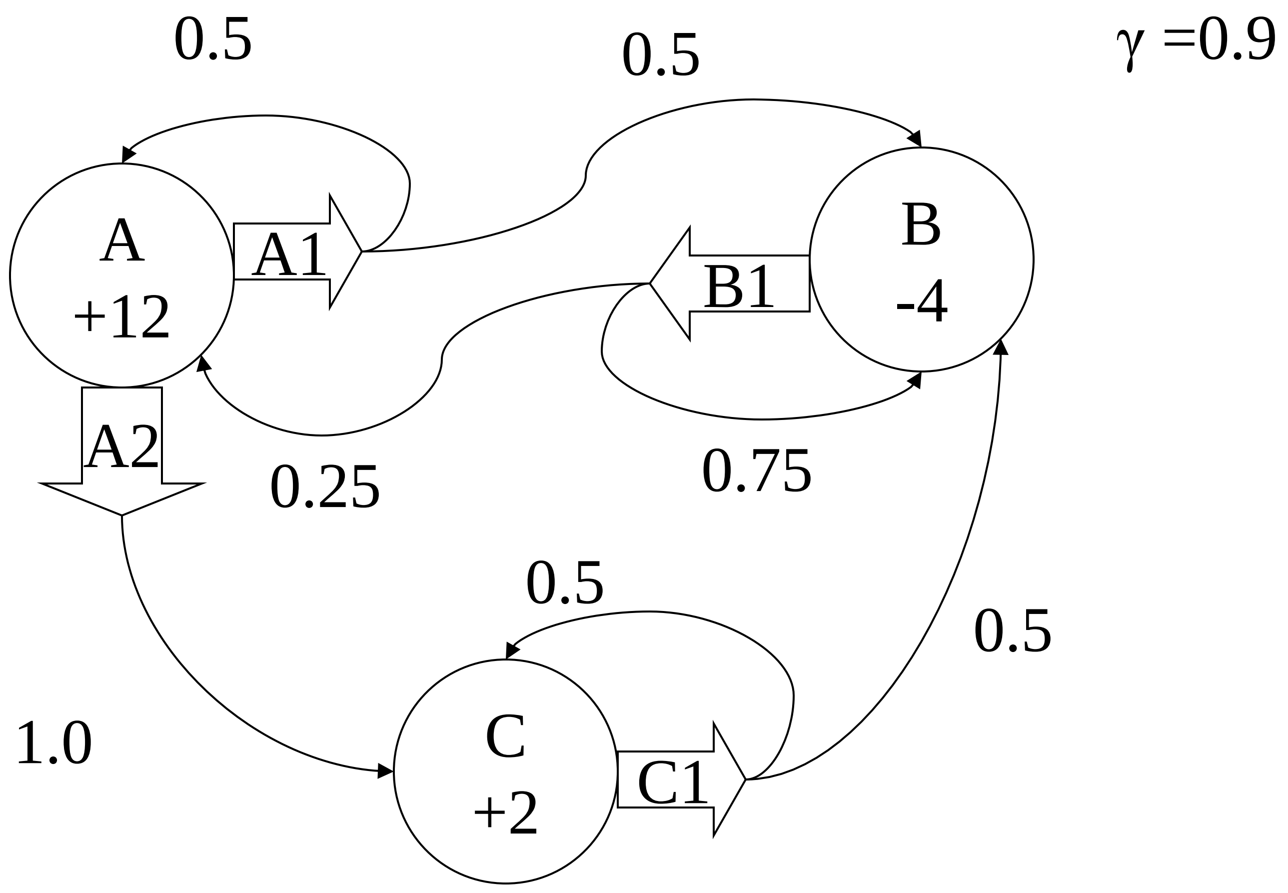
$$\pi^*(s) = \arg\max_a \sum_{s'} T(s, a, s') U^*(s')$$

Example

- We will use the following convention when drawing MDPs graphically:



Example



Example

$i=1$

$U_1(A) = R(A)=12$

$U_1(B) = R(B)=-4$

$U_1(C) = R(C)=2$

Example

$U_1(A)$	$U_1(B)$	$U_1(C)$
12	-4	2

i=2

$U_2(A) = 12 + (0.9) * \max\{(0.5)(12)+(0.5)(-4), (1.0)(2)\}$
$= 12 + (0.9)*\max\{4.0,2.0\} = 12 + 3.6 = 15.6$

$U_2(B) = -4 + (0.9) * \{(0.25)(12)+(0.75)(-4)\} = -4 + (0.9)*0 = -4$

$U_2(C) = 2 + (0.9) * \{(0.5)(2)+(0.5)(-4)\} = 2 + (0.9)*(-1) = 2-0.9 = 1.1$

Example

$U_2(A)$	$U_2(B)$	$U_2(C)$
15.6	-4	1.1

i=3

$U_3(A) = 12 + (0.9) * \max\{(0.5)(15.6)+(0.5)(-4),(1.0)(1.1)\} = 12 + (0.9) * \max\{5.8,1.1\} = 12 + (0.9)(5.8) = 17.22$

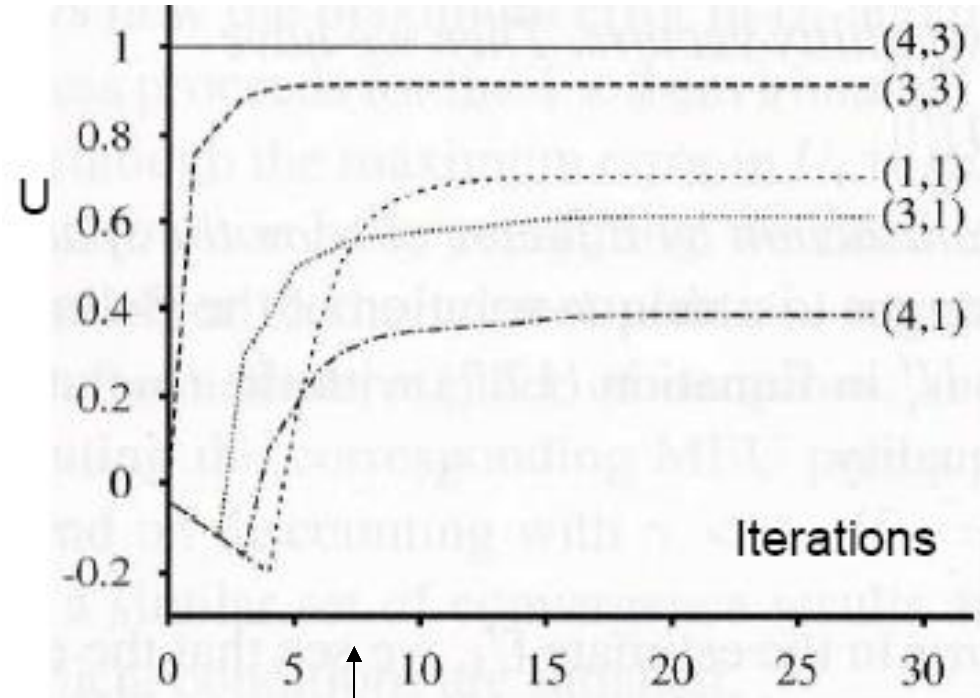
$U_3(B) = -4 + (0.9) * \{(0.25)(15.6)+(0.75)(-4)\} = -4 + (0.9)*(3.9-3) = -4 + (0.9)(0.9) = -3.19$

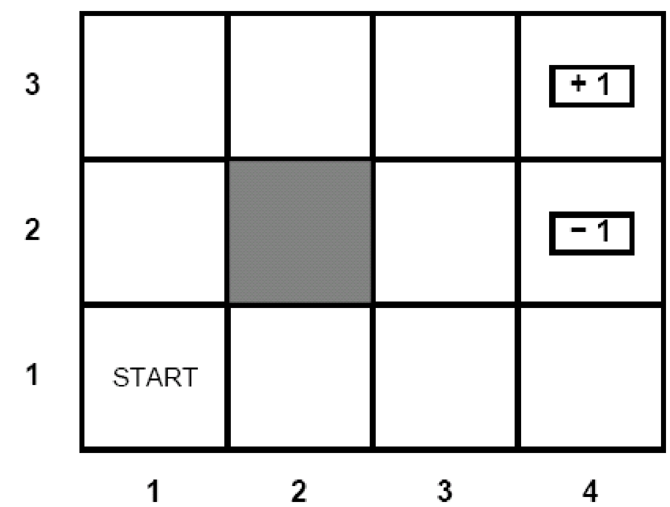
$U_3(C) = 2 + (0.9) * \{(0.5)(1.1)+(0.5)(-4)\} = 2 + (0.9)*\{0.55-2.0\} = 2 + (0.9)(-1.45) = 0.695$

The Bellman Update

- What exactly is going on?
- Think of each Bellman update as an update of each local state
- If we do enough local updates, we end up propagating information throughout the state space

Value Iteration on the Maze





Notice that rewards are negative until a path to (4,3) is found, resulting in an increase in U

Value-Iteration Termination

When do you stop?

In an iteration over all the states, keep track of the maximum change in utility of any state (call this δ)

When δ is less than some pre-defined threshold, stop

This will give us an approximation to the true utilities, we can act greedily based on the approximated state utilities

Comments

Value iteration is designed around the idea of the utilities of the states

The computational difficulty comes from the max operation in the bellman equation

Instead of computing the general utility of a state (assuming acting optimally), a much easier quantity to compute is the utility of a state assuming a policy

Utility of a policy at state s

- $U_\pi(s)$: the utility of policy π at state s
- $U^*(s)$ can be considered as $U_{\pi^*}(s)$ where π^* is an optimal policy
- Given a fixed policy, can compute its utility at state ***s*** as follows:

$$U_\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') \cdot U_\pi(s')$$

Note the difference from:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

Improving a Policy

Once we compute the utilities, we can easily improve the current policy by one-step look-ahead:

$$\pi'(s) = \arg\max_{a} \sum_{s'} T(s,a,s')U_{\pi}(s')$$

This suggests a different approach for finding optimal policy

Policy Iteration

- Start with a randomly chosen initial policy π_0
- Iterate until no change in utilities:
  1. Policy evaluation: given a policy π_i, calculate the utility $U_i(s)$ of every state s using policy π_i
  2. Policy improvement: calculate the new policy π_{i+1} using one-step look-ahead based on $U_i(s)$ ie.

$$\pi_{i+1}(s) = \arg\max_{a} \sum_{s'} T(s,a,s')U_i(s')$$

Policy Evaluation

- Policy improvement is straightforward
- Policy evaluation requires a simpler version of the Bellman equation
- Compute $U_i(s)$ for every state s using $\pi_{i:}$

$$U_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s')$$

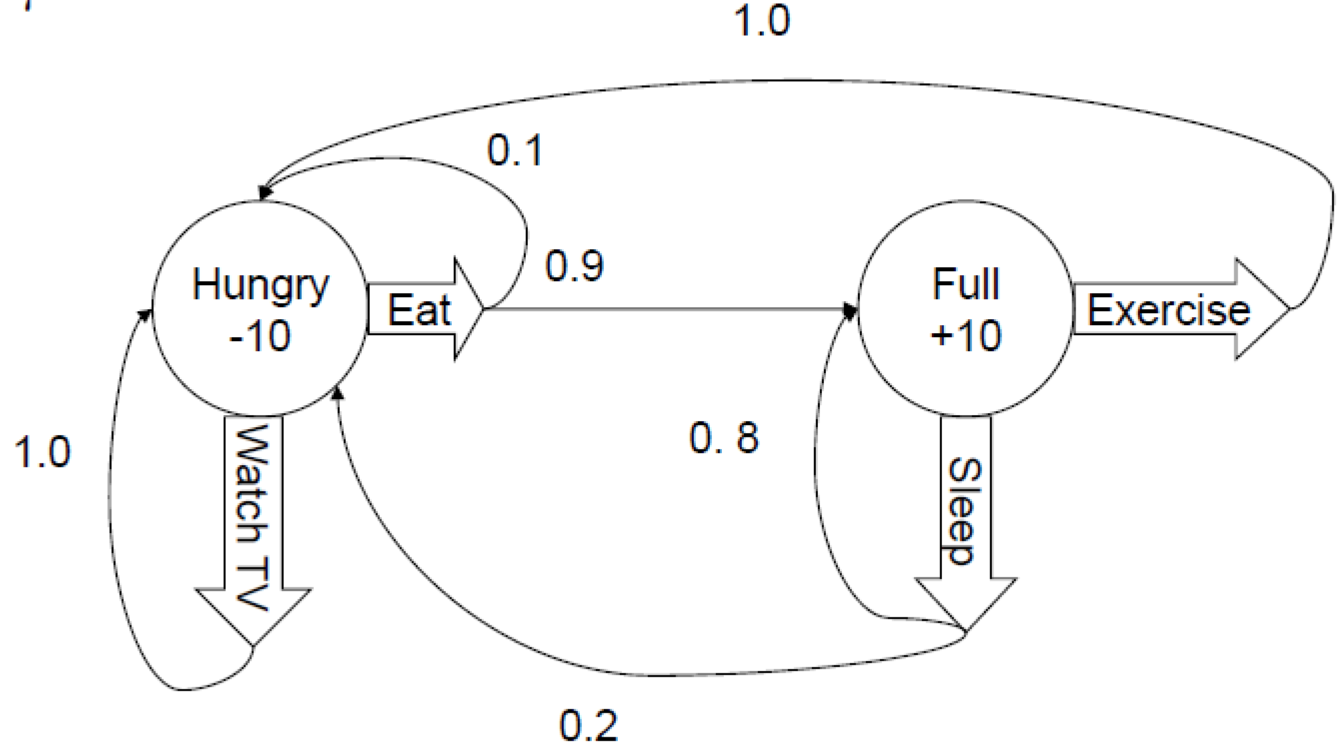
Notice that there is no max operator, so the above equations are linear, which can be solved directly and efficiently

Policy iteration comments

- In each step of policy iteration:
  - **Policy evaluation** involves solving a set of linear equations
  - **Policy improvement**: straightforward
- Each step of policy iteration is guaranteed to strictly improve the policy at some state when improvement is possible
- Converge to optimal policy
- Gives exact value of optimal policy

Policy Iteration Example

Do one iteration of policy iteration on the MDP below. Assume an initial policy of $\pi_1(\text{Hungry}) = \text{Eat}$ and $\pi_1(\text{Full}) = \text{Sleep}$.
Let $\gamma = 0.9$



Comparison

- Which would you prefer, policy or value iteration?
- Depends…
  - If you have lots of actions in each state: policy iteration
  - If you have a pretty good policy to start with: policy iteration
  - If you have few actions in each state: value iteration

Limitations

- Need to represent the utility (and policy) for every state
- In real problems, the number of states may be very large
- Leads to intractably large tables
- Need to find compact ways to represent the states eg
  - Function approximation
  - Hierarchical representations
  - Memory-based representations

What you should know

- What is MDP
- How to formulate a decision problem into a MDP
- The bellman equation
- How value iteration works
- How policy iteration works
- Pros and cons of both methods
- What is the big problem with both value and policy iteration