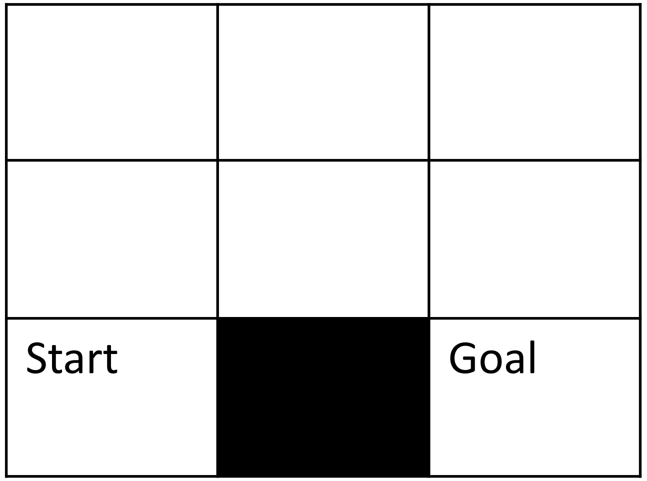
IT5005 Artificial Intelligence

Sirigina Rajendra Prasad
AY2025/2026: Semester 1

Markov Decision Processes (MDP)

Greedy best-first search - without heuristics

- Cost of each step is 1



Greedy best-first search - without heuristics

- Initialize heuristic cost with zero for all states
  - $h(s) = 0 \ \forall s$

- Assume neighbors have perfect information on cost of reaching the goal at each iteration

- Repeat the following until there is no change in $h(s)$:
  - $h(s) = \min(h(s') + c(s, s')) \ \forall s$, where $s' \in N(s)$
    - where, $N(s)$ is the set of neighbours of state s and $c(s, s')$ is the cost of moving from state s to state s'

- Select the action at each state through greedy best-first search

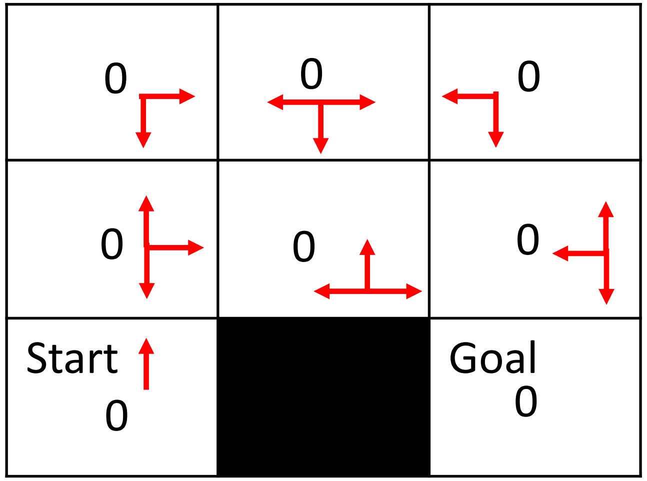
Similar to “Bellman-Ford Algorithm”

Greedy best-first search - without heuristics

- Initialize heuristic cost with zero
- Assume neighbors have perfect information on cost of reaching the goal
- Update the cost of reaching the goal from each state iteratively
    - At each iteration select the neighbor with least cost

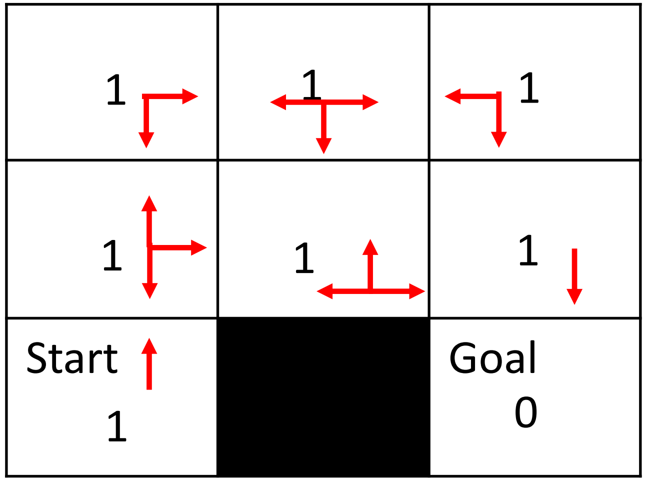
$h_k(s) = \min(h_{k-1}(s') + c(s,s'))\ \forall s$, where $s' \in N(s)$ and $c(s,s') = 1$

Initialization:

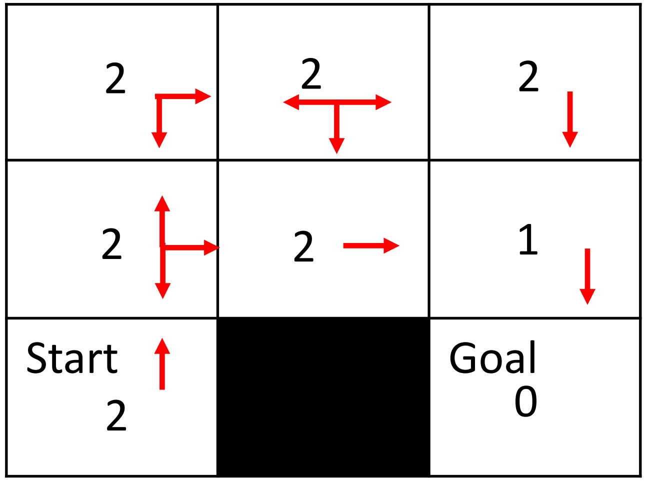


$h_0(s) = 0\ \forall s$

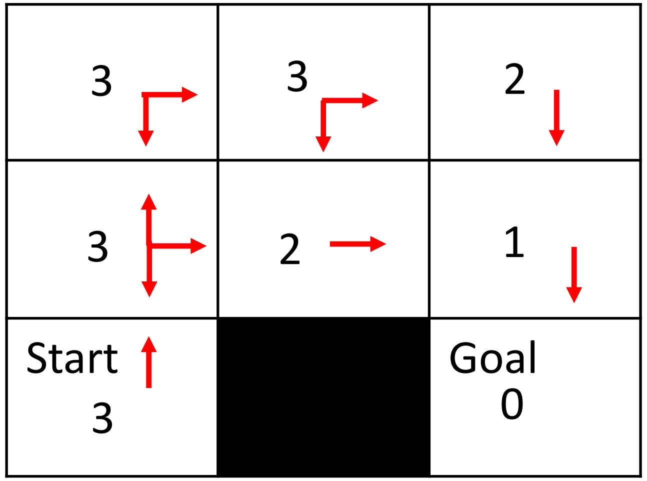
Iteration #1



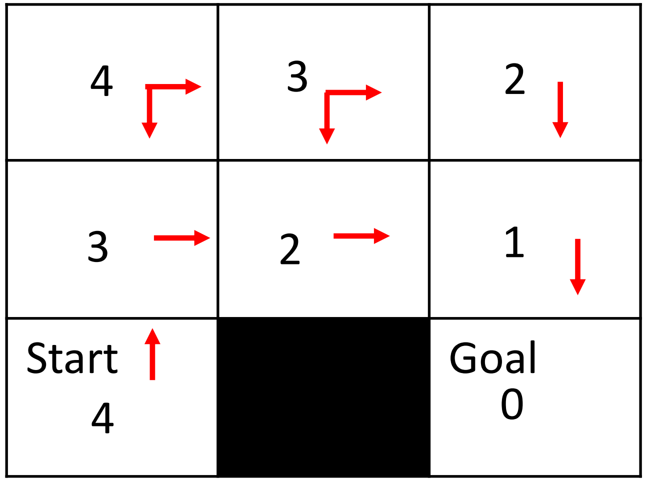
Iteration #2



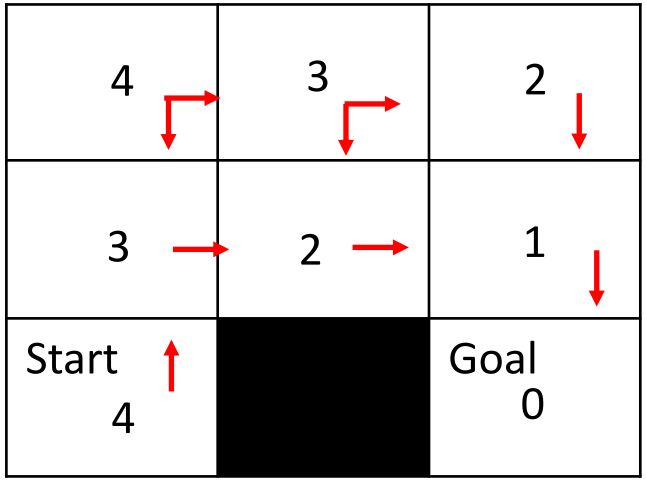
Iteration #3



Iteration #4



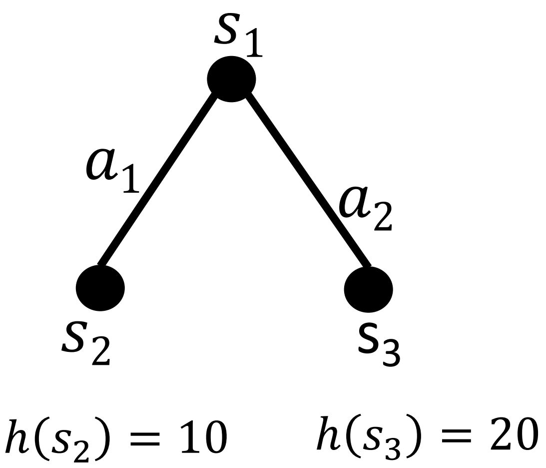
Iteration #5



No change

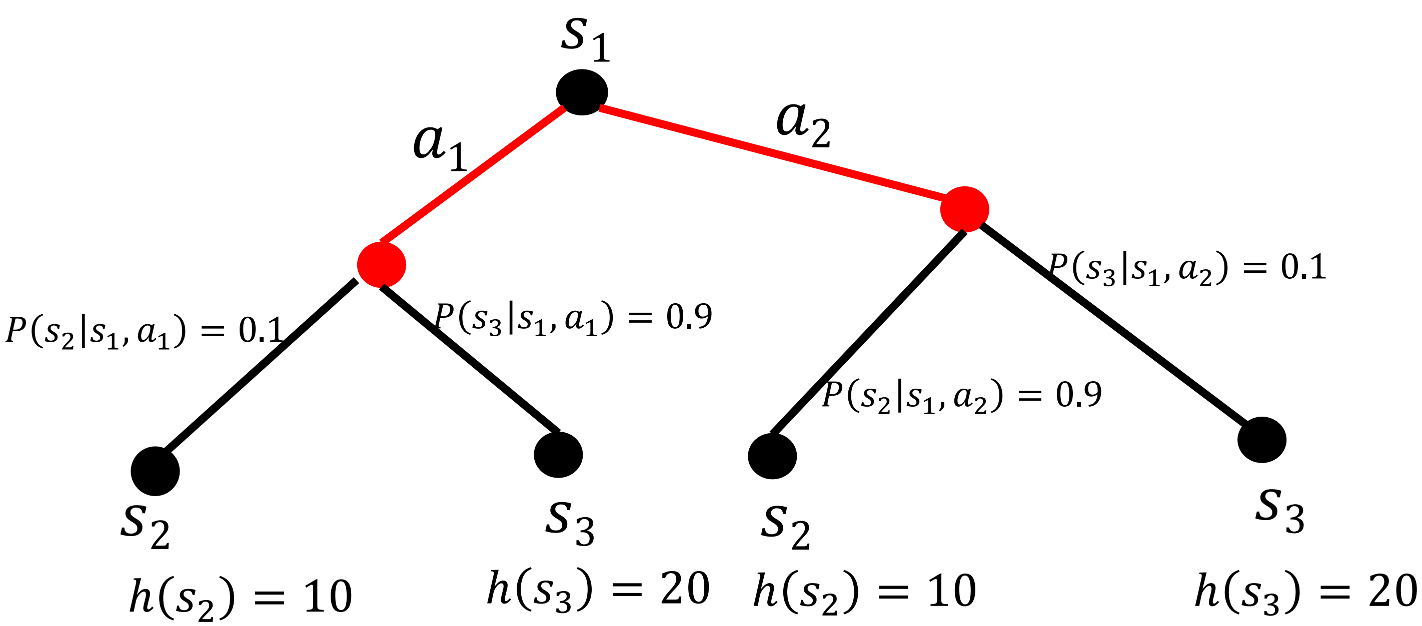
Greedy Search

without uncertainty



Select a neighbor with minimum heuristic cost

with uncertainty



Select a neighbor with minimum average heuristic cost

"expectimin"

$P(s_j|s_i, a_k)$: Probability that action a_k at state s_i leads to state s_j

Dynamic Programming (DP)

- DP can be used if a problem can be divided into subproblems and the problem has either **overlapping subproblems** or **optimal substructure**

- Overlapping subproblems
  - Stores solutions of subproblems to avoid repeated calculations
    - Ex: Fibonacci sequence

- Optimal substructure
  - Obtains optimal solution of a problem by using optimal solutions of its subproblems

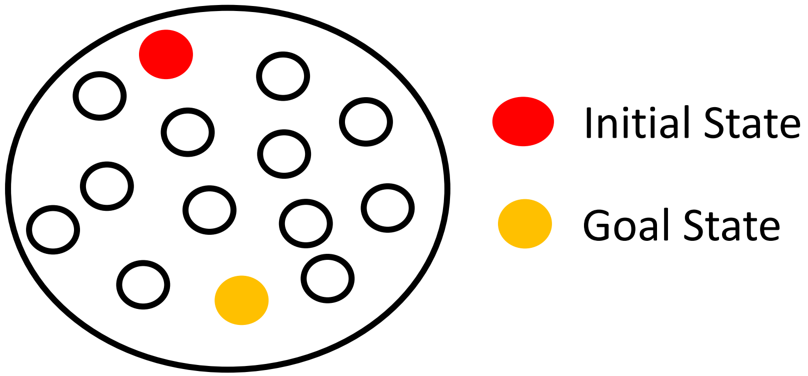
Why didn’t we consider DP for search?

Not suitable for state spaces with large number of states

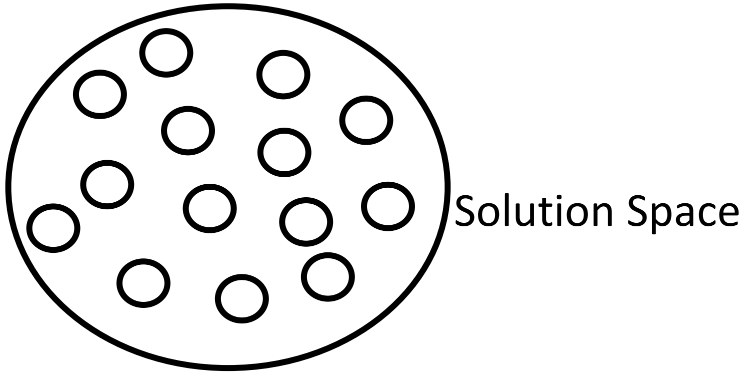
So far: (1) Reasoning in State Space

- Based on Atomic Representation
- Uninformed and Informed Search
- a sequence of actions that take the agent from initial state to goal state with minimal cost

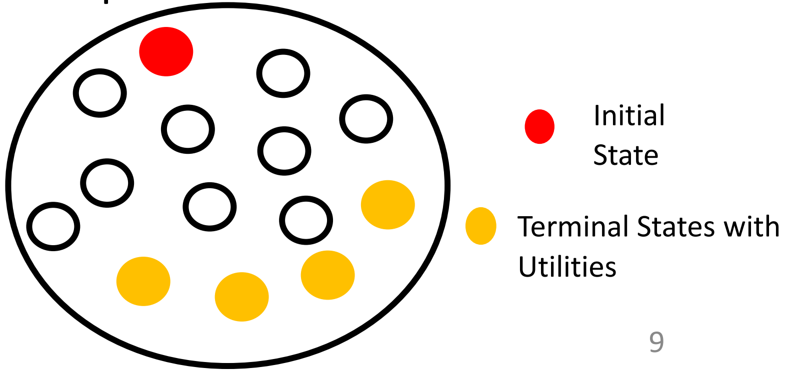
State Space for Uninformed and Informed Search



State Space for Local Search

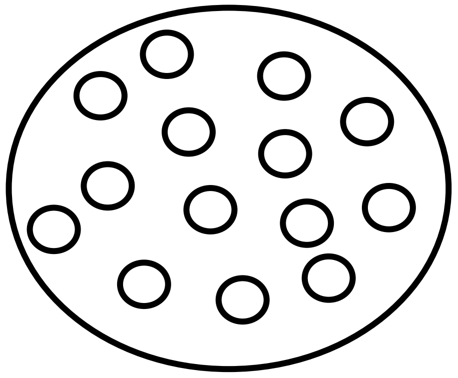


State Space for Adversarial Search



So far: (2) Reasoning in Feature Space

- Represent each state using features/variables
  - Discrete and Boolean variables
- Reason using features
  - Efficient search through
    - Consistency checking
    - Heuristics
  - Automated answering of queries using
    - Resolution Rule and Contradiction
    - Modus Ponens



So far: (3) Reasoning in Uncertain Environment

- Efficient Representation of knowledge using Bayesian Networks, Markov Chains, and Hidden Markov Models
- Identification of relevant variables via independence relations
- Answering queries using exact Inference

Markov Decision Process

Uncertainty in Action Effects

Markov Decision Process

"A sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards"

MDP Applications

- Self-driving car
  - Taking actions (steering, accelerating, breaking, etc.) to optimize performance (eg: ride quality)
- Games
  - Movement of pieces to maximize chances of winning
- Logistics planning
  - Inventory movements to minimize delays in delivery
- Humanoid robot
  - Movement planning to optimize time to complete a task
- Managing investment portfolio
  - Trades to optimize long-term investment gains
- Wireless networks
  - Scheduling and resource allocation for optimal utilization of resources

MDP

- Isn’t it an optimization framework?
  - Yes, provided the MDP model is known

- Then, why are we doing it in AI Module?
  - Foundation for reinforcement learning (RL)
  - If MDP model is unknown, agent learns the optimal actions by acting and collecting the rewards in the environment

Markov Decision Process

- An MDP is defined as:
  - A set of states
  - A set of actions $ACTIONS(s)$ for each state s
  - Transition Function - $P(s'|s,a)$
  - Reward Function $-\ R(s,a,s')$
  - Discount Factor $-\ \gamma$

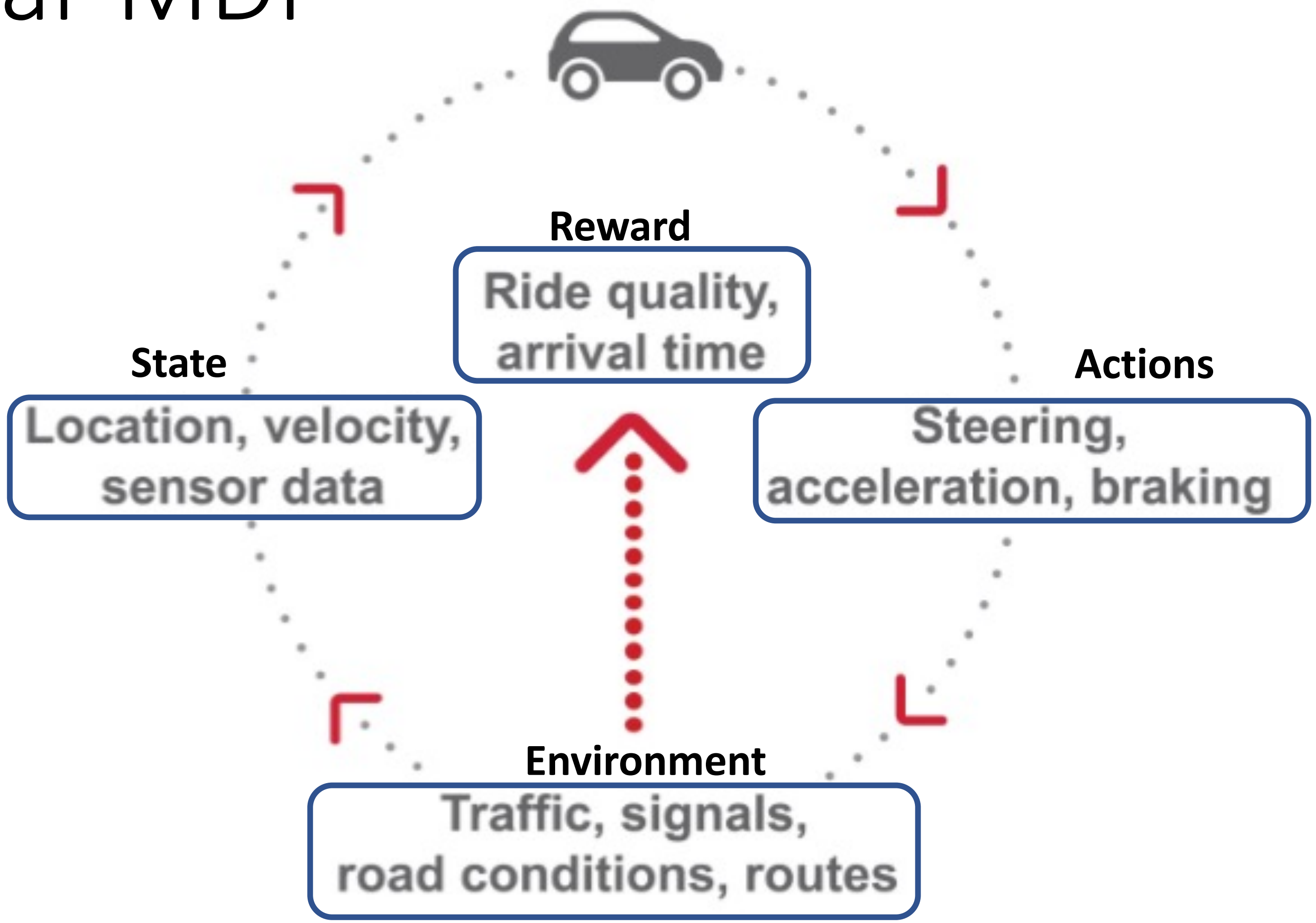
$P(s'|s,a)$: the probability of going from state s to state s' with action a
$R(s,a,s')$: Reward for the transition from state s to state s' with action a

For each state s and action a:

$$\sum_{s'} P(s'|s,a) = 1$$

Assumptions: state-based representation, finite number of states, actions and discrete time instants

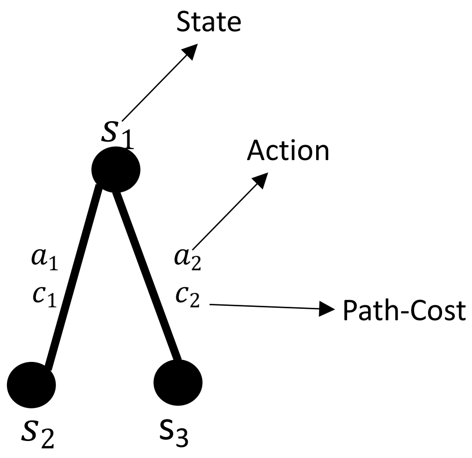
Self-driving Car MDP



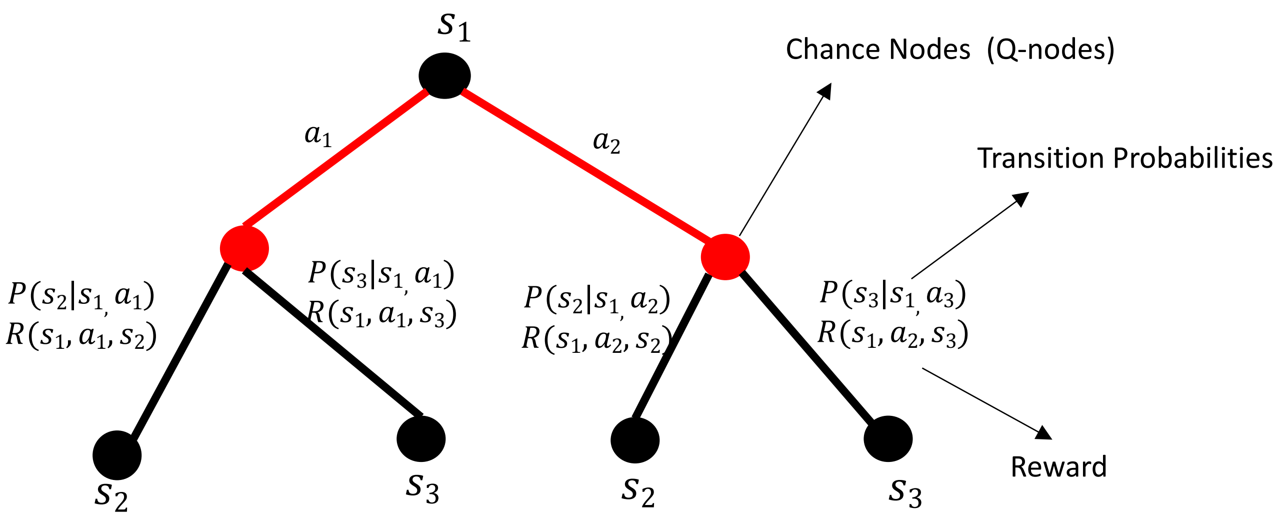
https://stanford.edu/~ashlearn/RLForFinanceBook/book.pdf

Traditional Models Vs MDPs

Traditional Model (no uncertainty)



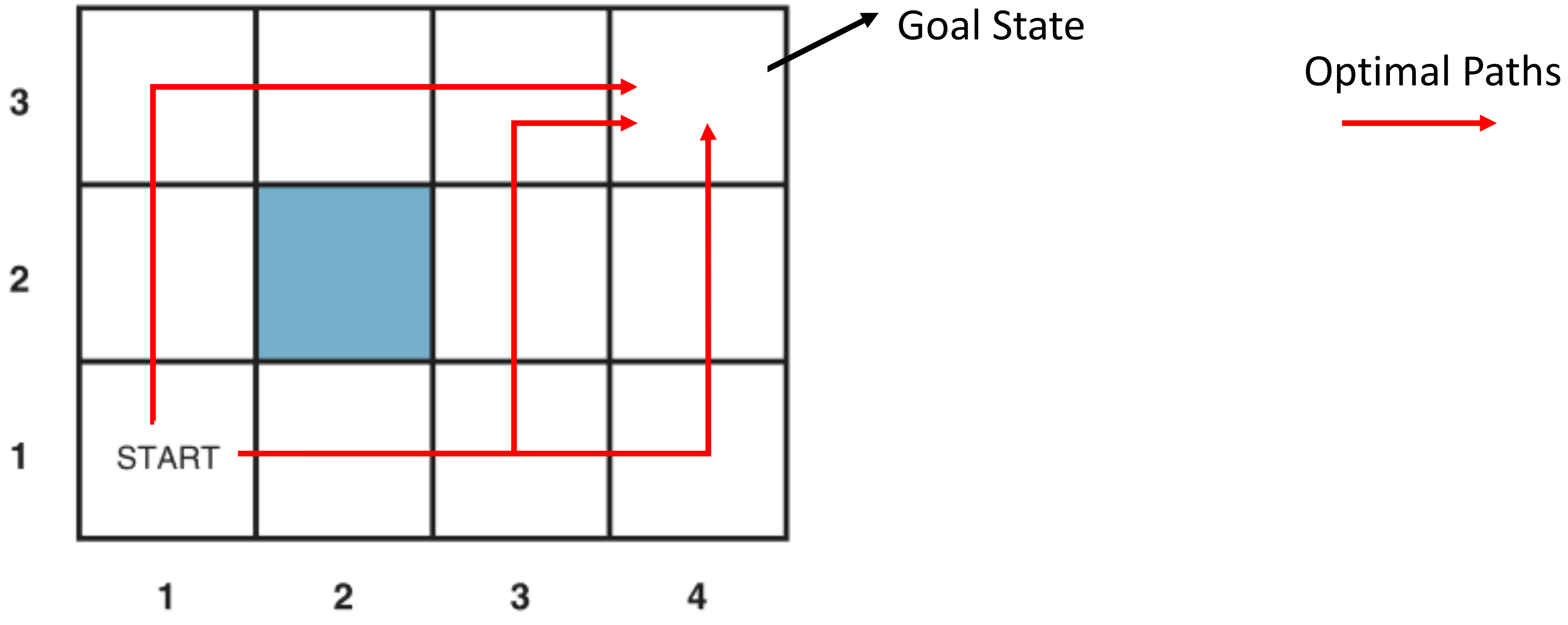
MDP (Effect Uncertainty)



Traditional Models can be treated as special case of MDPs

$$P(s_2|s_1, a_1) = P(s_3|s_1, a_2) = 1$$
$$R(s_1, a_1, s_2) = -c_1$$
$$R(s_1, a_1, s_3) = -c_2$$

Maze - Traditional Model



State Space Modeling:

Initial State: (1,1)

Actions: E, W, N, S

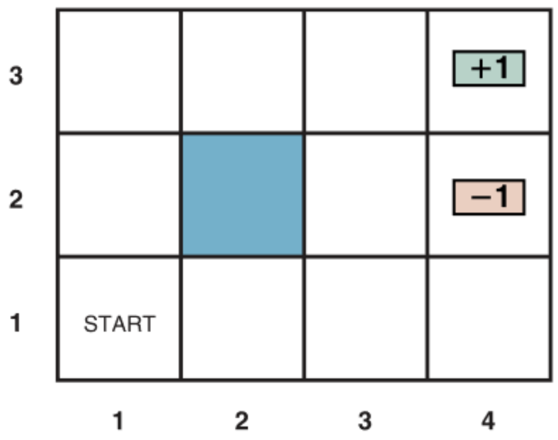
Transition Model: Example: Action E takes agent from $(1{,}1)$ to $(2{,}1)$

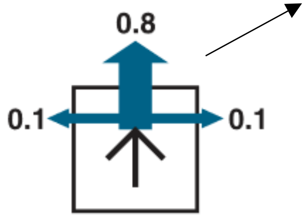
Step-Cost: 1

Goal-Test: State $(4{,}3)$

Search algorithms can find optimal solution (path with least cost)

Maze - MDP Model





Effect Uncertainty:
Desired effect with probability 0.8
Undesired effects with probability 0.2

Transition to State	Reward
Terminal State (4,3)	+1
Terminal State (4,2)	-1
Other states	-0.04

$$P((1,1)|(1,1),\rightarrow) = 0.1$$
$$P((2,1)|(1,1),\rightarrow) = 0.8$$
$$P((1,2)|(1,1),\rightarrow) = 0.1$$
$$P((1,3)|(1,1),\rightarrow) = 0$$

$$\uparrow$$

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

(Transition Function)

$$R\big((1,1),\rightarrow,(1,1)\big) = -0.04$$
$$R\big((1,1),\rightarrow,(2,1)\big) = -0.04$$
$$R\big((1,1),\rightarrow,(1,2)\big) = -0.04$$
$$R\big((3,3),\rightarrow,(4,3)\big) = +1$$
$$R\big((3,2),\rightarrow,(4,2)\big) = -1$$
$$R\big((4,1),\rightarrow,(4,2)\big) = -1$$

$$\uparrow$$

$$R(s,a,s')$$

(Reward Function)

Example:

Action $\uparrow$ at (1,1):

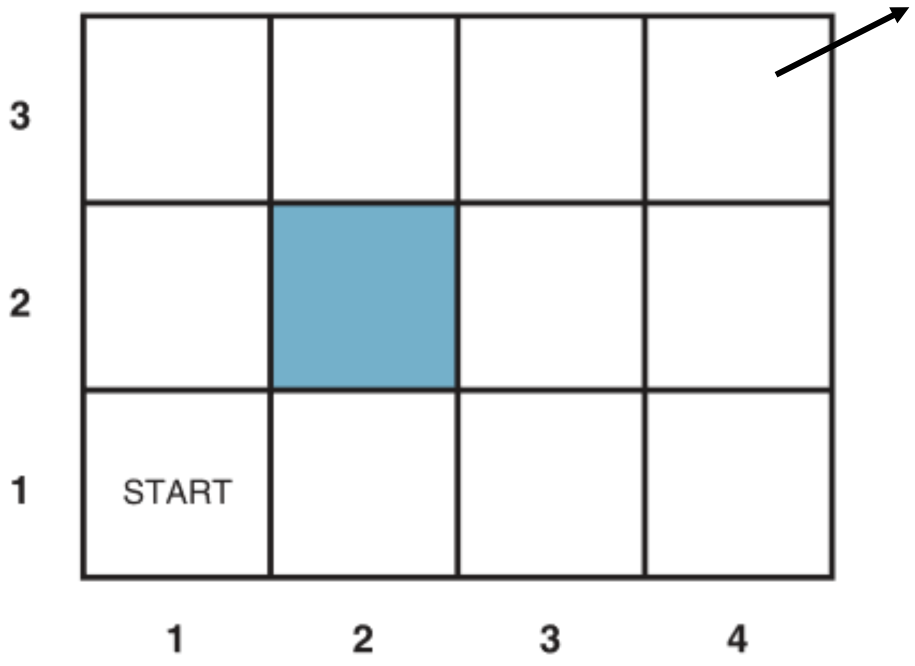
- Move to (1,2) with probability 0.8
- Stay at (1,1) with probability 0.1
- Move to (2,1) with probability 0.1

Action $\leftarrow$ at (1,1):

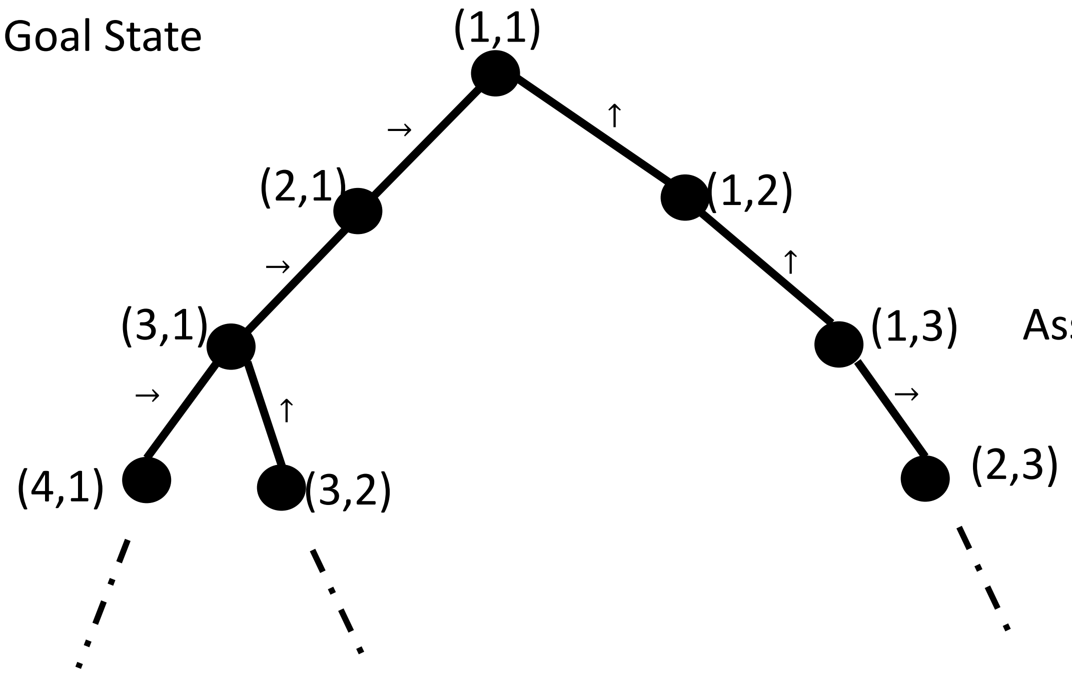
- Move to (1,2) with probability 0.1
- Stay at (1,1) with probability 0.9

Traditional Model - Search Tree

→ East (Right)
← West (Left)
↑ North (Up)
↓ South (Down)



Goal State

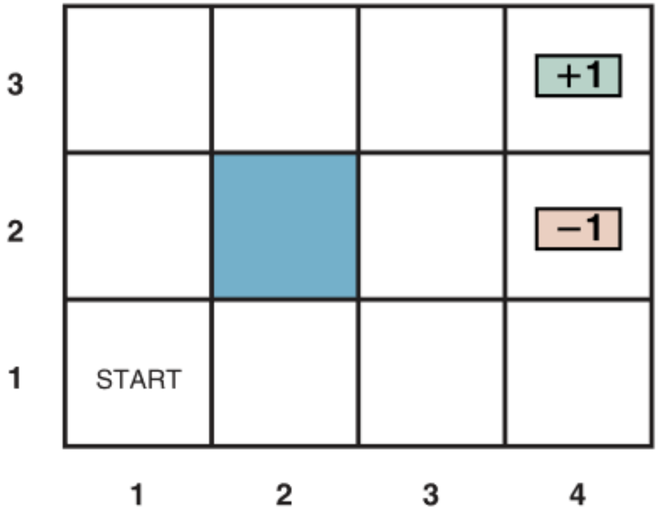


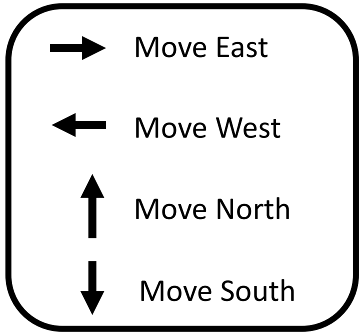
Assuming graph-search

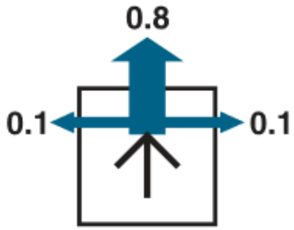
Goal-based agent:

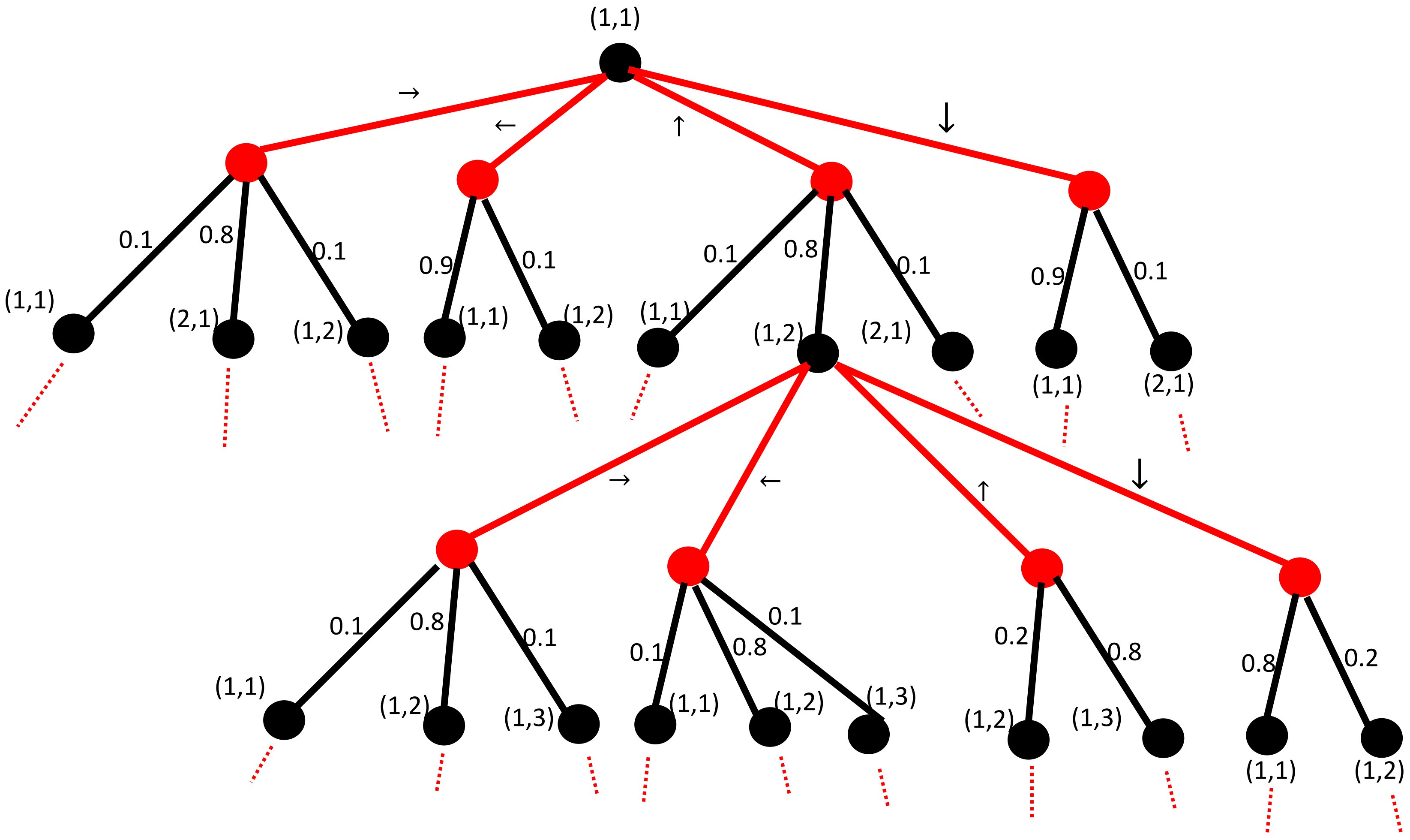
Search for a path to reach goal state (assume (4,3) is the goal state)

MDP Search Tree





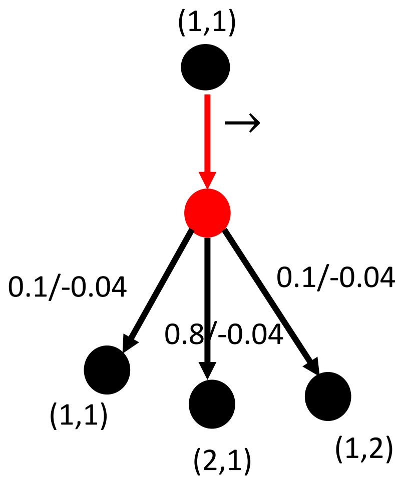


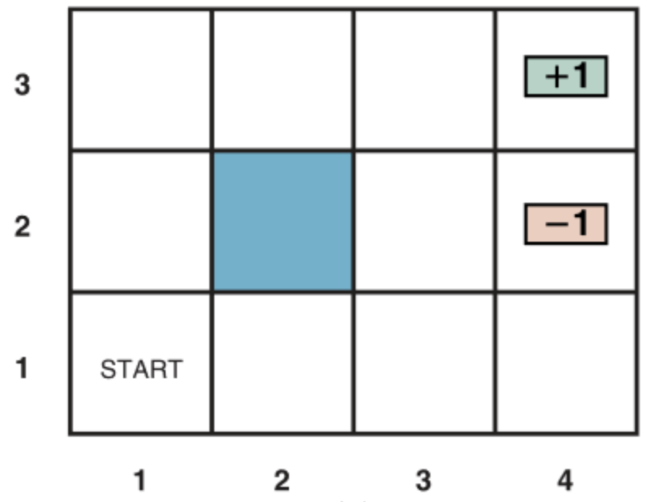


Q-State

A Closer Look at MDP Search Tree

- For state $(1,1)$ and action $\rightarrow$:





$$P((1,1)|(1,1),\rightarrow) = 0.1 \qquad R\big((1,1),\rightarrow,(1,1)\big) = -0.04$$
$$P((2,1)|(1,1),\rightarrow) = 0.8 \qquad R\big((1,1),\rightarrow,(2,1)\big) = -0.04$$
$$P((1,2)|(1,1),\rightarrow) = 0.1 \qquad R\big((1,1),\rightarrow,(1,2)\big) = -0.04$$

$$\sum_{s'=\{(1,1),(2,1),(1,2)\}} P(s'|(1,1),\rightarrow) = 1$$

MDP Terminology

- Policies
  - Policy (π)
  - Optimal Policy (π^*)

- Utilities
  - $U^{\pi}(s)$: Utility of a state s with policy π
  - $U(s)$: Optimal utility of a State

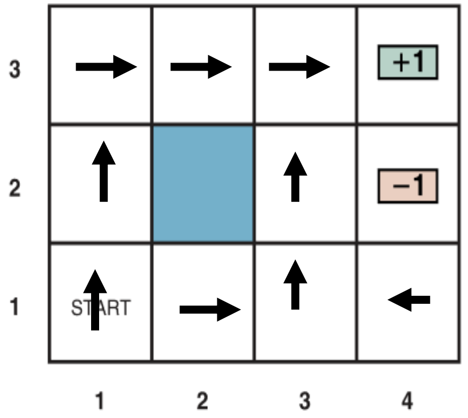
- Action-Utility Function or Q-function
  - $Q^{\pi}(s, a)$: Action-Utility value by taking action a at state s and then following policy π
  - $Q(s, a)$: Action-Utility value by taking action a at state s and then following optimal policy

Policy

- Policy $\pi(s)$ suggests an action a for each state s
  - $\pi: s \rightarrow a$

$\pi_1((1,1)) = \uparrow$
$\pi_1((2,1)) = \rightarrow$
$\pi_1((1,2)) = \uparrow$

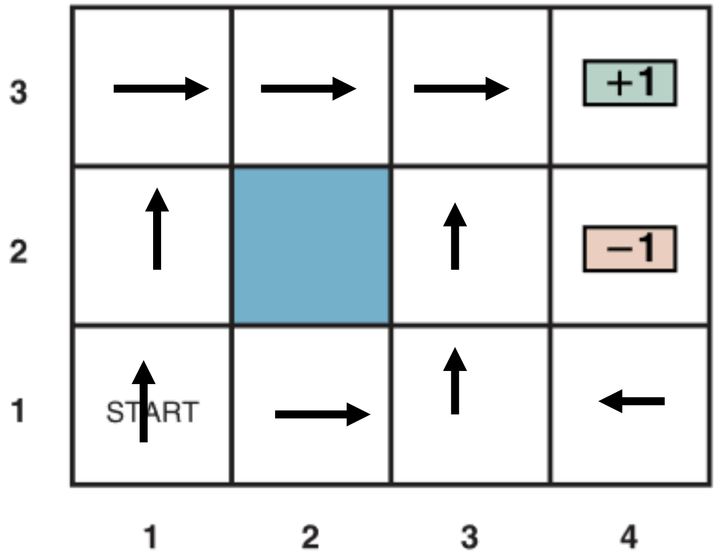
An Example: π_1



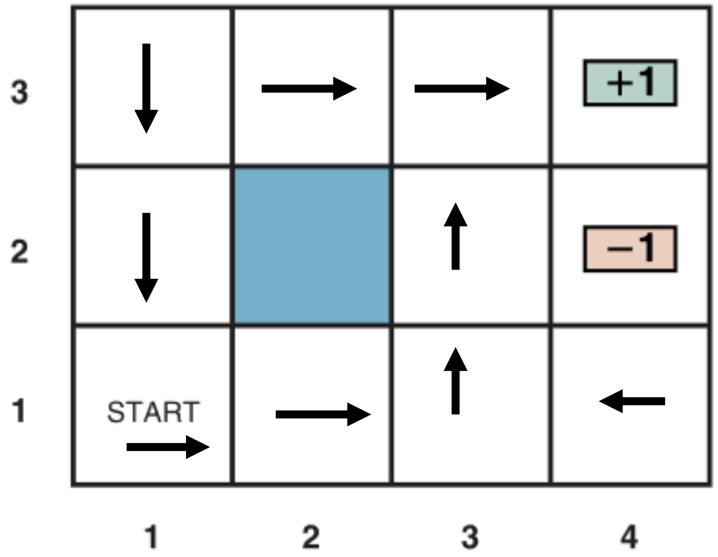
- Total number of policies: $|A|^{|S|}$
  - $|A|$ is the number of actions
  - $|S|$ is the number of states

Policy Examples

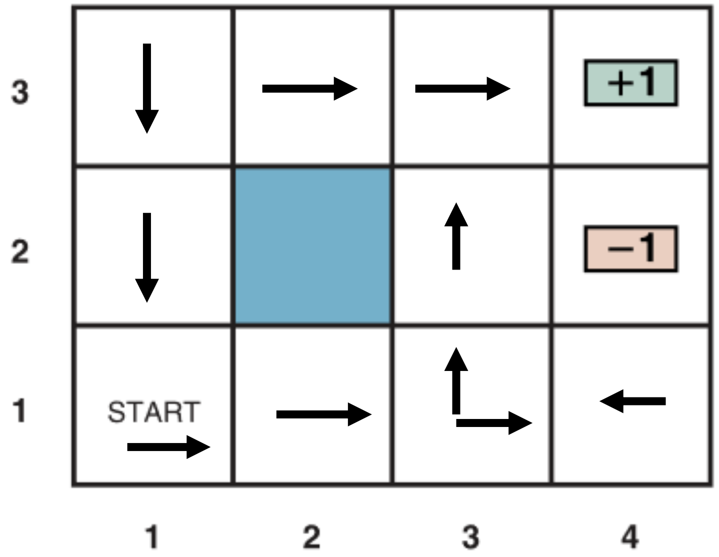
Policy 1 (π_1)

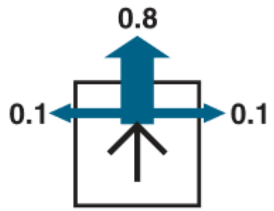


Policy 2 (π_2)



Policy 3 (π_3)





Questions:

Which policy is good (optimal)?
How to define optimality?
How to find optimal policy?

Utility of a State with a Policy π: $U^{\pi}(s)$

Expected (average) cumulative sum of rewards obtained by following the policy π from that state

Utility of a state is also called State-Value or Value of a State

Cumulative Sum of Rewards

- Sequence of states and actions with policy π from the initial state s_0

$$[s_0, \pi(s_0), s_1, \pi(s_1), \dots, \pi(s_{n-1}), s_{n,\dots}]$$

Trajectory with policy π for an episode

$\pi : s \rightarrow a$
$\pi(s_i) = a_i$

  - Sequence ends at terminal state
  - Sequence is also known as trajectory of an episode

- Cumulative sum of rewards for an episode with policy π from the initial state s_0:

$$U_h([s_0, \pi(s_0), s_1, \pi(s_1), \dots, \pi(s_{n-1}), s_n \dots]) = R(s_0, \pi(s_0), s_1) + R(s_1, \pi(s_1), s_2) + R(s_2, \pi(s_2), s_3) + \cdots$$

$$= \sum_{t=0}^{\infty} R(S_t, \pi(S_t), S_{t+1})$$

Utility with Policy π

reward at state S_t with action $\pi(S_t)$ and transition to state S_{t+1}

Cumulative Sum of Discounted Rewards

- Cumulative sum of discounted rewards for an episode:

$$U_h([s_0, \pi(s_0), s_1, \pi(s_1), \ldots., \pi(s_{n-1}), s_n \ldots]) = R(s_0, \pi(s_0), s_1) + \gamma\, R(s_1, \pi(s_1), s_2) + \gamma^2 R(s_2, \pi(s_2), s_3) + \cdots$$

$$= \sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})$$

$$0 \leq \gamma \leq 1$$

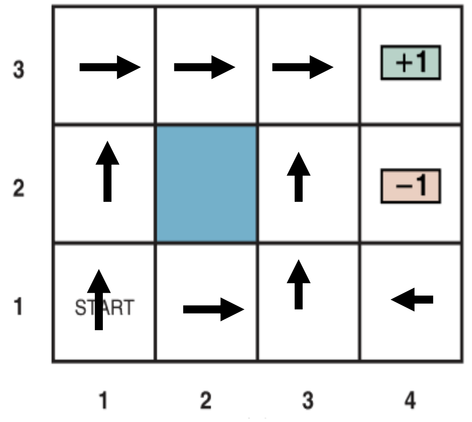
(discount factor)

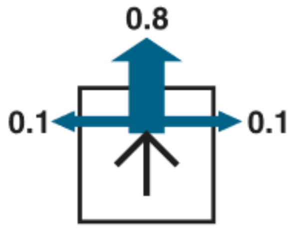
- Why discounts?
  - Guarantees convergence of iterative algorithms
  - Also guarantee convergence for infinite/indefinite horizon problems
  - Gives more weightage to immediate reward at state s_0

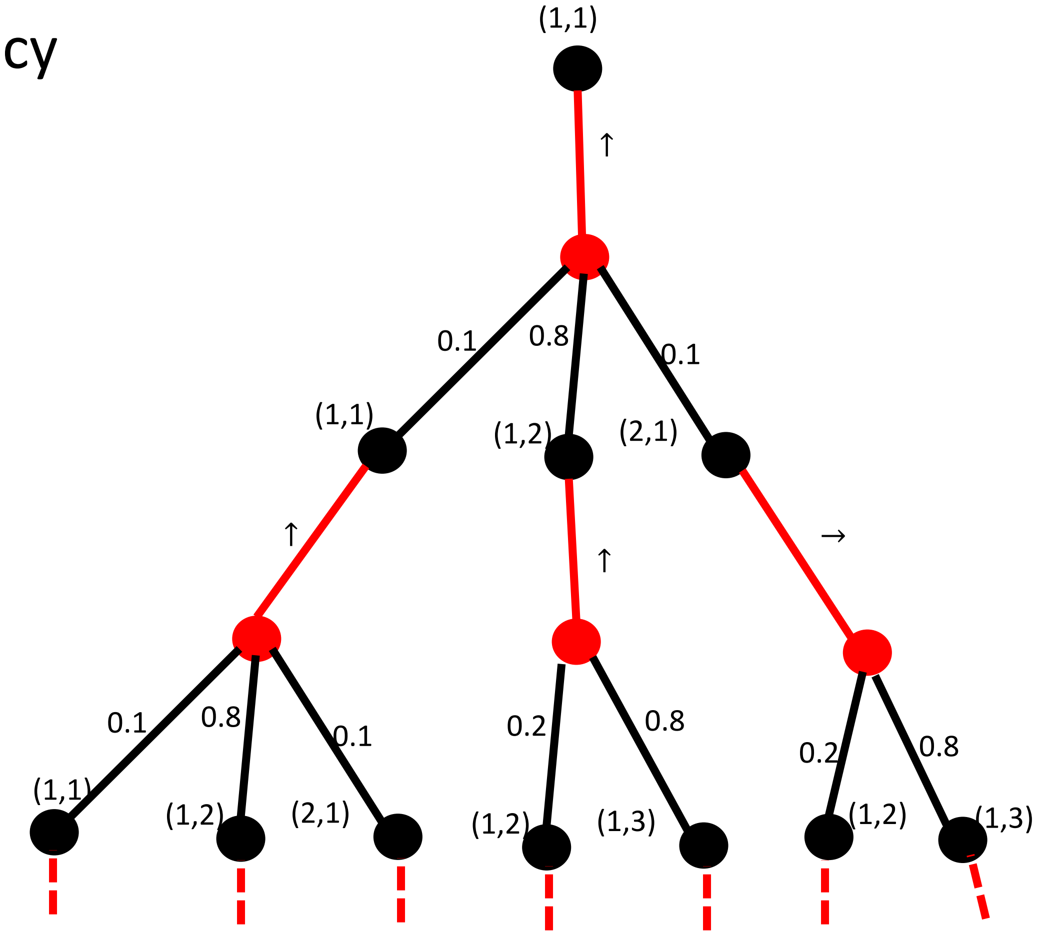
Example: Maze - Cumulative Rewards

- Trajectories are stochastic and not unique for a given policy
  - Different trajectories are possible for a given policy
- Cumulative reward is also stochastic
  - Different for each episode

Policy: π_1







Environmental History of State (1,1) with Policy π_1

- States and Actions after executing the policy π_1
  - $[s_0, \pi_1(s_0), s_1, \pi_1(s_1), \ldots., \pi_1(s_{n-1}), s_{n,} \ldots]$

- Example:
  - Environmental history of four episodes for $s_0 = (1,1)$

    1 $[(1,1), \uparrow, (1,2), \uparrow, (1,3), \rightarrow, (2,3), \rightarrow, (3,3), \rightarrow, (4,3)]$

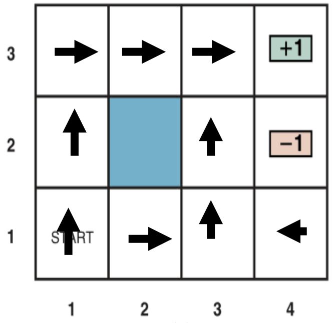
    2 $[(1,1), \uparrow, (1,1), \uparrow, (2,1), \rightarrow, (3,1), \uparrow, (3,2), \uparrow, (3,3) \rightarrow, (4,3)]$

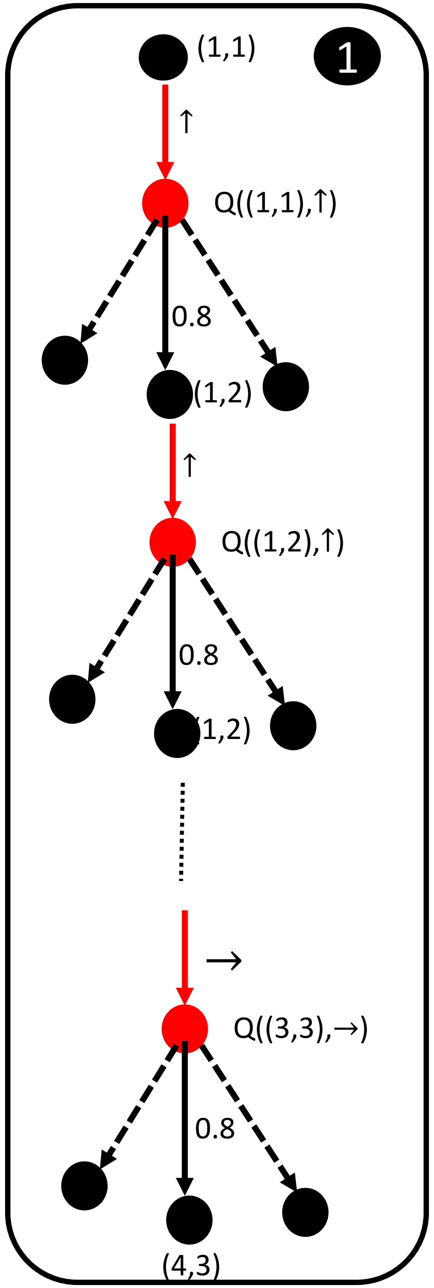
    3 $[(1,1), \uparrow, (1,1), \uparrow, (1,2), \uparrow, (1,3), \rightarrow, (2,3), \rightarrow, (3,3), \rightarrow, (4,3)]$

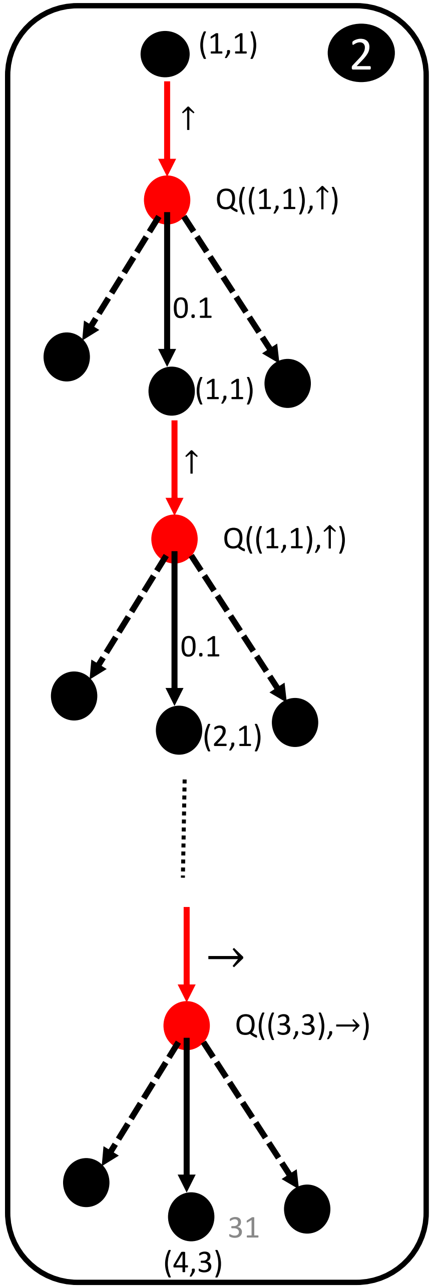
    4 $[(1,1), \uparrow, (1,1), \uparrow, (2,1), \rightarrow, (3,1), \uparrow, (3,2), \uparrow, (4,2)]$

  - Each episode has a different trajectory with same policy π_1

Policy: π_1

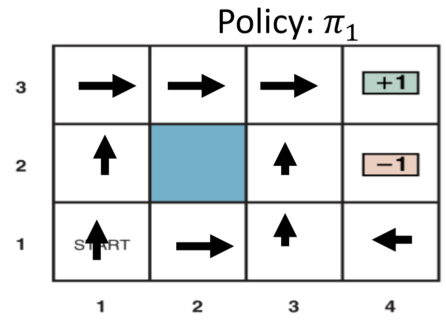




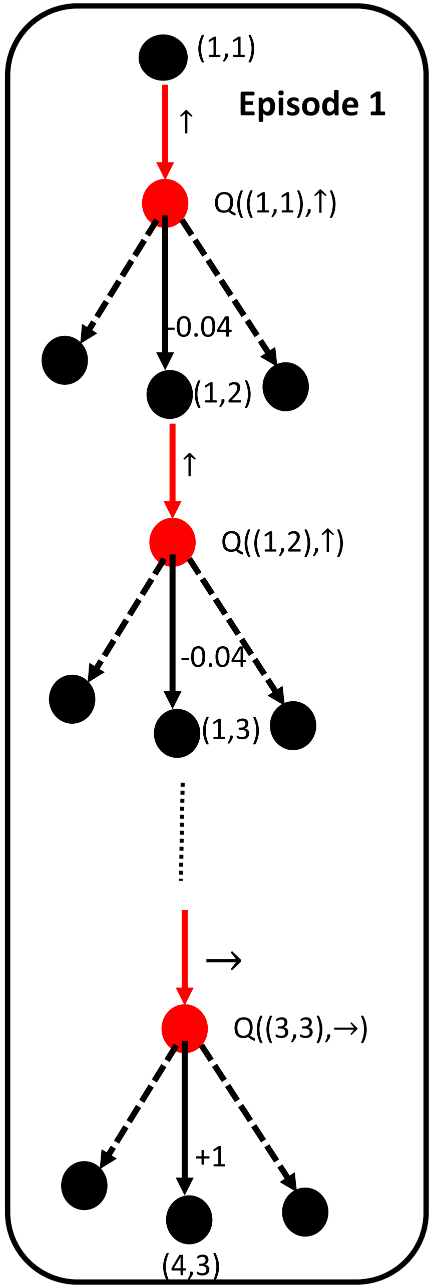


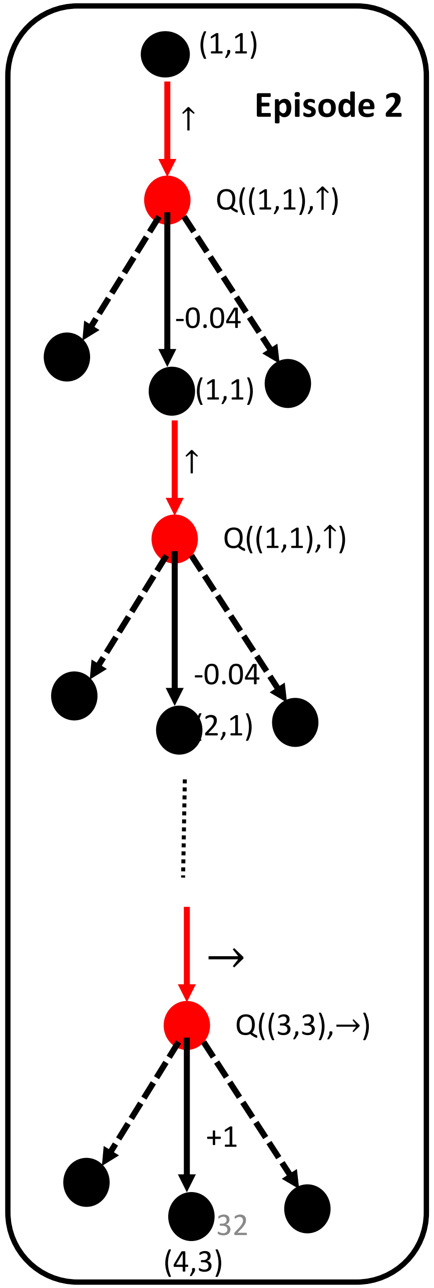
Cumulative Rewards of State (1,1) with Policy π_1

- States and Actions after executing the policy π_1
  - $[s_0, \pi_1(s_0), s_1, \pi_1(s_1), \ldots, \pi_1(s_{n-1}), s_n, \ldots]$



- Examples:
  - Episode 1
    - Trajectory: $[(1,1), \uparrow, (1,2), \uparrow, (1,3), \rightarrow, (2,3), \rightarrow, (3,3), \rightarrow, (4,3)]$
    - Reward: -0.04 - γ 0.04 - γ^2 0.04 - γ^3 0.04 + $\gamma^4$1
  - Episode 2
    - Trajectory: $[(1,1), \uparrow, (1,1), \uparrow, (2,1), \rightarrow, (3,1), \uparrow, (3,2), \uparrow, (3,3) \rightarrow, (4,3)]$
    - Reward: -0.04 - γ 0.04 - γ^2 0.04 - γ^3 0.04 - $\gamma^4$0.04 + γ^5 1
  - Episode 3
    - Trajectory: $[(1,1), \uparrow, (1,1), \uparrow, (1,2), \uparrow, (1,3), \rightarrow, (2,3), \rightarrow, (3,3), \rightarrow, (4,3)]$
    - Reward: -0.04 - γ 0.04 - γ^2 0.04 - γ^3 0.04 - $\gamma^4$0.04 + γ^5 1
- Cumulative rewards are random
- **Questions**:
  - What is the average cumulative reward of state (1,1) with policy π_1?
  - What are the probabilities of these rewards?





Cumulative Sum of Discounted Rewards

- A recursive representation of cumulative sum of discounted rewards

$$U_h([s_0, \pi(s_0), s_1, \pi(s_0), \ldots, \pi(s_{n-1}), s_n \ldots]) = R(s_0, \pi(s_0), s_1) + \gamma\boxed{[R(s_1, \pi(s_1), s_2) + \gamma R(s_2, \pi(s_2), s_3) + \cdots}$$

$$= R(s_0, \pi(s_0), s_1) + \gamma\boxed{U_h([s_1, \pi(s_1), \ldots, s_n, \ldots])}$$

Reward of a specific episode with initial state s_0 and policy π

It is a random value and changes with episode

We need average cumulative reward

Rewards due to transition from current state (s_0) to next state (s_1) with action $\pi(s_0)$

Cumulative sum of rewards from next state (s_1) onwards if the policy π is followed

Section 15.1: AIMA4e

Utility of State s_0 with a Policy π

- Expected (average) utility of a state s_0 with policy π

$$U^{\pi}(s_0) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\right] \qquad 0 \le \gamma \le 1$$

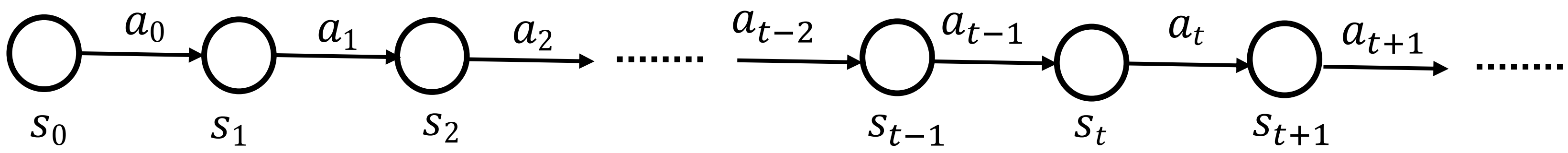
- Expectation is with respect to probability distribution of state sequences (determined by s_0 and π)
  - Infinite dimensional expectation
  - Markov assumption simplifies the above expectation

Next, a small digression into Markovity

Markov Condition

- Given the current state and action, next state is independent of previous states and actions

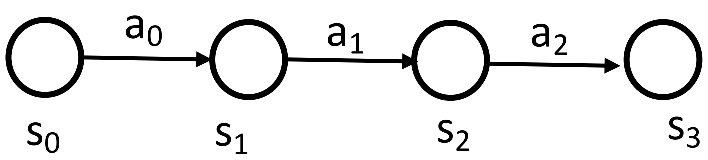
$$\Pr(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \ldots\ldots) = \Pr(s_{t+1} | s_t, a_t)$$



Markov Condition Enables….

- Factoring of joint distribution

$$\Pr(s_0, a_{0,} s_1, a_1, s_2, a_2, s_3) = \Pr(s_3|s_2, a_2) \Pr(s_2|s_1, a_1) \Pr(s_1|s_0, a_0) \Pr(s_0)$$



- Decomposition of complex problem into small sub-problems

Deterministic Policy:
$\Pr(a_i|s_i)$ = 1

Expected Utility of State s_0 with a Policy π: $U^{\pi}(s_0)$

$$U^{\pi}(s_0) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\right]$$

$$\Big\Downarrow \quad \text{Only if Markov conditions are satisfied}$$

$$\boxed{U^{\pi}(s_0) = \sum_{s'} P(s' | \pi(s_0), s_0)[R(s_0, \pi(s_0), s') + \gamma U^{\pi}(s')]}$$

Proof: last slide

Bellman Update Equation for Utility

- Expected utility of state s with a policy π

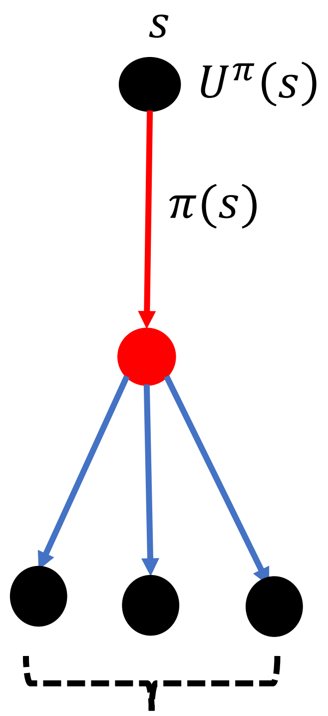
Probability of moving to state s' with action $\pi(s)$ at state s

Utility of state s' with policy π

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

Utility of state s with policy π

Reward for taking an action $\pi(s)$ at state s and moving to state s'



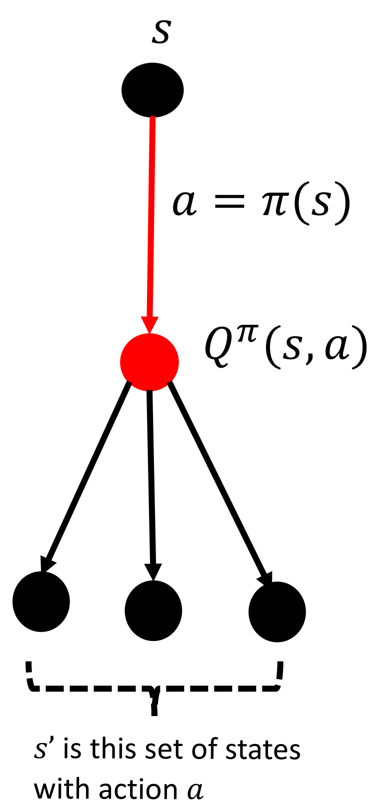
s' is this set of states

Utility of a state is expressed in terms of utility of neighbors

Action-Utility or Q-Value

- Q value indicates expected value of an action at a state

- $Q^{\pi}(s, a)$: expected utility of state s with an action a at state s, and then subsequently following the policy π



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

where $a = \pi(s)$

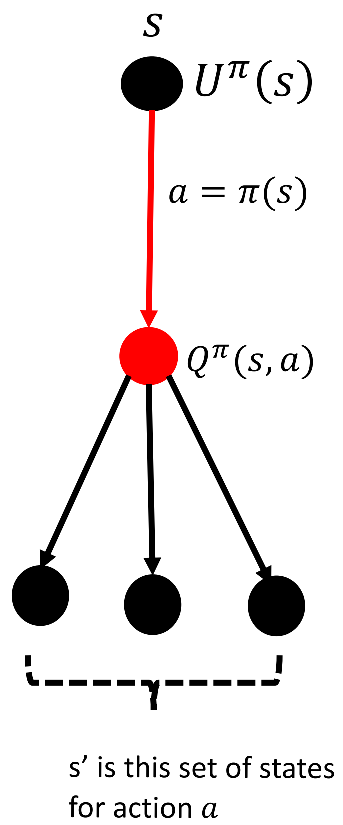
Relation between $Q^{\pi}(s,a)$ and $U^{\pi}(s)$

- If the policy π is fixed, utility is same as action-utility

$$\boxed{U^{\pi}(s) = Q^{\pi}(s,a)}$$

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

$$Q^{\pi}(s,a) = \sum_{s'} P(s'|a,s_1)[R(s,a,s') + \gamma U^{\pi}(s')]$$



Deterministic policy ➔ Action fixed for each state ➔ Utility of state is same as utility of state-action

Bellman Update Equation for Action-Utility

Q-value in terms of Q-values of neighbors

$$Q^{\pi}(s,a) = \sum_{s'} P(s'|a,s)[R(s,a,s') + \gamma Q^{\pi}(s',a')]$$

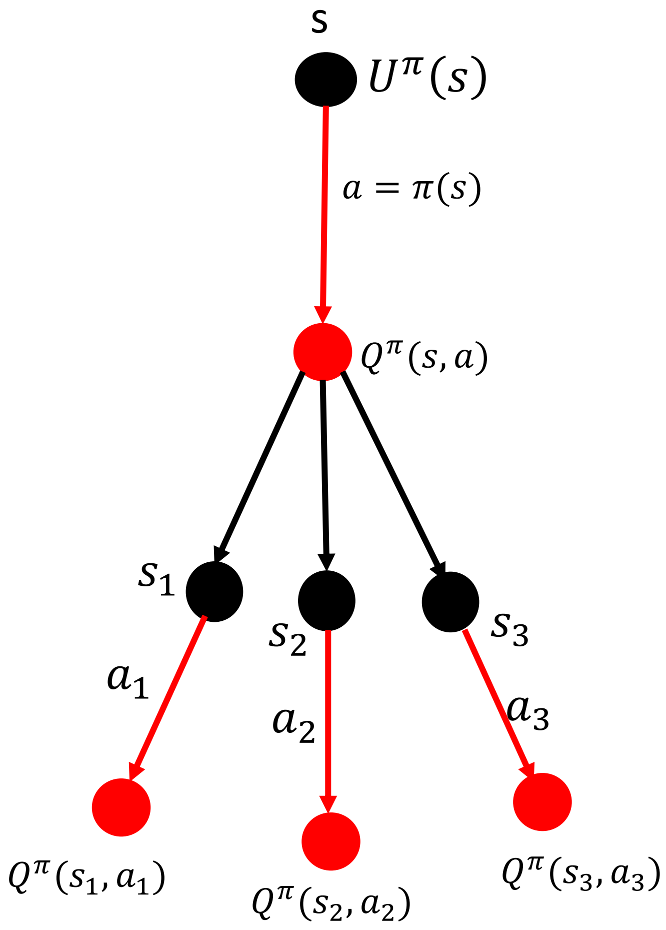
$U^{\pi}(s') = Q^{\pi}(s',a')$

$a = \pi(s)$

$a' = \pi(s')$

$s' = \{s_1, s_2, s_3\}$

$a' = \{a_1, a_2, a_3\}$



U-value vs Q-value

- $U(s)$ indicates utility or value or desirability of a state s

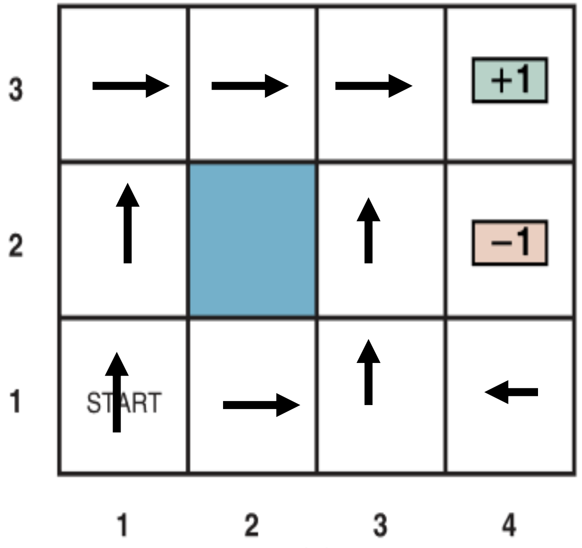
- $Q(s, a)$ indicates value of action a at state s
  - Can be used to select optimal action at state s
    - Find value of all actions at a state
    - Select action with largest Q-value
  - In other words, Q-value is used to make optimal decision

- Q-learning is one of the paradigms in reinforcement learning (RL)
  - Q-learning involves finding Q-values when model (MDP) parameters are unknown
  - Q-values implicitly stores optimal policy, i.e., optimal Q-values reveal optimal policy

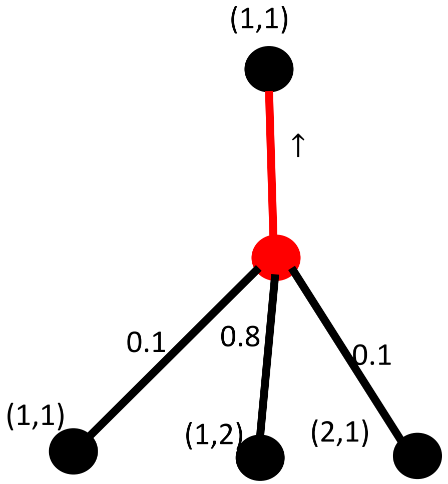
Maze: Expected Utility of state (1,1) with policy π_1

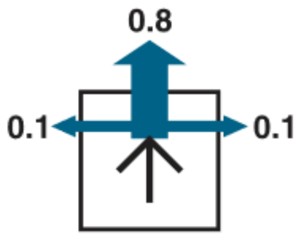
Policy: π_1

Policy π_1



$$U^{\pi_1}((1,1)) = 0.1[-0.04 + \gamma U^{\pi_1}((1,1))] + 0.8[-0.04 + \gamma U^{\pi_1}((1,2))] + 0.1[-0.04 + \gamma U^{\pi_1}((2,1))]$$

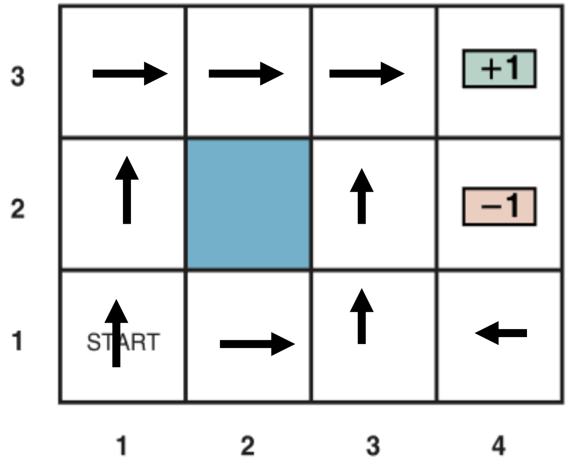




$$U^{\pi}(s) = \sum_{s'} P(s'|\pi(s), s)[R(s, \pi(s), s') + \gamma U^{\pi}(s')]$$

Maze: Bellman Update Equations

Policy: π_1



- We can get similar equations for other states:
- Example:

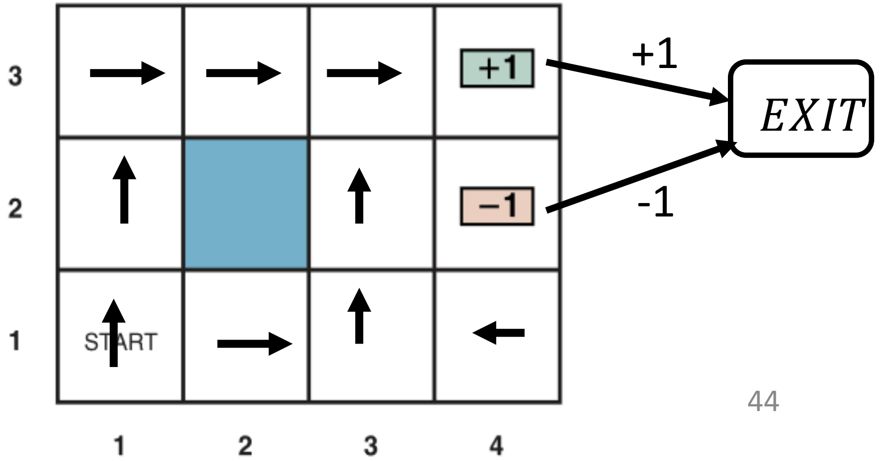
$$U^{\pi_1}((1,2)) = 0.2[-0.04 + \gamma U^{\pi_1}((1,2))] + \ 0.8[-0.04 + \gamma U^{\pi_1}((1,3)$$

Utilities of terminal states*:

$$U^{\pi_1}((4,3)) = 0$$
$$U^{\pi_1}((4,2)) = 0$$

$$\forall a \ P\big((4,2)|(4,2), a\ \big) = 1, \mathrm{R}\big((4,2), \mathrm{a}, (4,2)\big) = 0$$
$$\forall a \ P\big((4,3)|(4,3), a\ \big) = 1, \mathrm{R}\big((4,3), \mathrm{a}, (4,3)\big) = 0$$

Policy: π_1



*Terminal states can also move to exit states with a specific reward:

$$\left.\begin{array}{l} \forall a \ P(EXIT|(4,2), a\ ) = 1, \mathrm{R}\big((4,2), \mathrm{a}, \mathrm{EXIT}\big) = -1 \\ \forall a \ P(EXIT|(4,3), a\ ) = 1, \mathrm{R}\big((4,3), \mathrm{a}, \mathrm{EXIT}\big) = 1 \end{array}\right\} \begin{array}{l} U^{\pi_1}((4,2)) = -1 \\ U^{\pi_1}((4,3)) = 1 \end{array}$$

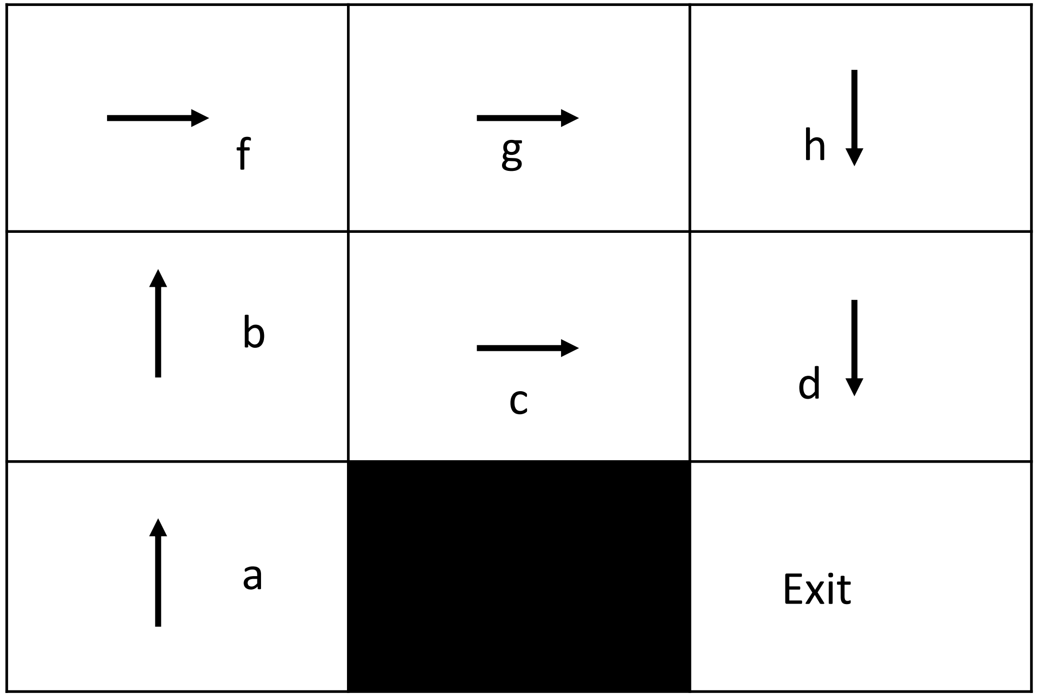
Inferencing with MDP

- Prediction Problem
  - Finding utility of a state given a policy π
  - i.e., calculation of $U^{\pi}(s)$

- Control Problem
  - Finding an optimal policy π^*
  - i.e., finding an action for each state (s) that maximizes its utility ($U(s)$)

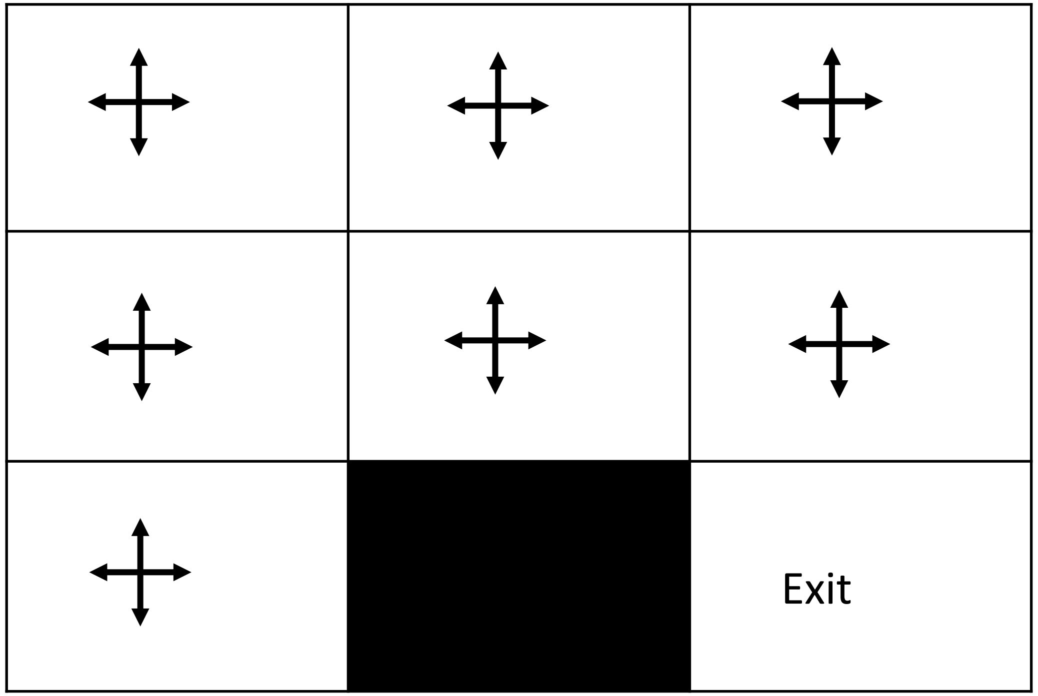
$$\pi^*(s) = \arg\max_{\pi} U^{\pi}(s)$$

Prediction or Policy Evaluation



Given a policy, what is the reward at each state?

Control



What is the optimal policy?

Finding optimal action at each state such that the average reward is maximized

Prediction Problem

- Calculation of expected utility of state s with a policy π

Known quantities (Model Parameters)

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

unknowns

- $|S|$ linear equations with $|S|$ unknowns
- How to solve these equations?
- Two methods:
  - Direct method using linear algebra
  - Iterative Methods

Prediction Problem

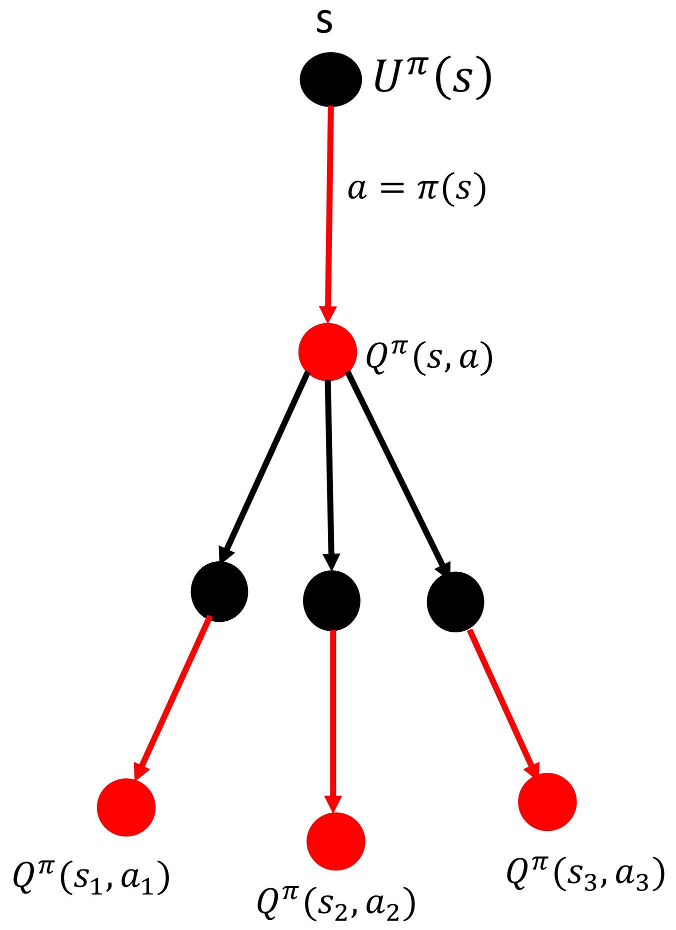
- Calculation of Action-Utility for a given policy

Known quantities (Model Parameters)

$$\boxed{Q^{\pi}(s,a)} = \sum_{s'} \boxed{P(s'|a, s_1)}\left[\boxed{R(s,a,s')} + \gamma \boxed{Q^{\pi}(s',a')}\right]$$

unknowns

$a = \pi(s)$ $\qquad$ $a' = \pi(s')$



$s' = \{s_1, s_2, s_3\}$

$a' = \{a_1, a_2, a_3\}$

Direct Method: Example

Linear Equations

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$
..
$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n$$

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ . \\ b_n \end{bmatrix}$$

$$A\,\boldsymbol{x} = \boldsymbol{b} \Rightarrow \boldsymbol{x} = A^{-1}\boldsymbol{b}$$

Complexity: $O(|S|^3)$
Ok for MDPs with small state spaces
Challenging for MDPs with large state spaces

Iterative Methods

Iterative method

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$
..
$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n$$

$$x_{1,k} = -\frac{a_{12}}{a_{11}}x_{2,i} - \cdots - \frac{a_{1n}}{a_{11}}x_{n,k-1} + \frac{b_1}{a_{11}}$$
$$x_{2,k} = -\frac{a_{21}}{a_{22}}x_{1,i} - \cdots - \frac{a_{2n}}{a_{22}}x_{n,k-1} + \frac{b_2}{a_{22}}$$
$$x_{n,k} = -\frac{a_{12}}{a_{nn}}x_{2,k-1} - \cdots - \frac{a_{n-1n}}{a_{nn}}x_{n-1,k-1} + \frac{b_n}{a_{nn}}$$

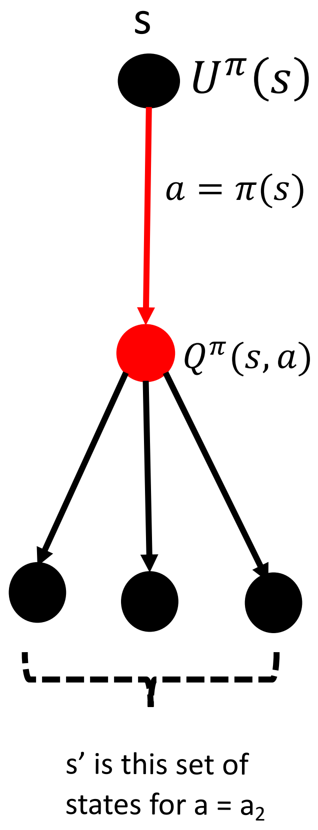
- Initialize the vector $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix}$ with zeros and evaluate the above equations iteratively
- Stop when there is no change in the values or after a fixed number of iterations
- Suitable for nonlinear equations as well
  - We need them later in Value Iteration algorithm
- Need to take care of convergence

So far: Prediction

- Policy Evaluation Given a Policy

- $U^{\pi}(s)$: utility of state s with a given policy π

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



Next: Control

- Control:
  - Finding optimal policy

- How?
  - Brute Force Algorithm
  - Policy Iteration (PI)
  - Value Iteration (VI), etc.

Finding Optimal Policy: Brute Force Algorithm

- Find utilities of states with all policies

- Policy with maximum utility at each state is optimal policy
  - $\pi^*(s) = \arg\max_{\pi} U^{\pi}(s)$

- Number of Policies: $|A|^{|S|}$
  - If size of state space ($|S|$) is small and number of actions per state ($|A|$) are also small, we can evaluate all policies and select the policy with best utilities across all states
  - Usually $|S|$ is large

- Evaluation of all possible policies is impossible

Finding Optimal Policy: VI and PI

- Relies on greedy approach
- Choose the action that maximizes the reward for next step plus discounted utility of subsequent state

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|a,s)[R(s,a,s') + \gamma\, U(s')]$$
$$= \arg\max_{a \in A(s)} Q(s,a)$$

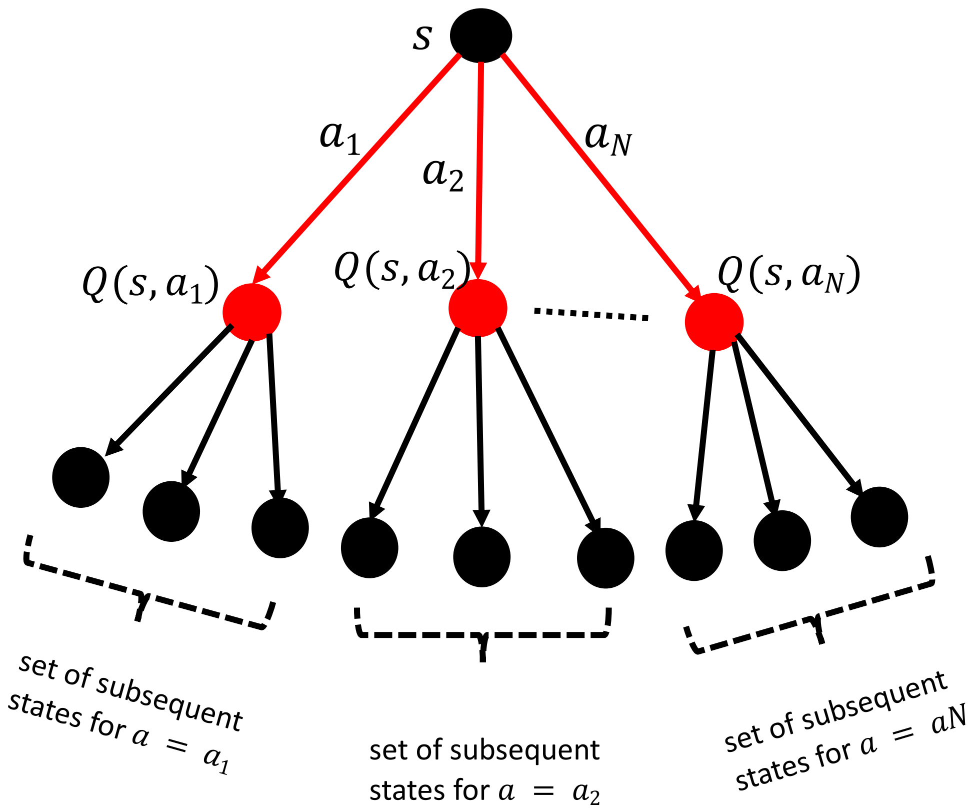
(one-step look-ahead)

- But we should know optimal utility of subsequent states (i.e., s')!
  - Otherwise, greedy approach is suboptimal

- That means we should also know the optimal action for subsequent states!!

Find optimal action from $A(s)$ for all s

$A(s) = \{a_1, a_2, \ldots, a_N\}$



Optimal Utility

- Utility (value) of a state $U(s)$ is the expected reward for the next transition plus the discounted utility of next state, assuming that the agent chooses the optimal action

Bellman equation for utility

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

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

$$U(s) = \max_{a \in A(s)} Q(s,a)$$

Optimal Action-Utility

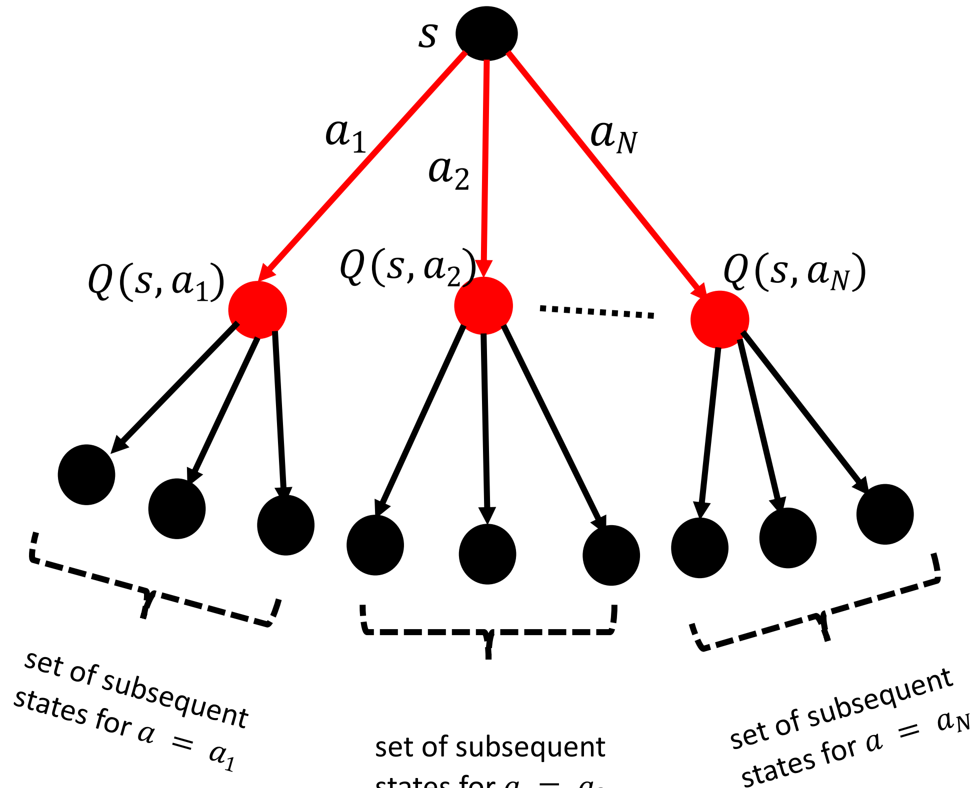
- Action-Utility $Q(s,a)$ of a state-action pair (s,a) is the expected reward for action a at state s, assuming that the agent chooses the optimal actions at subsequent states

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

Bellman equation for Q-value

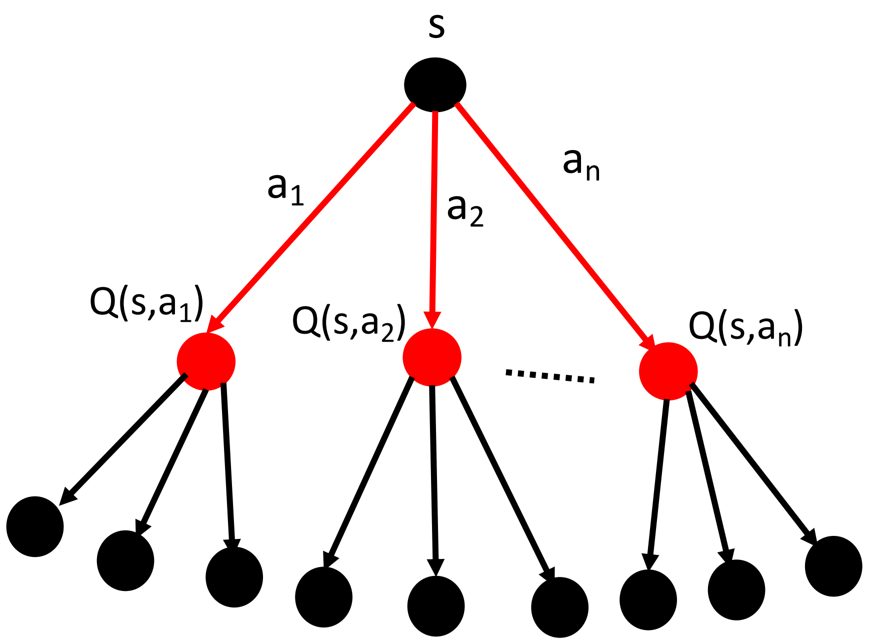
$$\boxed{Q(s,a) = \sum_{s'} P(s'|a,s)\left[R(s,a,s') + \gamma \max_{a' \in A(s')} Q(s',a')\right]}$$

$$\pi^*(s) = \arg\max_{a \in A(s)} Q(s,a)$$



Value Iteration (VI)

- Start with $U_i(s) = 0$ for all s
- Calculate Q values and update Utility values



$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|a,s)[R(s,a,s') + \gamma U_i(s')]$$

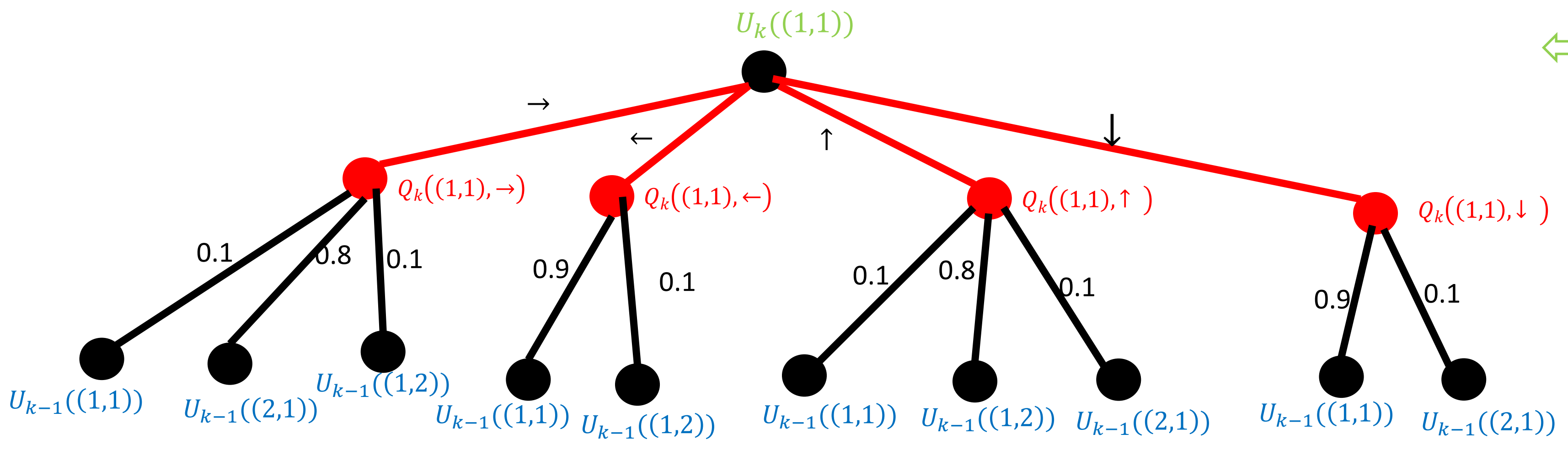
$$U_{i+1}(s) = \max_{a \in A(s)} Q_{i+1}(s,a)$$

Bellman update equations

- Once utility values converge to optimal values, select the corresponding policy

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

Maze: Bellman Equation for state (1,1)



Updated utility of state (1,1) at k-th iteration

Old utilities of neighbors of (1,1) from (k-1)-th iteration

$$Q_k((1,1),\rightarrow) = 0.8\left(-0.04+\gamma U_{k-1}((1,1))\right) + 0.1\left(-0.04+\gamma U_{k-1}((2,1))\right) + 0.1\left(-0.04+\gamma U_{k-1}((1,2))\right)$$

$$Q_k((1,1),\leftarrow) = 0.9(-0.04+\gamma U_{k-1}((1,1))) + 0.1(-0.04+\gamma U_{k-1}((1,2))),$$

$$Q_k((1,1),\uparrow) = 0.8\left(-0.04+\gamma U_{k-1}((1,2))\right) + 0.1\left(-0.04+\gamma U_{k-1}((2,1))\right) + 0.1\left(-0.04+\gamma U_{k-1}((1,1))\right)$$

$$Q_k((1,1),\downarrow) = 0.9(-0.04+\gamma U_{k-1}((1,1))) + 0.1(-0.04+\gamma U_{k-1}((2,1))),$$

$$U_k((1,1)) = \max\{Q_k((1,1),\rightarrow), Q_k((1,1),\leftarrow), Q_k((1,1),\uparrow), Q_k((1,1),\downarrow)\}$$

Value Iteration (VI) Algorithm

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s, a, s')$, 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 relative change in the utility of any state

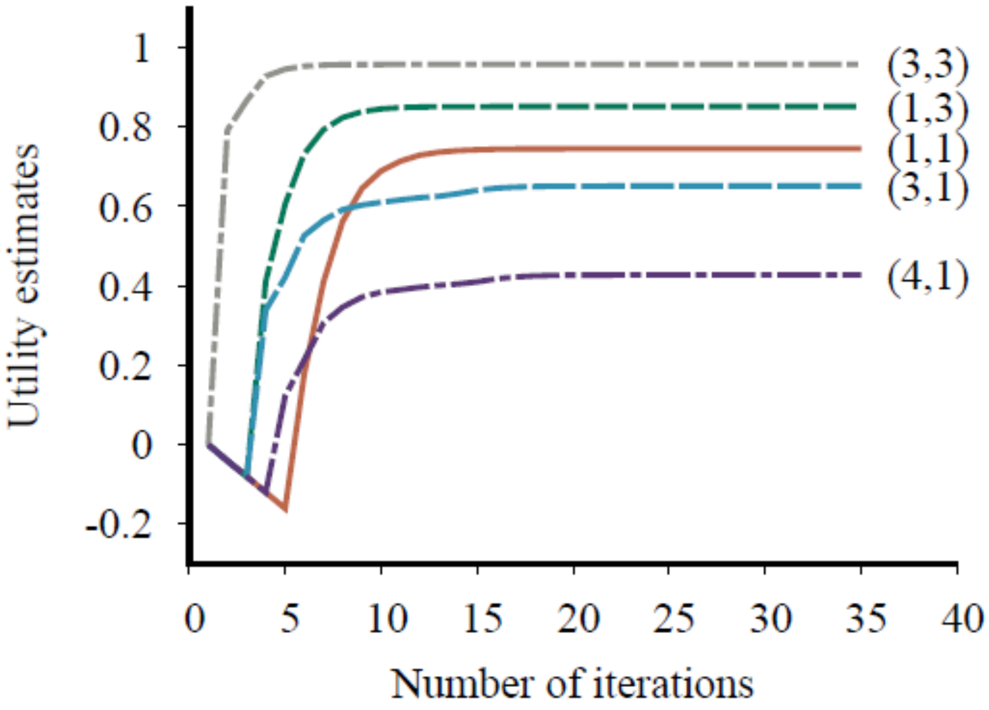
  repeat
   $U \leftarrow U'$; $\delta \leftarrow 0$
   for each state s **in** S **do**
    $U'[s] \leftarrow \max_{a \in A(s)}$ Q-VALUE(mdp, s, a, U) → Bellman update equation
    if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$ → Smallest change in U values among all states
  until $\delta \leq \epsilon(1-\gamma)/\gamma$ → Checking for convergence
  return U

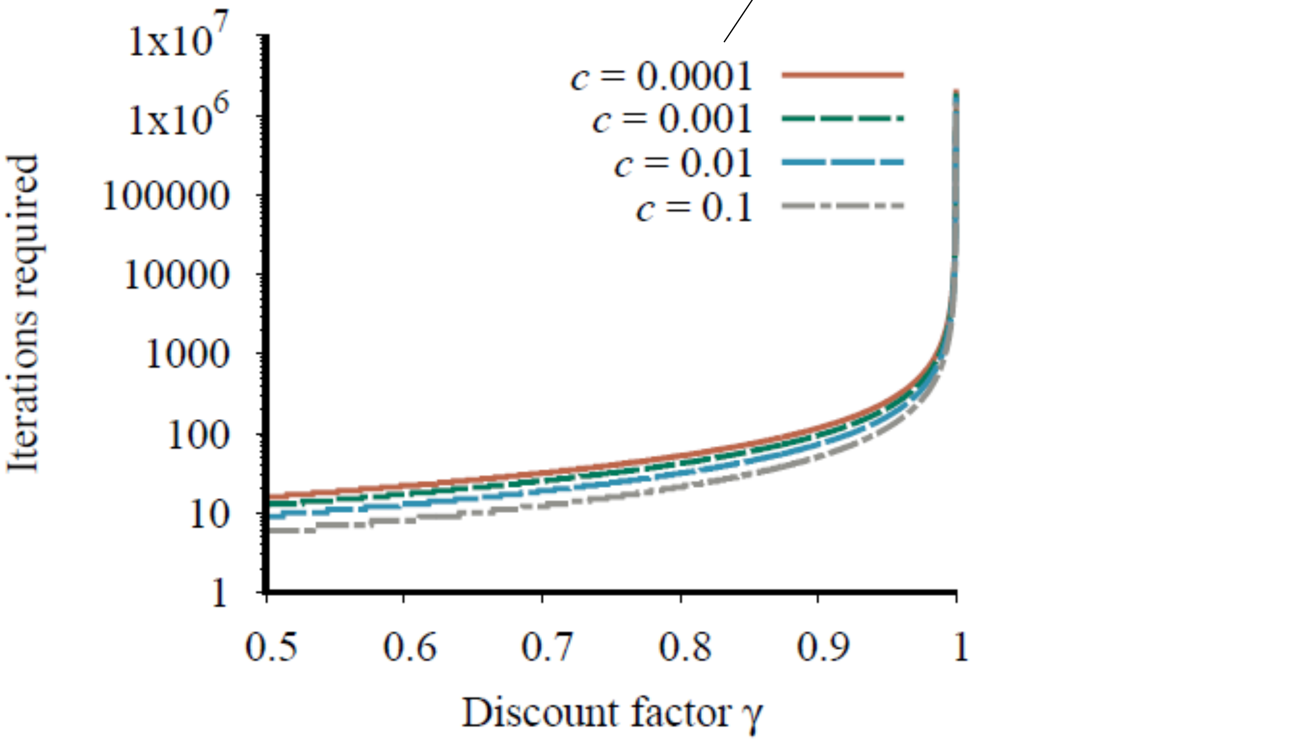
function $Q\text{-}VALUE(mdp, s, a, U)$ **returns** a utility value
  return $\sum_{s'} P(s'|a, s_1)[R(s,a,s') + \gamma\, U(s')]$

VI algorithm calculates optimal values
Perform **one-step look-ahead** to extract optimal policy from optimal values

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

Value Iteration: Analysis





$$N = \left\lceil \frac{\log\left(\frac{2R_{max}}{\epsilon(1-\gamma)}\right)}{\log\left(\frac{1}{\gamma}\right)} \right\rceil$$

Converges faster for small discount factor

Optimal Policies: Example

Depends on Reward and Transition Probability

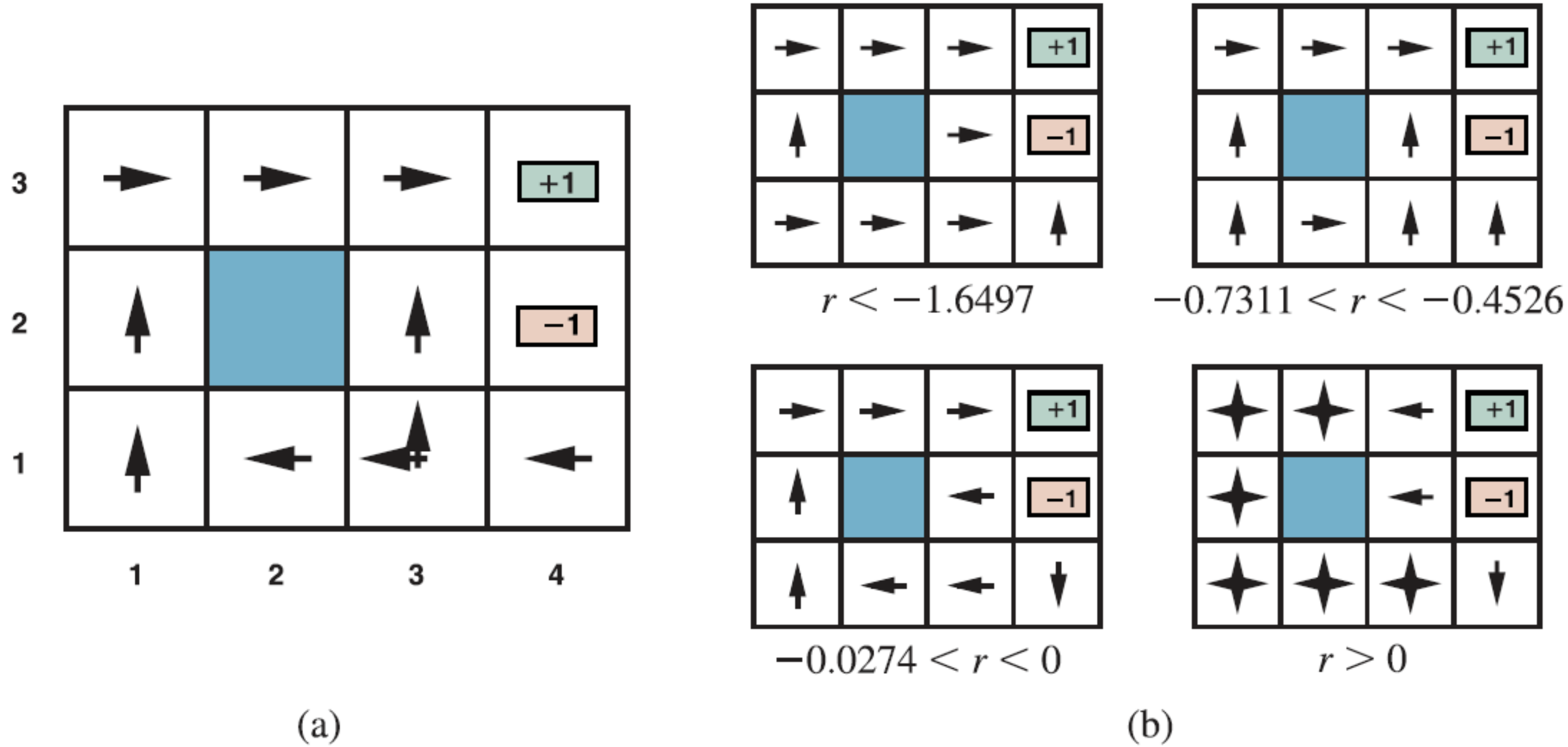
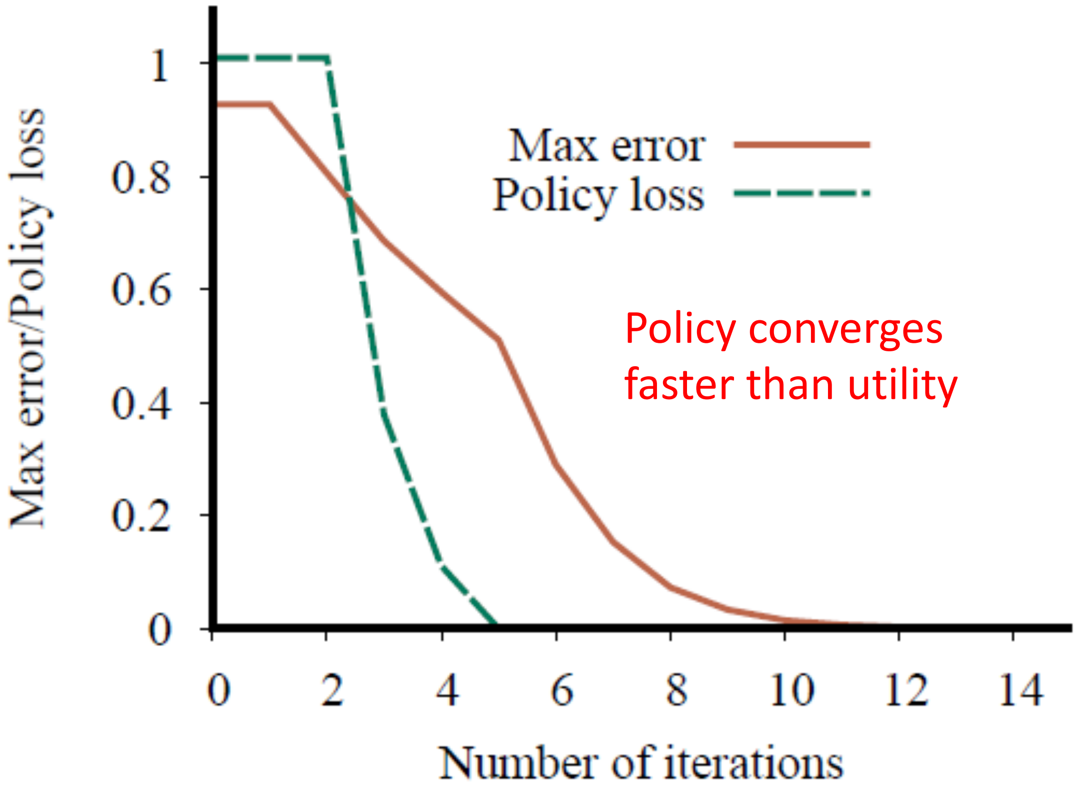


Figure 17.2 (a) The optimal policies for the stochastic environment with $r = -0.04$ for transitions between nonterminal states. There are two policies because in state (3,1) both *Left* and *Up* are optimal. (b) Optimal policies for four different ranges of r.

Value Iteration: Analysis



Policy converges faster than utility

$$Policy\ Loss = ||U^{\pi_i} - U||$$

$$Max\ Error = ||U_i - U||$$

$$\|U\| = \max_s |U(s)|$$

U^{π_i} : Utility with policy π_i

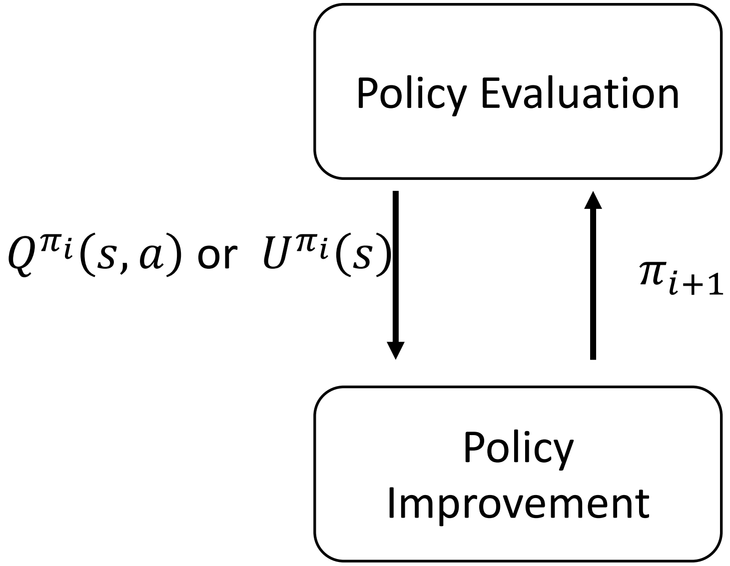
U_i: Utilities at i-th iteration

π_i: Policy recommended by value iteration if algorithm stops at i-th iteration

U: Optimal Utility

Policy Iteration

- Start with a random policy π_i
- Policy Evaluation:
  - Calculate utility values for the policy π_i
- Policy Improvement:
  - Calculate new policy π_{i+1} based on utility values
- Repeat the above steps till no change in policy



Policy Iteration

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  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(π, U, mdp) → Calculate utility of each state with policy π
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      $a^* \leftarrow \underset{a \in A(s)}{\operatorname{argmax}}$ Q-VALUE(mdp, s, a, U) → Update the policy π based on utilities
      if Q-VALUE(mdp, s, a^*, U) $>$ Q-VALUE($mdp, s, \pi[s], U$) **then**
        $\pi[s] \leftarrow a^*$; $unchanged? \leftarrow$ false → If Q values of new policy are strictly larger than old policy, update the policy
  until $unchanged?$
  return π

U values cannot be used for policy update (why?)
So obtain Q values from U values

function $Q\text{-}VALUE(mdp, s, a, U)$ **returns** a utility value
  return $\sum_{s'} P(s'|a, s_1)[R(s, a, s') + \gamma\, U(s')]$

Conclusion

- Markov Decision Processes
  - Models uncertainty in environment
  - Utility-based reasoning

- Inferencing
  - Value Iteration
  - Policy iteration

Expected Utility of State s_0 with a Policy π

$$
\begin{aligned}
U^{\pi}(s_0) &= E\left[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, \pi(s_t), s_{t+1}\right) | s_0, \pi(s_0)\right] \\
&= E\left[R\left(s_t, \pi(s_t), s_{t+1}\right) + \sum_{t=1}^{\infty} \gamma^t R\left(s_t, \pi(s_t), s_{t+1}\right) | s_0, \pi(s_0)\right] \\
&= \sum_{s_1} P(s_1|s_0, \pi(s_0)) \left[R\left(s_0, \pi(s_0), s_1\right) + E\left[\sum_{t=1}^{\infty} \gamma^t R\left(s_t, \pi(s_t), s_1\right) | s_1, \pi(s_1)\right]\right] \\
&= \sum_{s_1} P(s_1|s_0, \pi(s_0)) \left[R\left(s_0, \pi(s_0), s_1\right) + \gamma E\left[\sum_{t=1}^{\infty} \gamma^{t-1} R\left(s_t, \pi(s_t), s_{t+1}\right) | s_1, \pi(s_1)\right]\right] \\
&= \sum_{s_1} P(s_1|s_0, \pi(s_0)) \left[R\left(s_0, \pi(s_0), s_1\right) + \gamma U^{\pi}(s_1)\right]
\end{aligned}
$$

$$
P(s_0, \pi(s_0), s_1, \pi(s_1), \ldots) = P(s_0) P(s_1|s_0, \pi(s_0)) P(s_2|s_1, \pi(s_1)) P(s_3|s_2, \pi(s_2)) \ldots
$$