CS 581

Advanced Artificial Intelligence

April 8, 2024

Announcements / Reminders

- Please follow the Week 12 To Do List instructions (if you haven't already)

- Programming Assignment #03: OPTIONAL/NOT FOR CREDIT

- FINAL EXAM is on Monday (04/22/2024) in RE 104!
  - different room!!!
  - IGNORE Registrar's FINAL EXAM date
  - Section 02: contact Mr. Charles Scott (scott@iit.edu) to make arrangements

Plan for Today

- K-Armed Bandit Problem revisited
- Making Complex Decisions

Multi-Armed Bandits
(K-Armed Bandits)

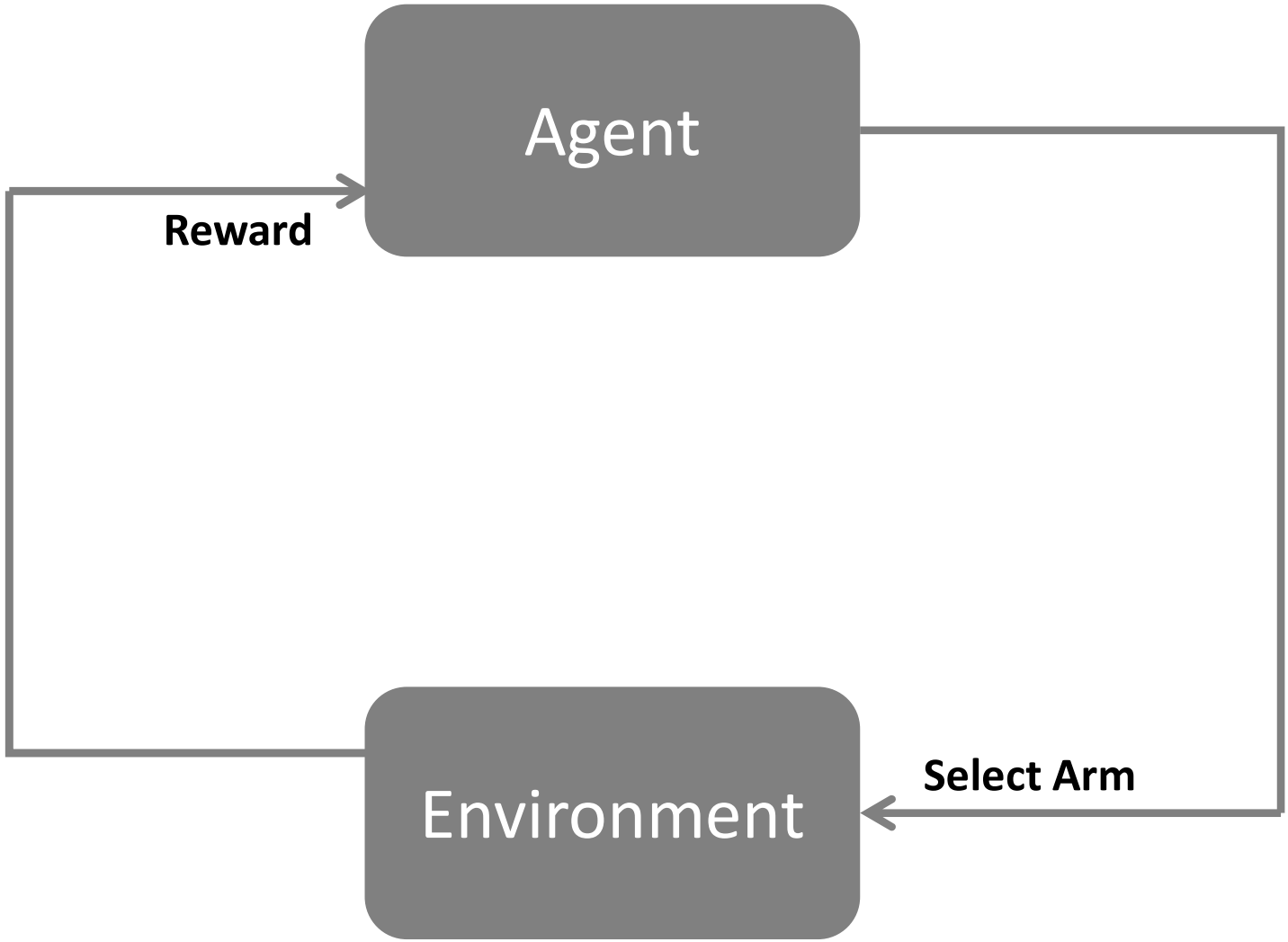
K-Armed Bandit Problem

K-Armed Bandit Problem

The K-armed bandit problem is a problem in which a fixed limited set of resources must be allocated between competing (alternative) choices in a way that maximizes their expected gain.

Each choice's properties are only partially known at the time of allocation, and may become better understood as time passes or by allocating resources to the choice.

K-Armed Bandit



Problem Summary

- K slot machines
  - Each one is a one-armed bandit
- Unknown reward functions
  - The distribution of rewards for each machine are unknown
- Limited resources
  - Have a limited number of tries (or alternatively, future rewards are discounted)
- Can gather information
  - Each try gives a (potentially) zero/negative reward, but also is useful for information gathering purposes
- Main objective
  - Maximize rewards
- Exploration vs exploitation trade-off
  - How should you balance exploitation (sticking with a machine that looks good) versus exploration (trying new machines)?

Notation

- A_t : "action at time" t
- R_t : "reward at time" t
- **Sequence of actions and rewards:** $A_1, R_1, A_2, R_2, \cdots, A_T, R_T$
- **Expected reward:** $q_*(a) = E[R_t \mid A_t = a]$
  - Not given; otherwise, the solution is trivial
- $Q_t(a)$: "Average reward for action " a "up to [excluding] time" t
  - Estimated using prior experiences
  - If action a has never been taken before time t, $Q_t(a)$ is initialized to a default value

$$Q_t(a) = \frac{sum\ of\ rewards\ when\ (a)\ taken\ prior\ to\ (t)}{number\ of\ times\ (a)\ taken\ prior\ to\ (t)}$$

Optimal Strategy

- **Pick an action a such that:**

$$\underset{a}{\operatorname{argmax}}\, q_*(a)$$

- **In other words:**
  - **at any time t, choose the action that has the highest expected reward**
- **Challenge?**
  - **we do not know $q_*(a)$**

Greedy Strategy (Exploit Only)

- **Pick an action a such that:**

$$\underset{a}{\operatorname{argmax}}\, Q_t(a)$$

- **In other words:**
  - **Calculate the average reward for each action, up to time t, and choose the maximum one**
  - **Challenge?**
  - **we do not know $q_*(a)$**

Greedy Strategy (Exploit Only)

- **Pick an action a such that:**

$$\underset{a}{\operatorname{argmax}}\, Q_t(a)$$

- **Problems:**
  - **Initial value of $Q_t(a)$ plays a big role**
  - **It simply tries the best action it has found; purely exploitation**
  - **Could easily miss other actions that are better but not tried**

Random Strategy (Explore Only)

- Pick an action a at random

- Pros:
  - Explores all the time
- Cons:
  - Does not exploit = does not learn

ε-greedy Algorithm

```
generate random number p ∈ [0,1]

if (p < ε)              // explore

    select random arm

else                    // exploit

    select current best arm

end
```

ε-greedy Algorithm

```
generate random number p ∈ [0,1]

if (p < ε)              // explore

    select random arm

else                    // exploit

    select current best arm

end
```

$\epsilon = 0$ → fully greedy | $\epsilon = 1$ → fully random

Exploration vs. Exploitation

The crucial tradeoff the gambler faces at each trial is between "exploitation" of the machine that has the highest expected payoff and "exploration" to get more information about the expected payoffs of the other machines.

Rewards Distributions / Simulations

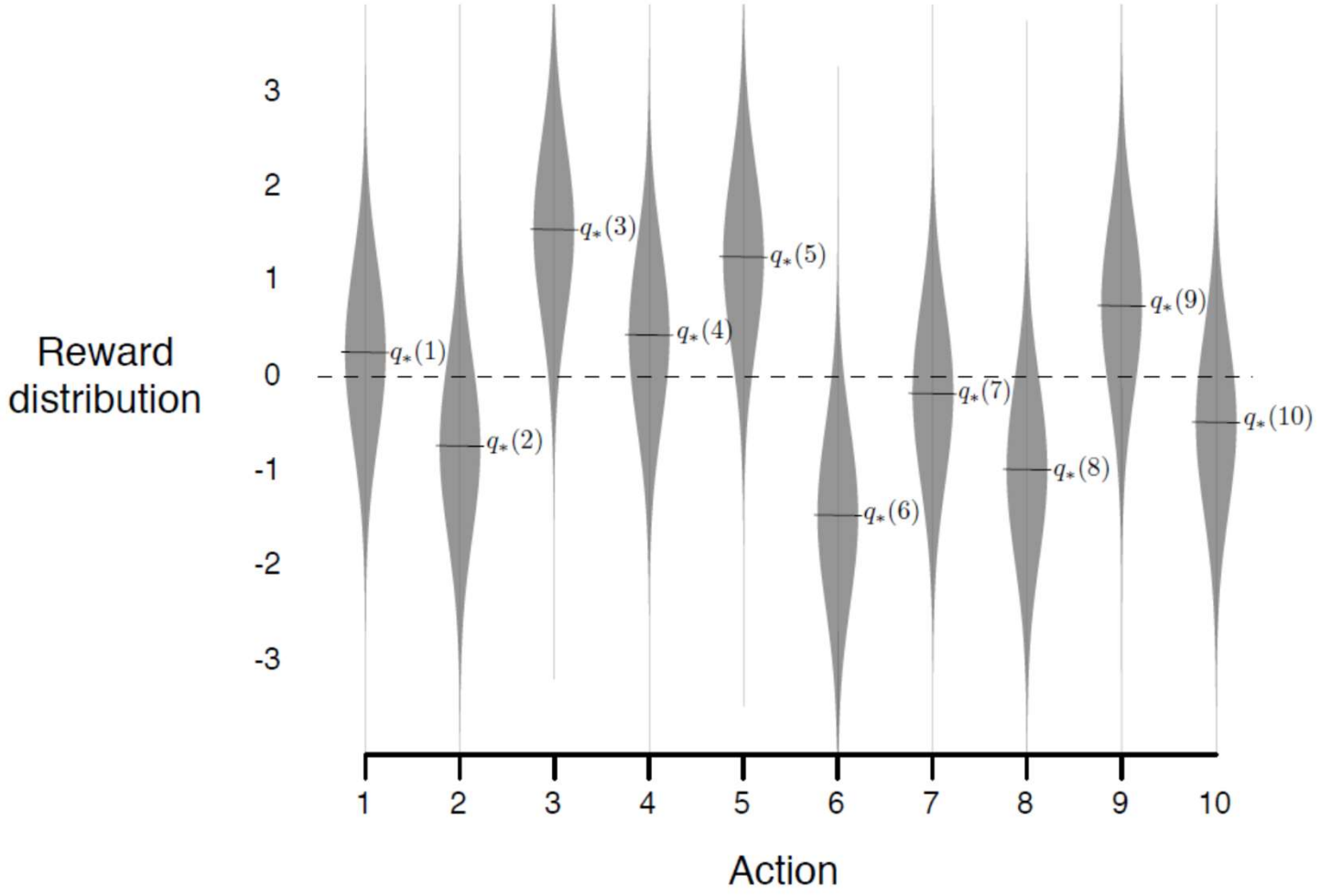


Figure 2.1: An example bandit problem from the 10-armed testbed. The true value $q_*(a)$ of each of the ten actions was selected according to a normal distribution with mean zero and unit variance, and then the actual rewards were selected according to a mean $q_*(a)$, unit-variance normal distribution, as suggested by these gray distributions.

Source: http://www.incompleteideas.net/book/the-book-2nd.html

Results = f(epsilon)

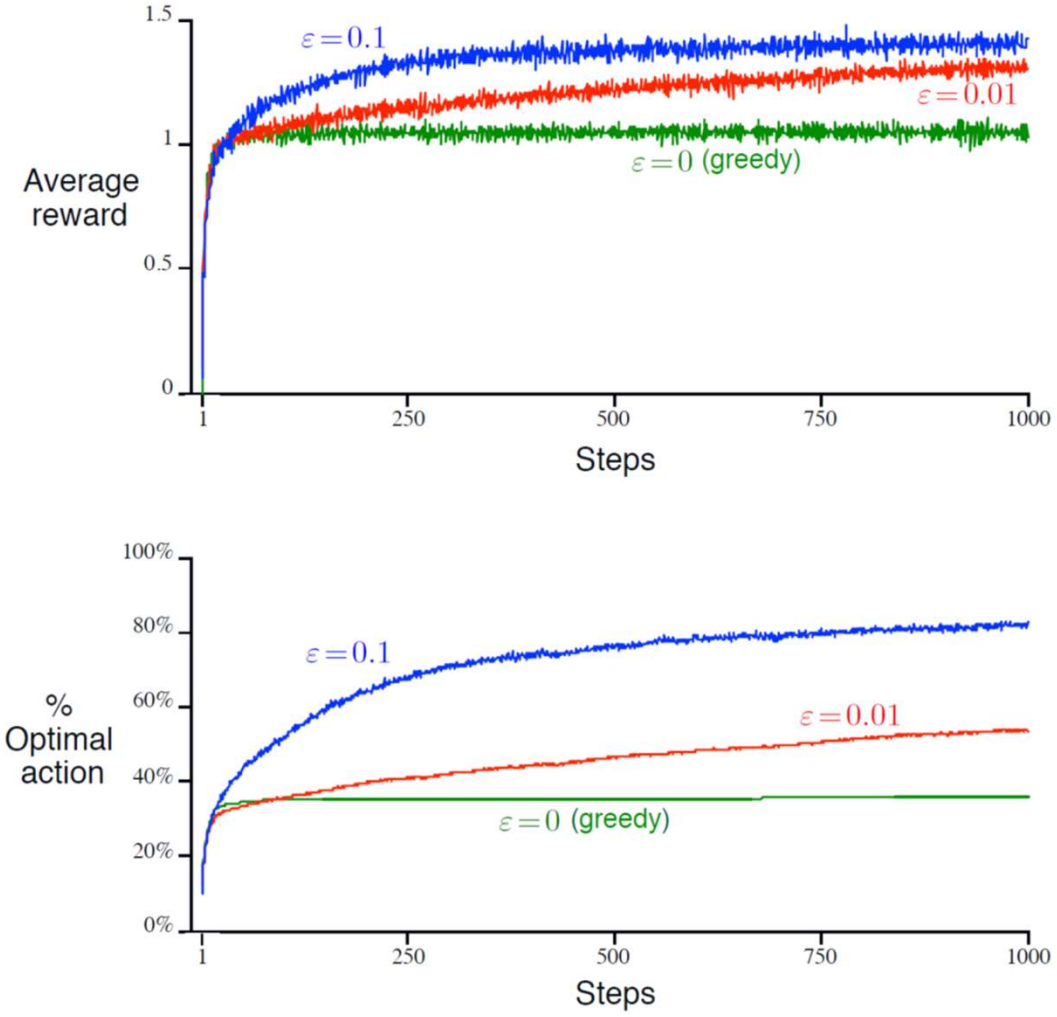


Figure 2.2: Average performance of ε-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates.

Source: http://www.incompleteideas.net/book/the-book-2nd.html

Upper Confidence Bound

- **The upper-confidence bound (UCB) method calculates the upper bound on the mean for each slot and chooses the machine with max value**
  - $Q_t(a) + c\sqrt{\frac{\ln(t)}{N_t(a)}}$, **where** $N_t(a)$ **is the number of times the action** a **is tried and** c **is the exploration/exploitation trade-off parameter**
- **The term in the square root is a measure of uncertainty in** $\mathrm{Q_t(a)}$
  - **hence the name upper confidence bound**
- **The exploration grows with** $\ln(\mathrm{t})$**, shrinks with** $N_t(a)$

Upper Confidence Bound

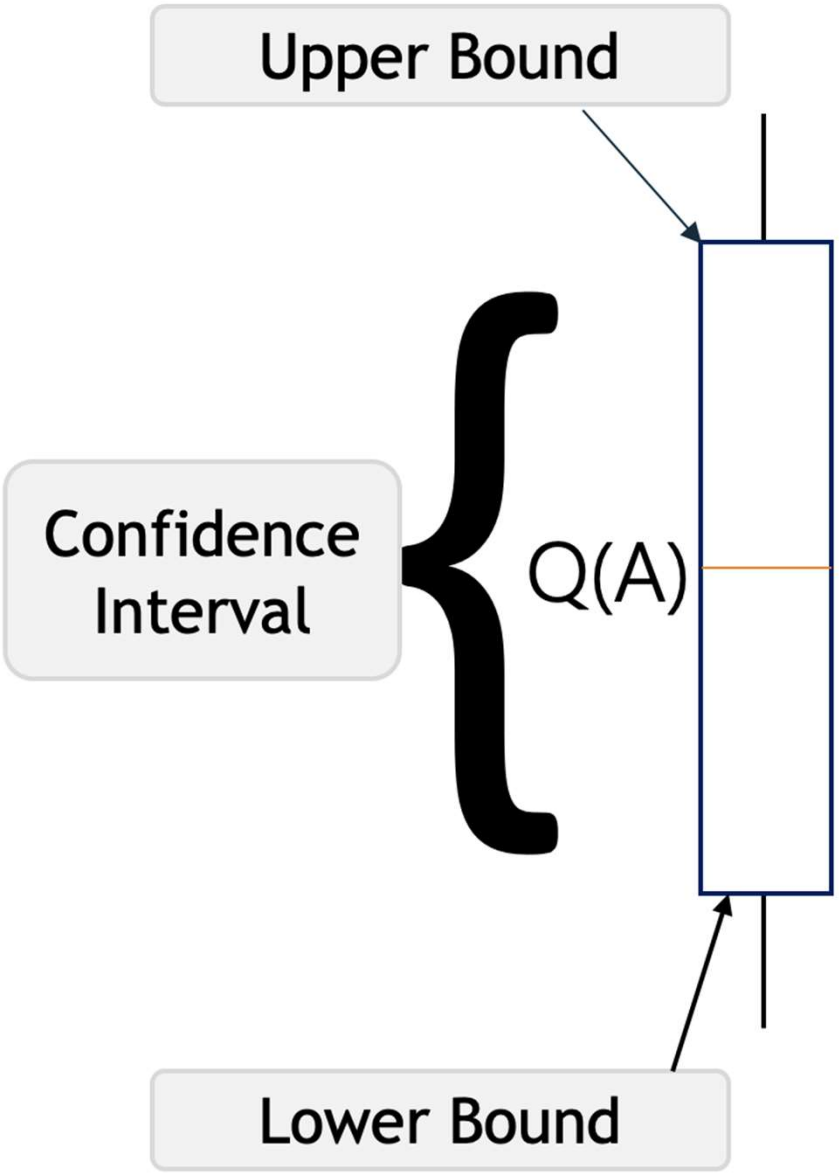
$$A_t = \text{argmax}_a \left(Q_t(a) + c \sqrt{\frac{\ln(t)}{N_t(a)}} \right)$$

Exploit

Explore

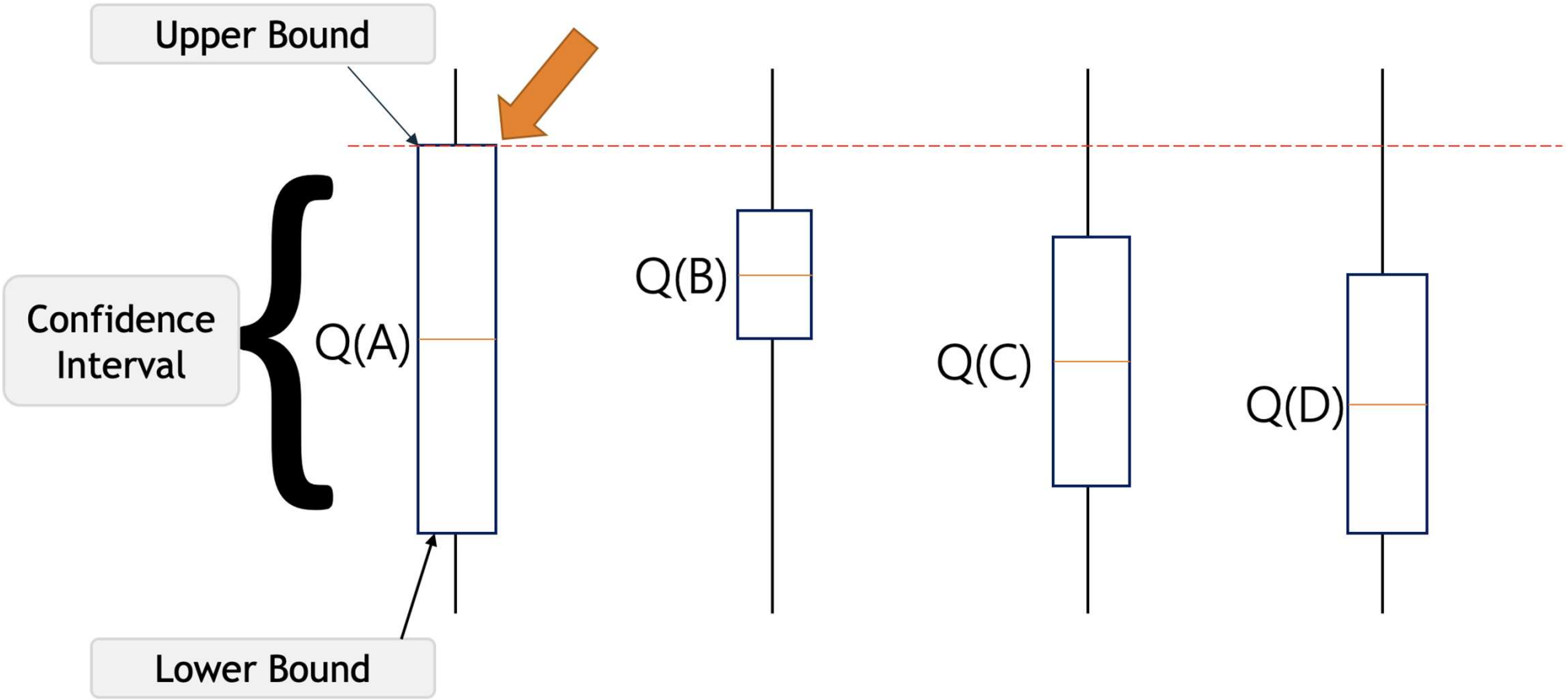
Source: https://www.geeksforgeeks.org/upper-confidence-bound-algorithm-in-reinforcement-learning/

Upper Confidence Bound



Source: https://www.geeksforgeeks.org/upper-confidence-bound-algorithm-in-reinforcement-learning/

Upper Confidence Bound



Source: https://www.geeksforgeeks.org/upper-confidence-bound-algorithm-in-reinforcement-learning/

UCB vs. Epsilon-greedy

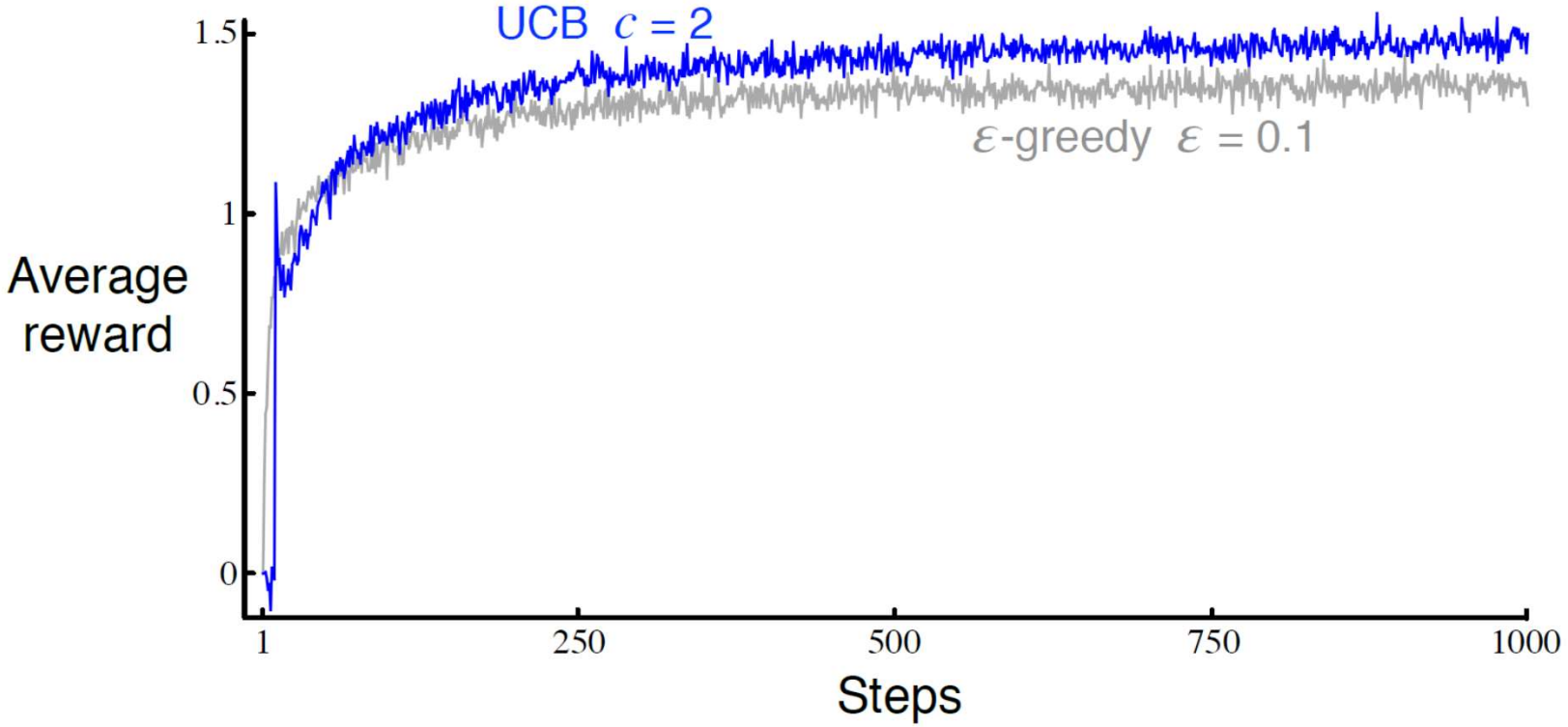


Figure 2.4: Average performance of UCB action selection on the 10-armed testbed. As shown, UCB generally performs better than ε-greedy action selection, except in the first k steps, when it selects randomly among the as-yet-untried actions.

Source: http://www.incompleteideas.net/book/the-book-2nd.html

Running Average Instead of Average

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
Q_{n+1} &= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}
$$

Update Rule

$$Q_{n+1} = Q_n + \frac{1}{n}(R_n - Q_n)$$

- **new = old + stepSize*(target – old)**
- **For computing the exact average, stepSize is a function of *n***

Non-stationary Rewards

- What if the reward distribution changes over time?
- We'd like to give recent rewards more weight
- A simple approach
  - $Q_{n+1} = Q_n + \alpha(R_n - Q_n)$
- Earlier rewards have lower weights

$$
\begin{aligned}
Q_{n+1} &= Q_n + \alpha(R_n - Q_n) \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)[\alpha R_{n-1} + (1-\alpha)Q_{n-1}]
\end{aligned}
$$

Sequential Decision Process

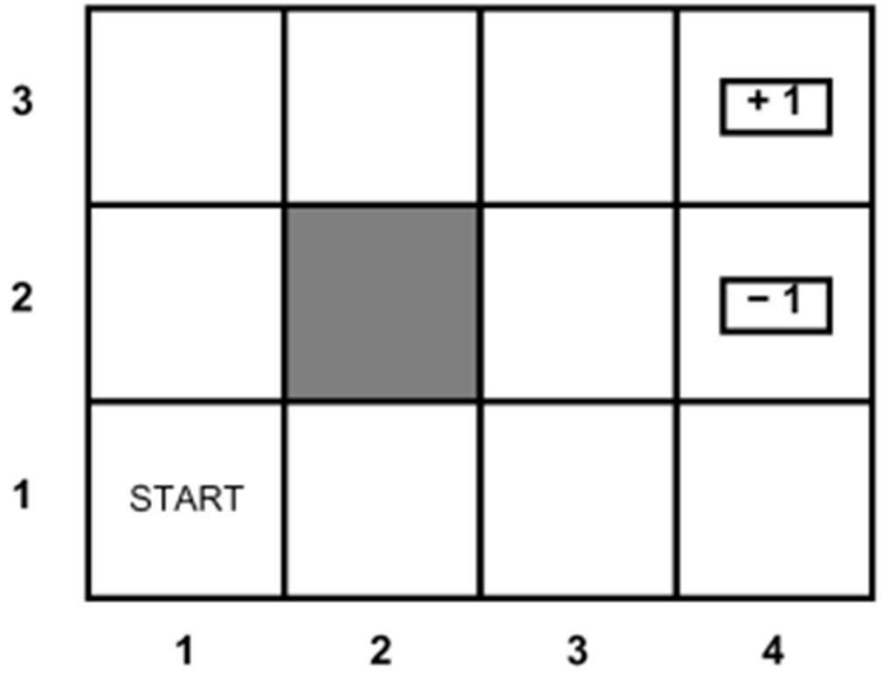
Sequential Decision Problem

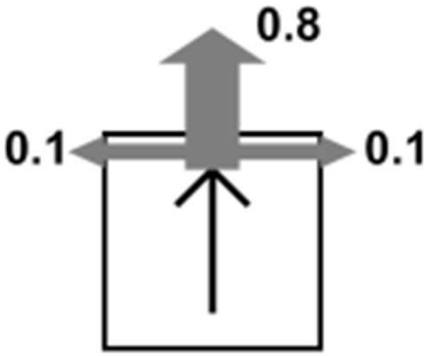
- Sequential decision problems are problems in which the agent's utility depends on a sequence of decisions.

- Sequential decision problems incorporate utilities, uncertainty, and sensing,
  - and include search and planning problems as special cases

Sequential Decision Problem

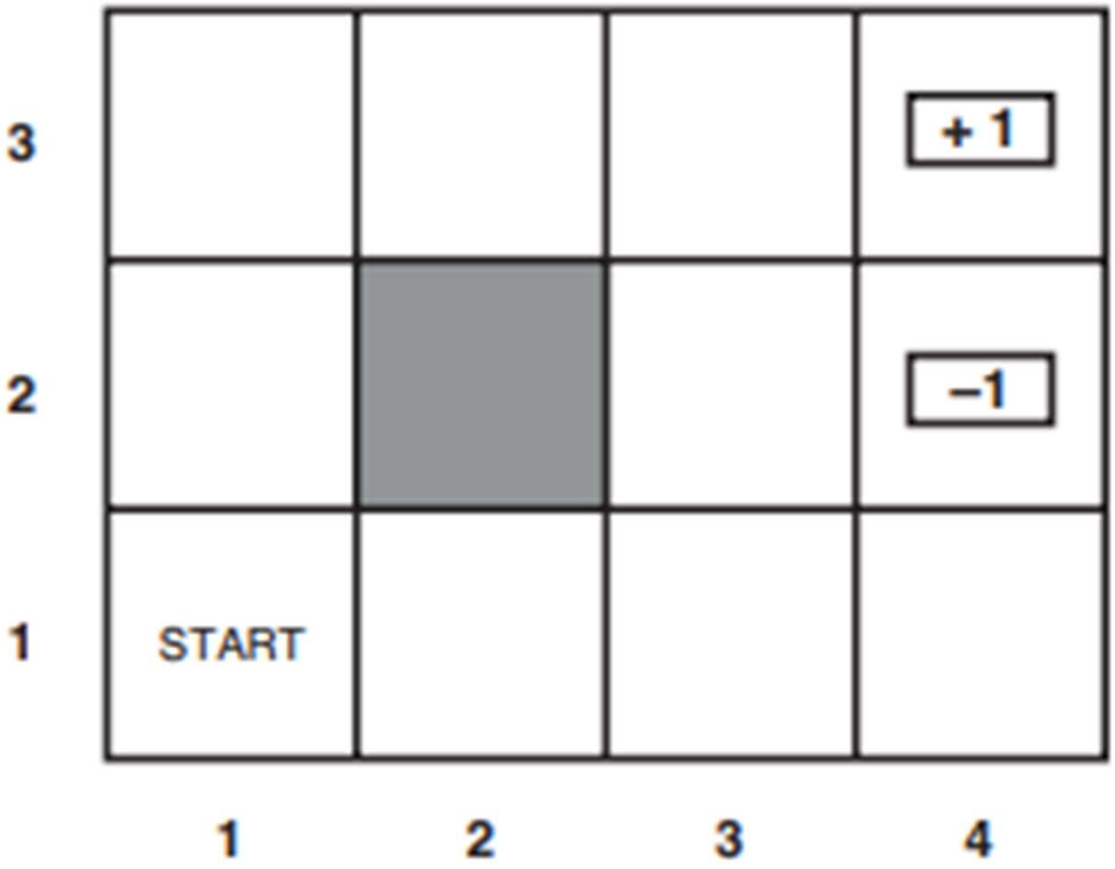
- The agent lives in a grid
- Walls block the agent's path
- The agent's actions do not always go as planned:
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Small "living" reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards*

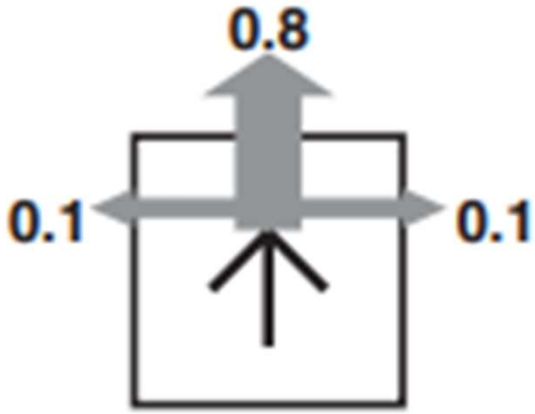




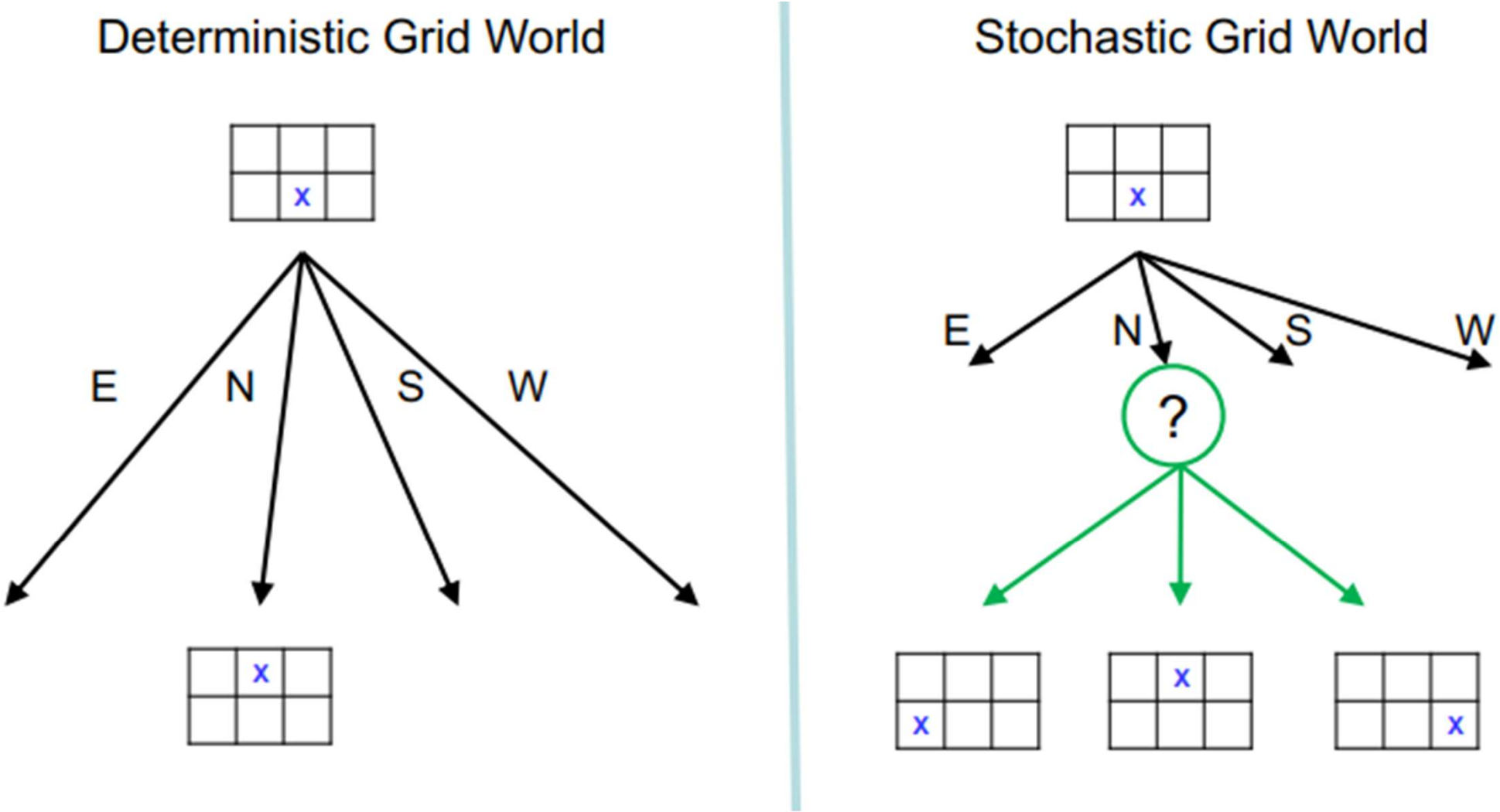
Example: Grid World

Fully-observable, non-deterministic, sequential



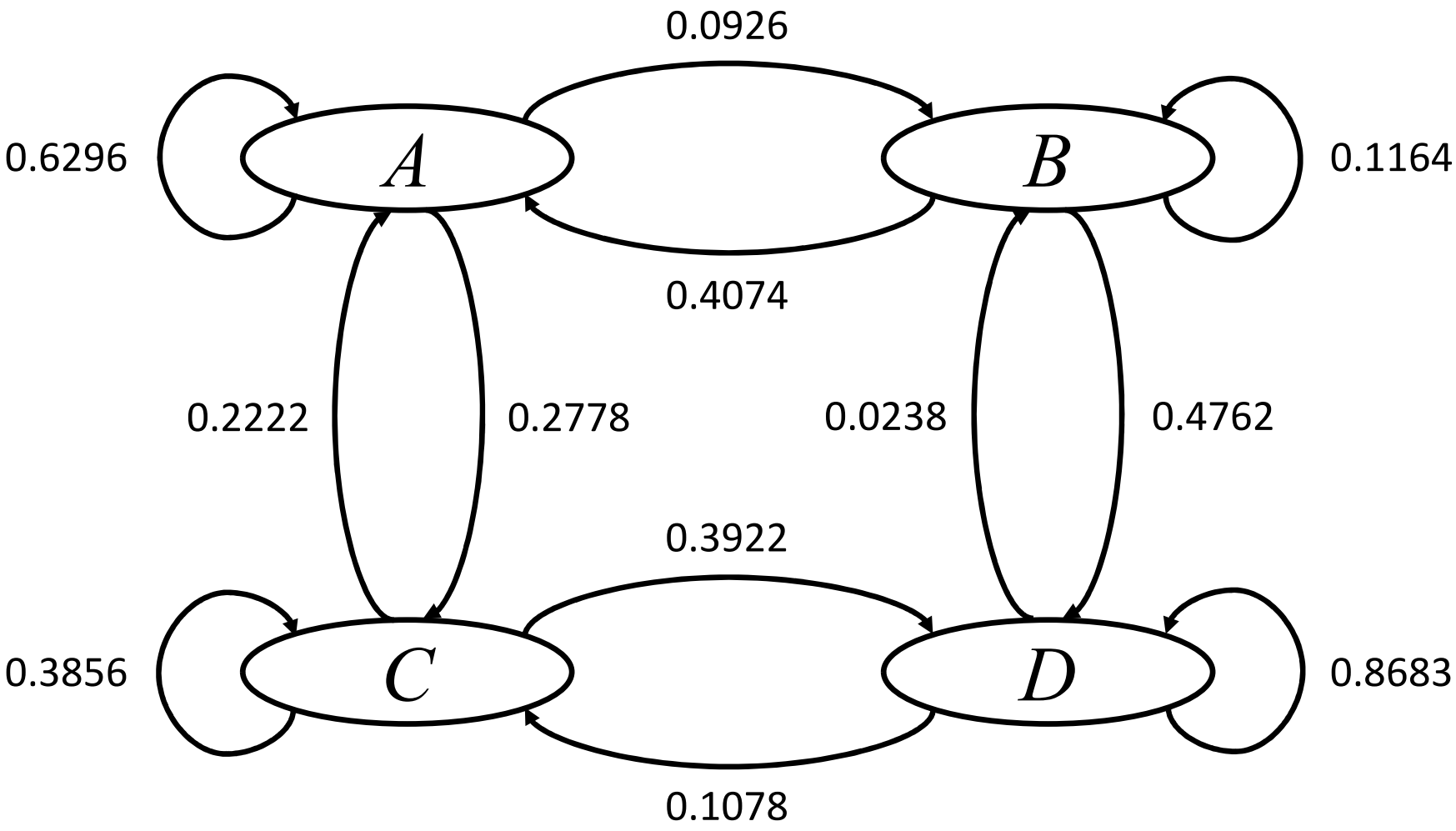


Deterministic vs. Non-Deterministic



Markov Decision Process (MDP)

State Space and Transition Model

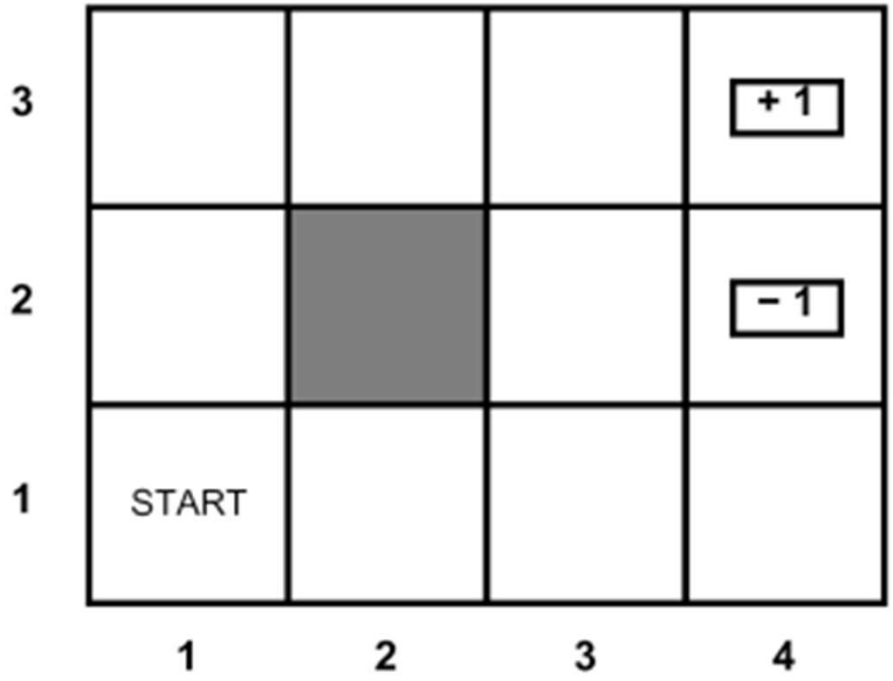


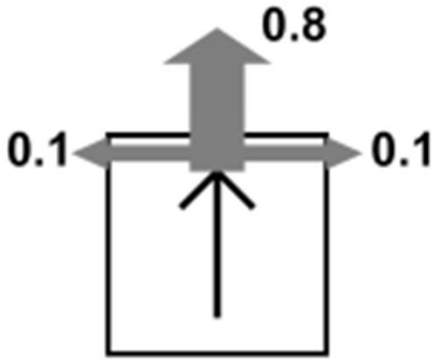
Problem Setting

- The world is represented through states
- At each state, an agent is given 0 (terminal states) or more actions to choose from
- Each action moves the agent, probabilistically, to a state (could be the current state) and results, probabilistically, in a reward (could be zero, negative, positive)
- The agent needs to maximize the sum of the rewards it accumulates over time
- Greedy strategy with respect to immediate rewards often do not work; the agent needs to consider the long-term consequences of its actions

Markov Decision Process

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function T(s,a,s')
    - Prob that a from s leads to s'
    - i.e., P(s' | s,a)
    - Also called the model
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state (or distribution)
  - Maybe a terminal state





Notation

- $P(s' \mid s, a)$ Probability of arriving at state s' given we are at state s and take action a
- $R(s, a, s')$ The reward the agent receives when it transitions from state s to state s' via action a

Markov Decision Process

- A sequential decision-making process
- Stochastic environment
- Markov transition model
- Additive rewards

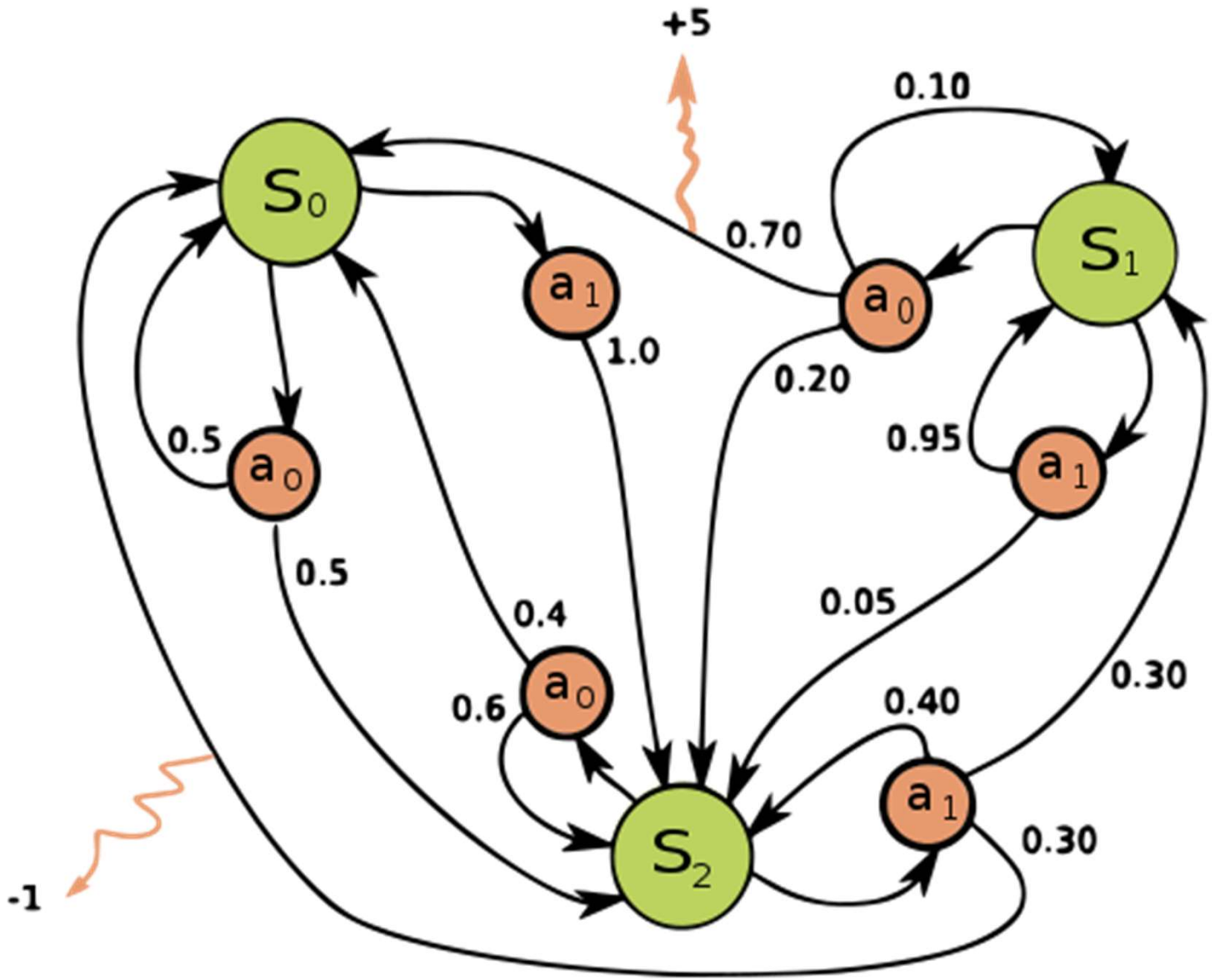
What is Markovian About MDP?

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

$$=$$

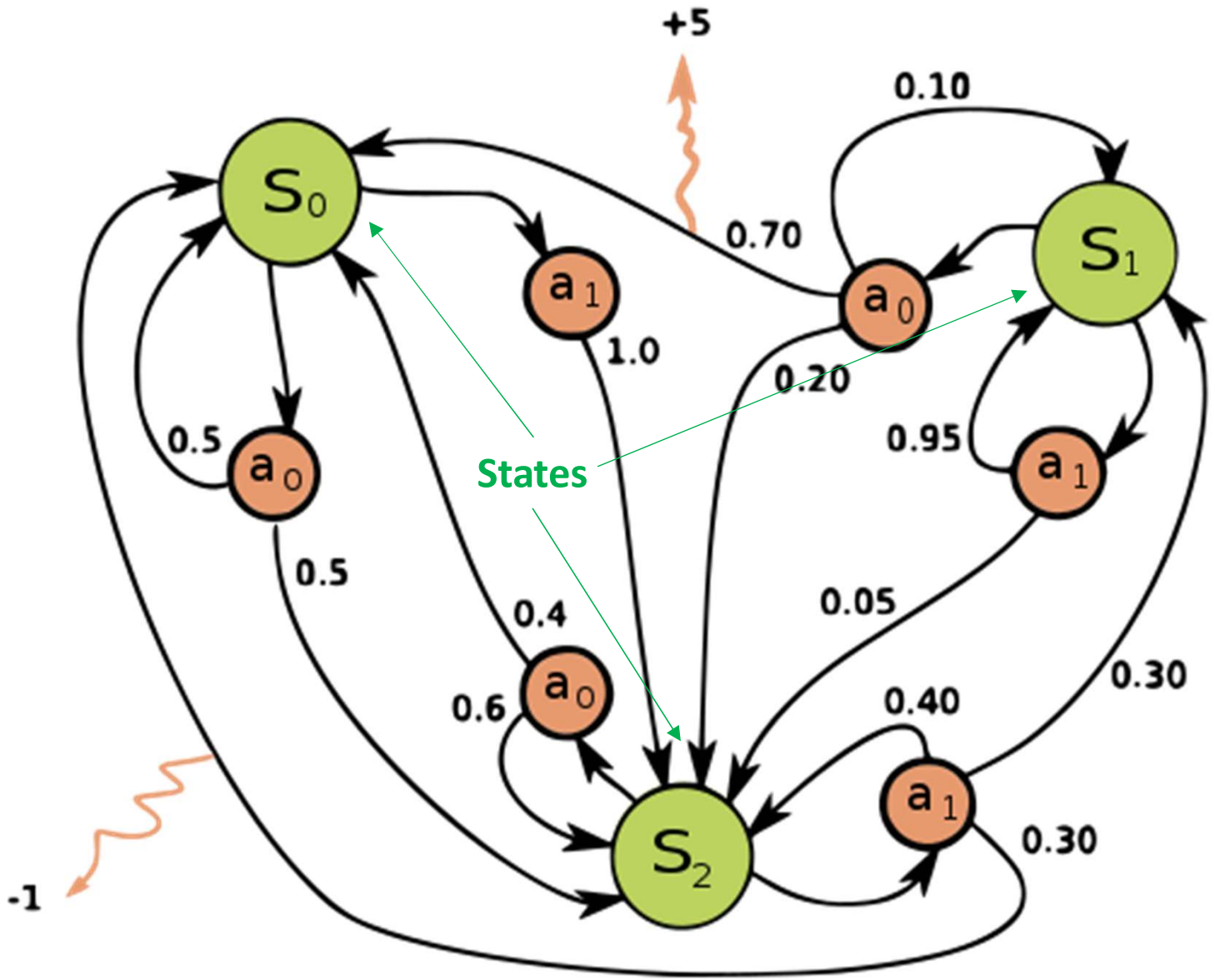
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

Markov Decision Process (MDP)



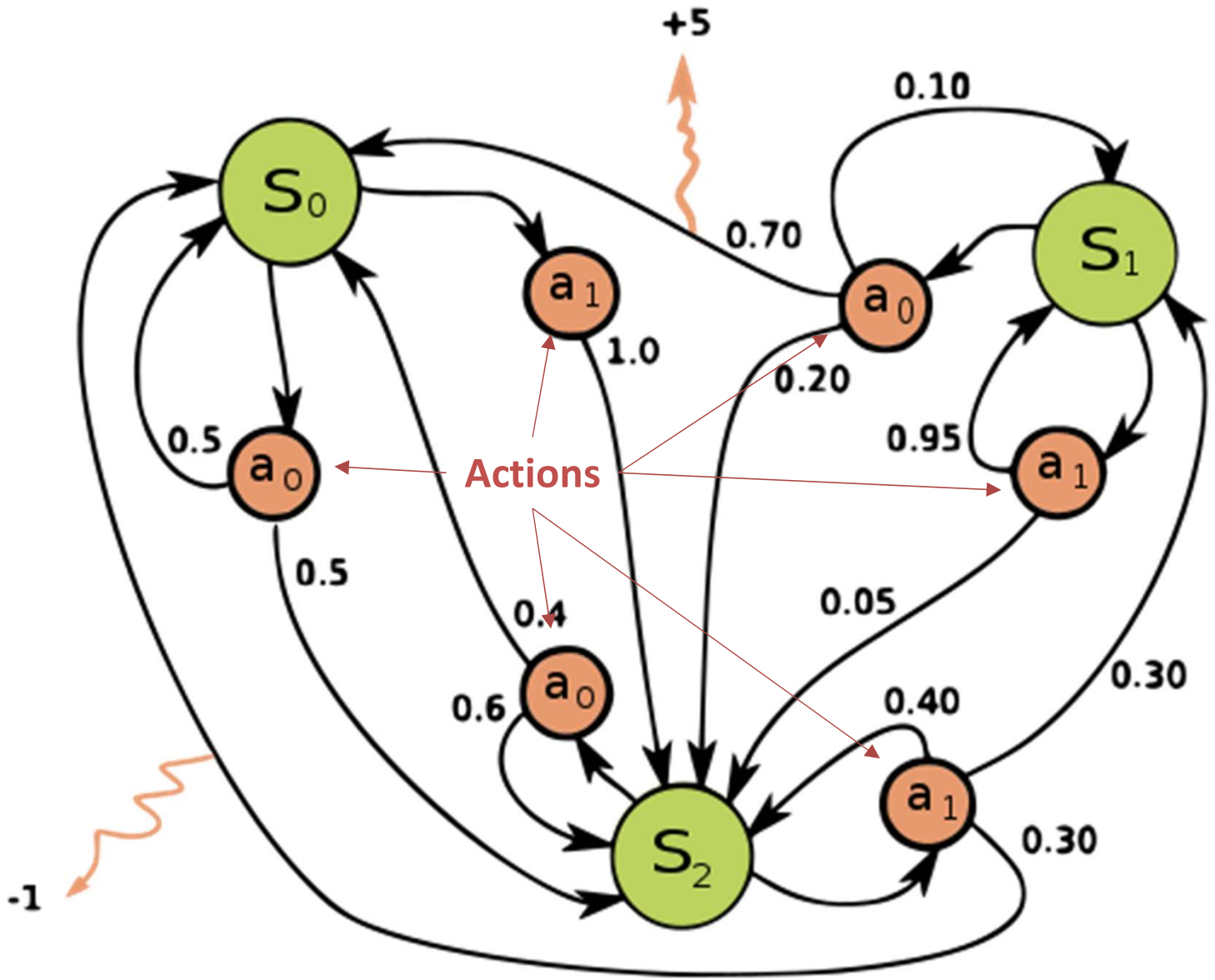
Source: Wikipedia

Markov Decision Process (MDP)



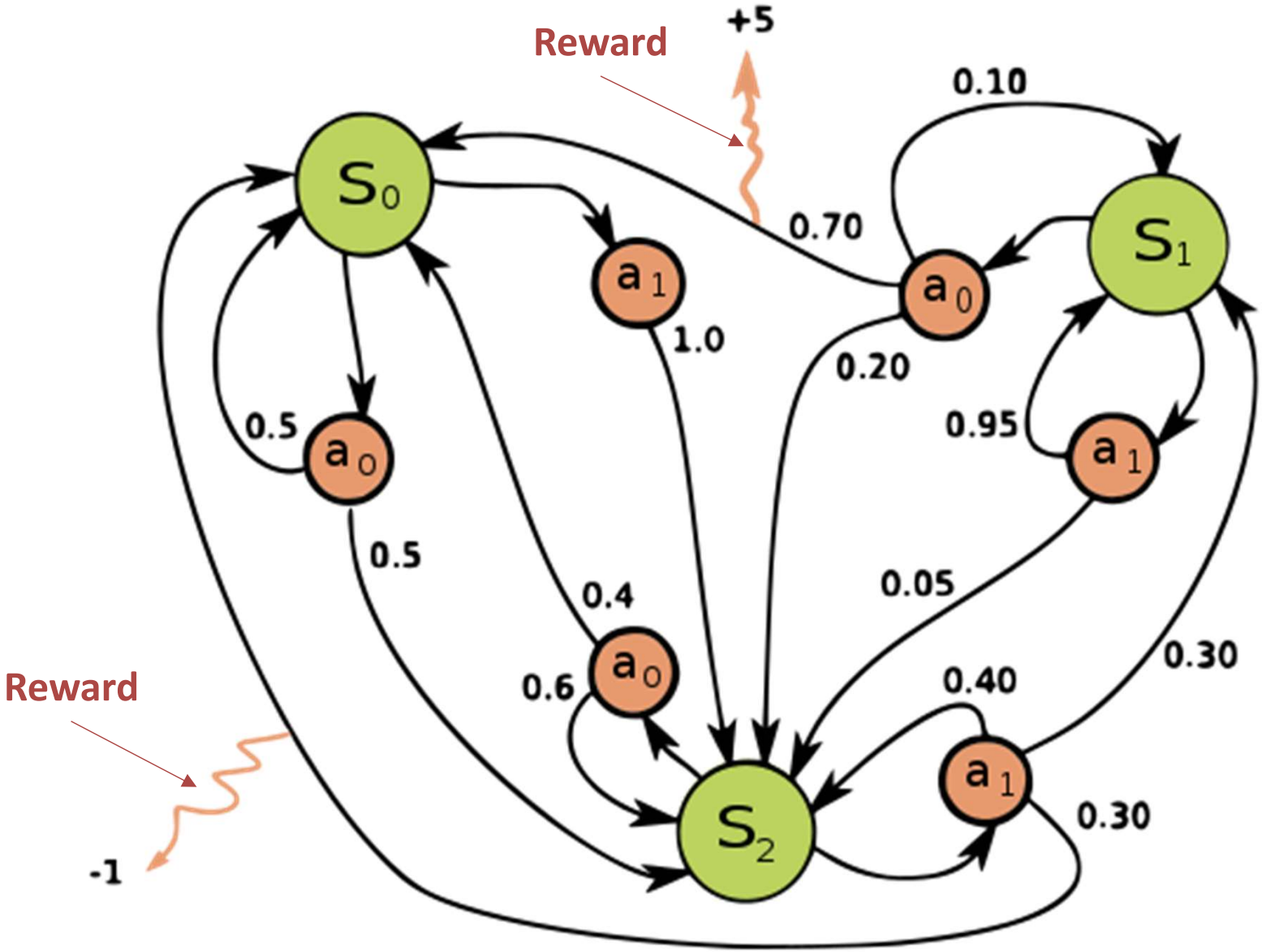
Source: Wikipedia

Markov Decision Process (MDP)



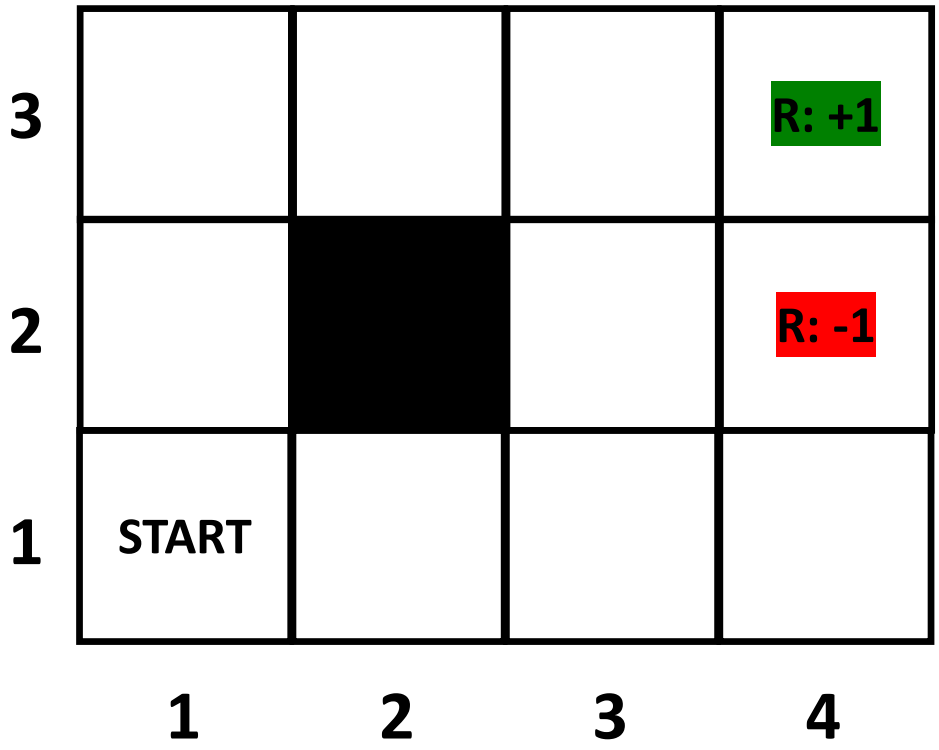
Source: Wikipedia

Markov Decision Process (MDP)

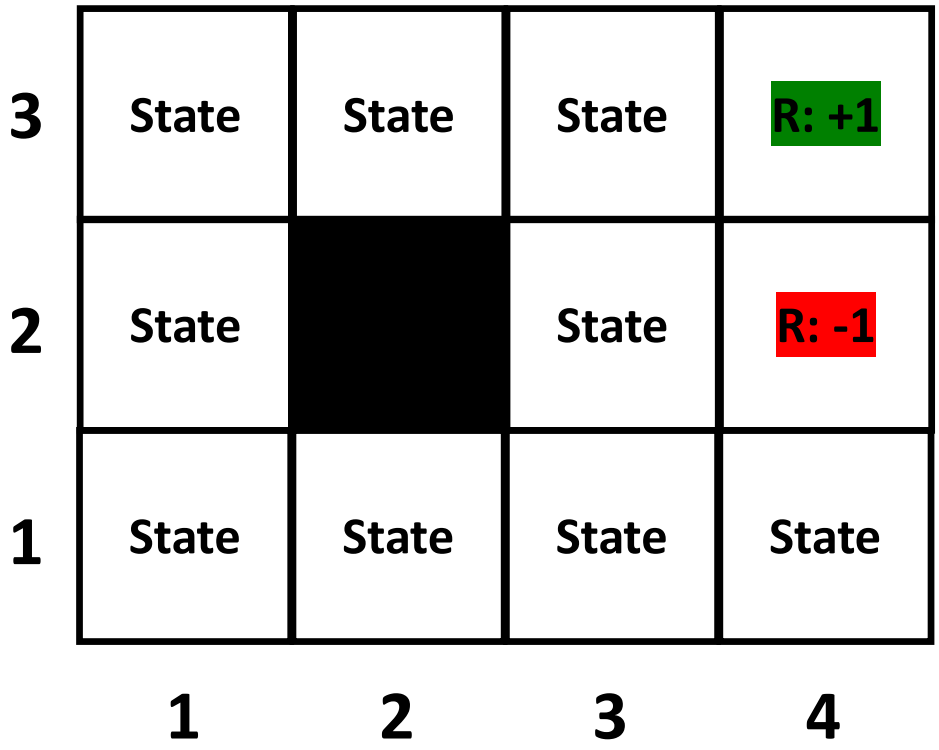


Source: Wikipedia

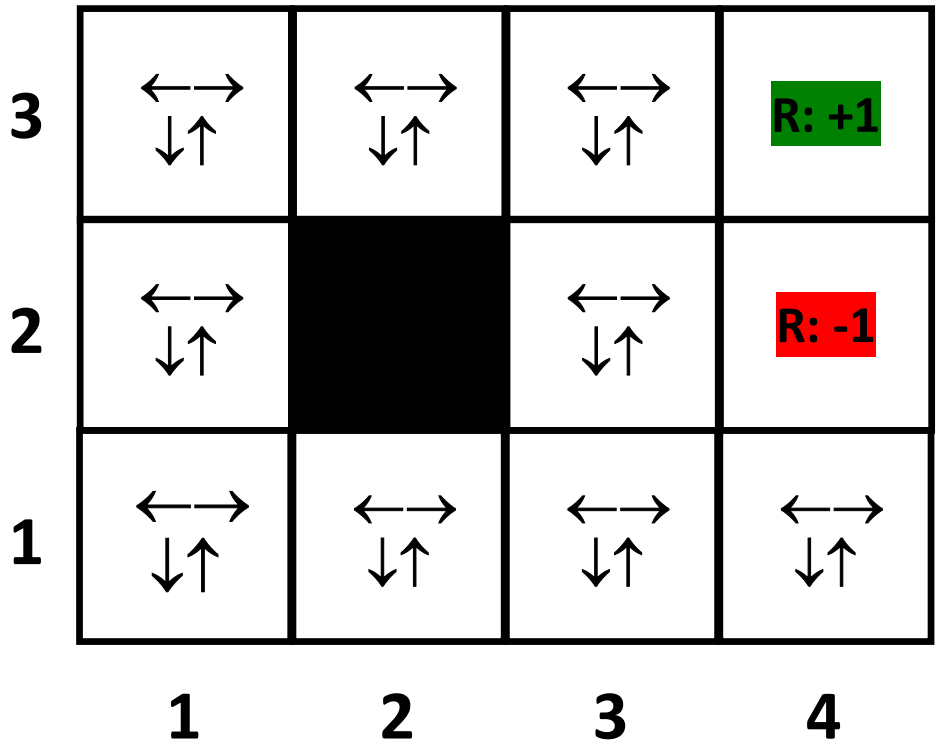
Fully Observable/Non-deterministic

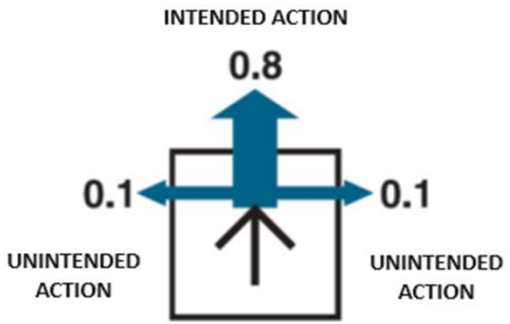


Sequential Decision Problem: States

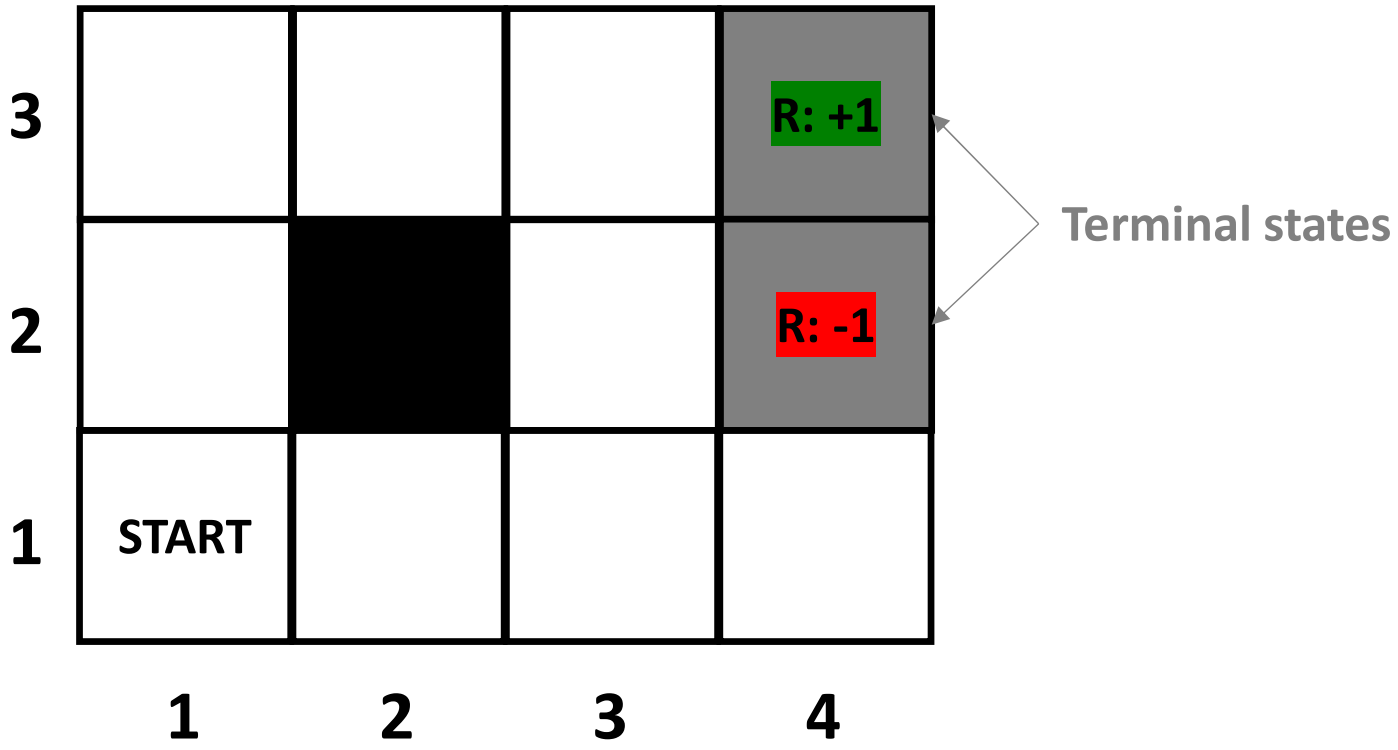


Sequential Decision Problem: Actions

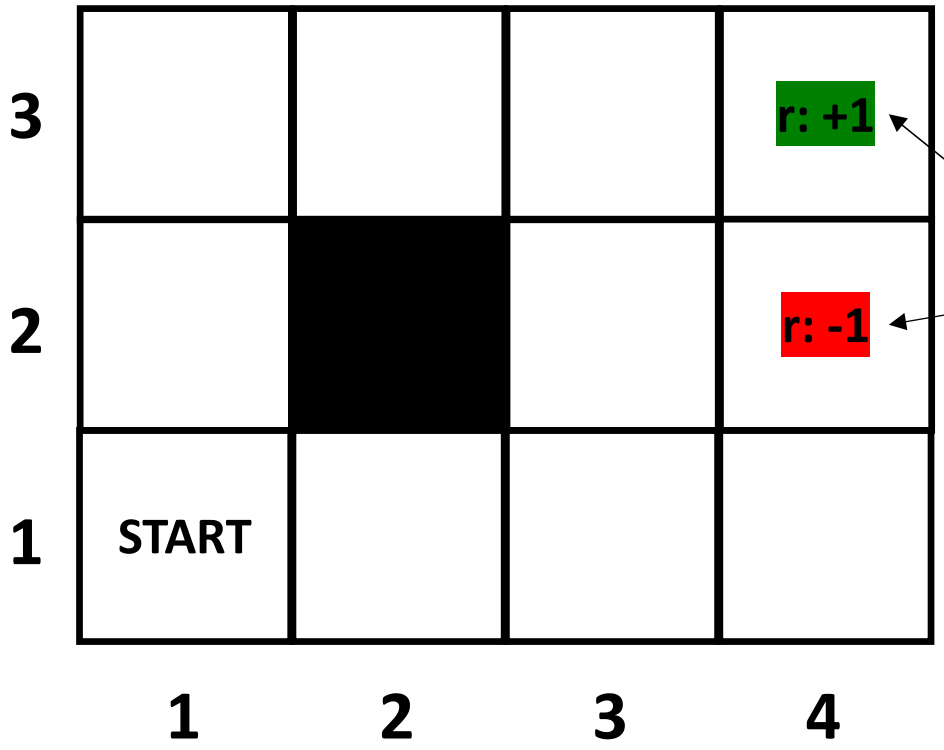




Sequential Decision Problem: Terminal



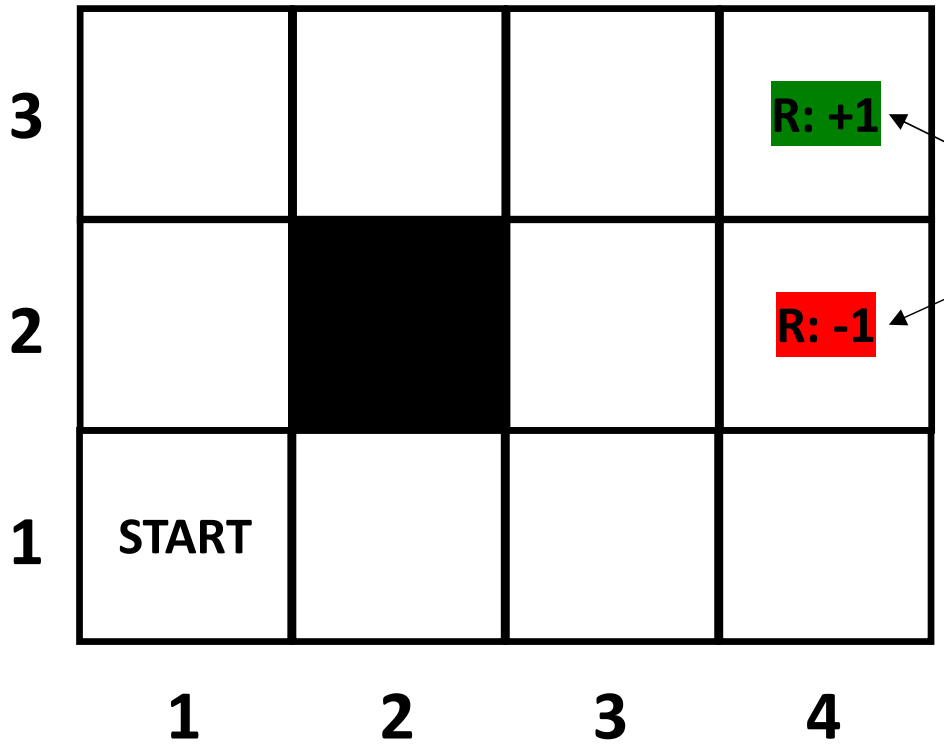
Sequential Decision Problem: Rewards



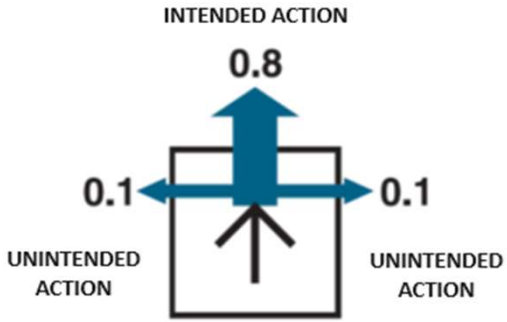
Terminal State rewards

Note: ALL other transitions have a reward r = -0.04

Fully Observable/Non-deterministic



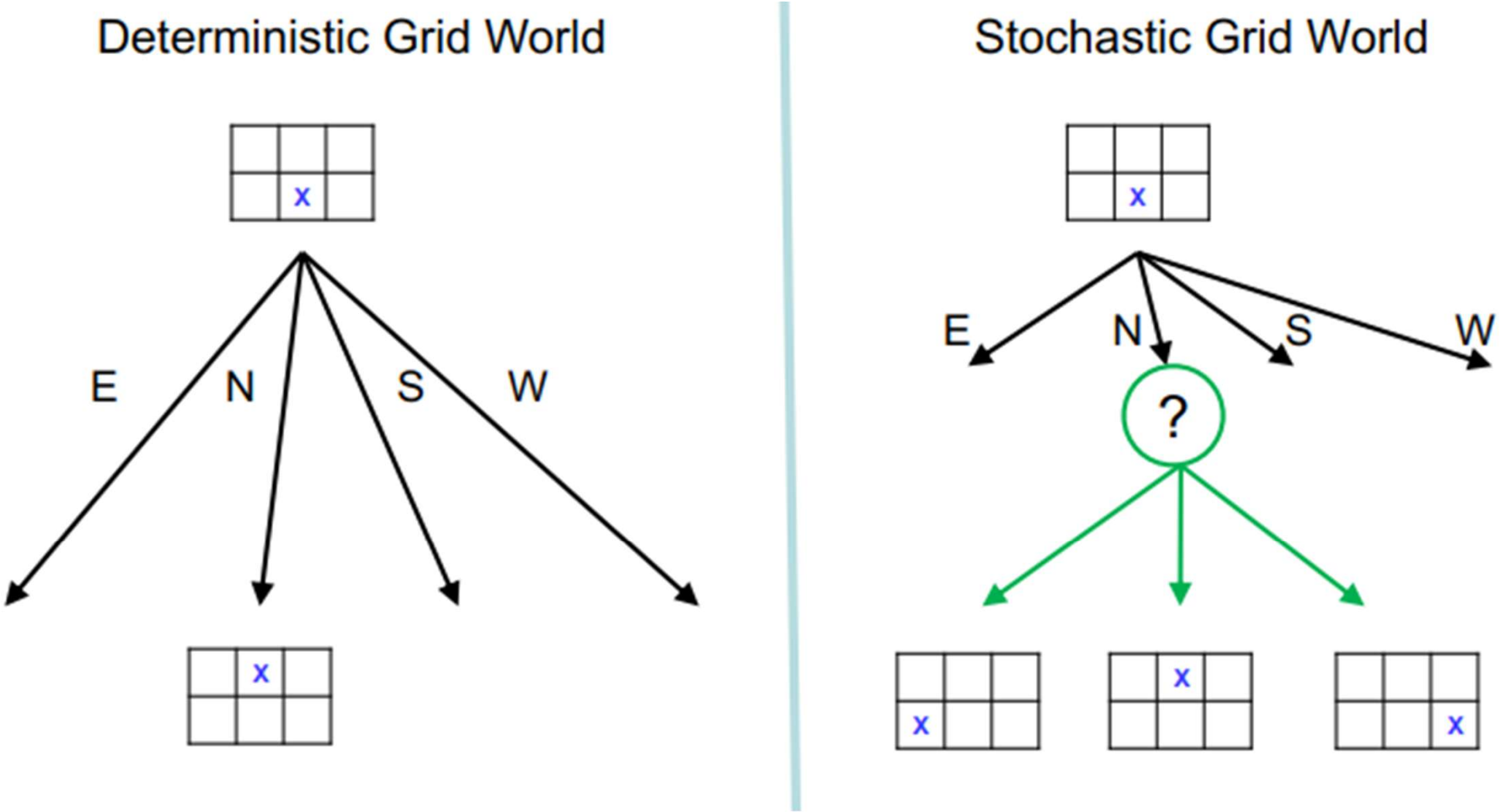
Note: ALL other transitions have a reward r = -0.04



Solution Approach

- A fixed action sequence is not the answer due to stochasticity
  - For example, [Up, Up, Right, Right, Right] is not a solution
  - It would be a solution if the environment was deterministic
- A solution must specify what the agent should do in any state that the agent might reach
  - This is called a **policy**
- Policy notation: $\boldsymbol{\pi}$
  - $\pi(s)$ specifies what action the agent should take at state s
- An **optimal policy** is the one that maximizes the expected utility → $\boldsymbol{\pi}^*$

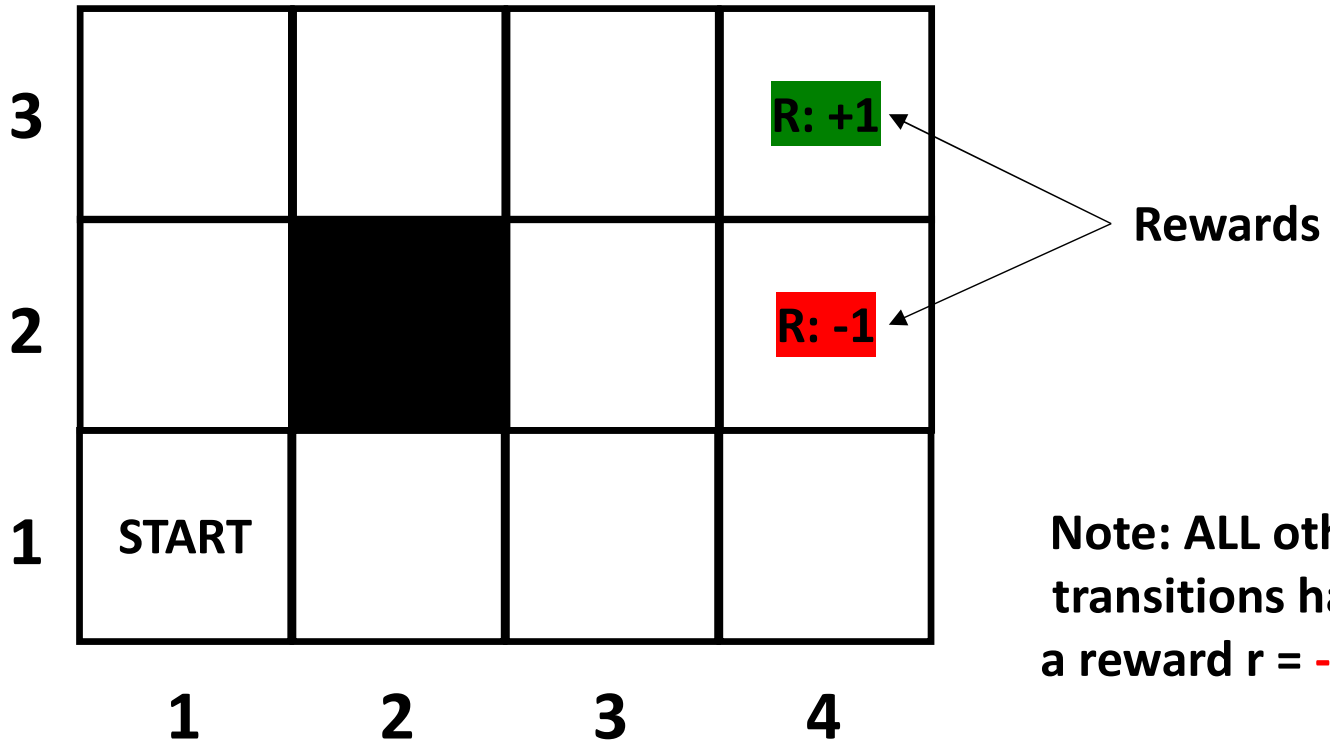
Plan vs. Policy [Solution]



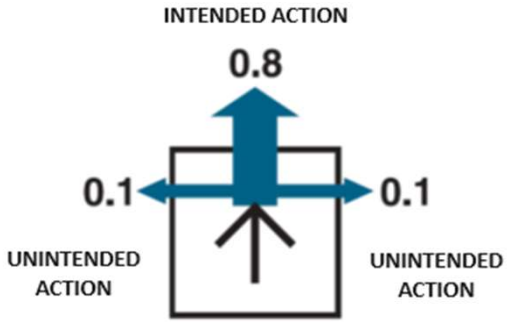
Notation

- $P(s' | s, a)$ Probability of arriving at state s' given we are at state s and take action a
- $R(s, a, s')$ The reward the agent receives when it transitions from state s to state s' via action a
- $\pi(s)$ The action recommended by policy π at state s
- π^* Optimal policy
- $U^{\pi}(s)$ The expected utility obtained via executing policy π starting at state s
- $U^{\pi^*}(s)$ is often abbreviated as $U(s)$
- $Q(s, a)$ expected utility of taking action a at state s
- γ Discount factor [0, 1]

What Are We After?



Note: ALL other transitions have a reward r = -0.04



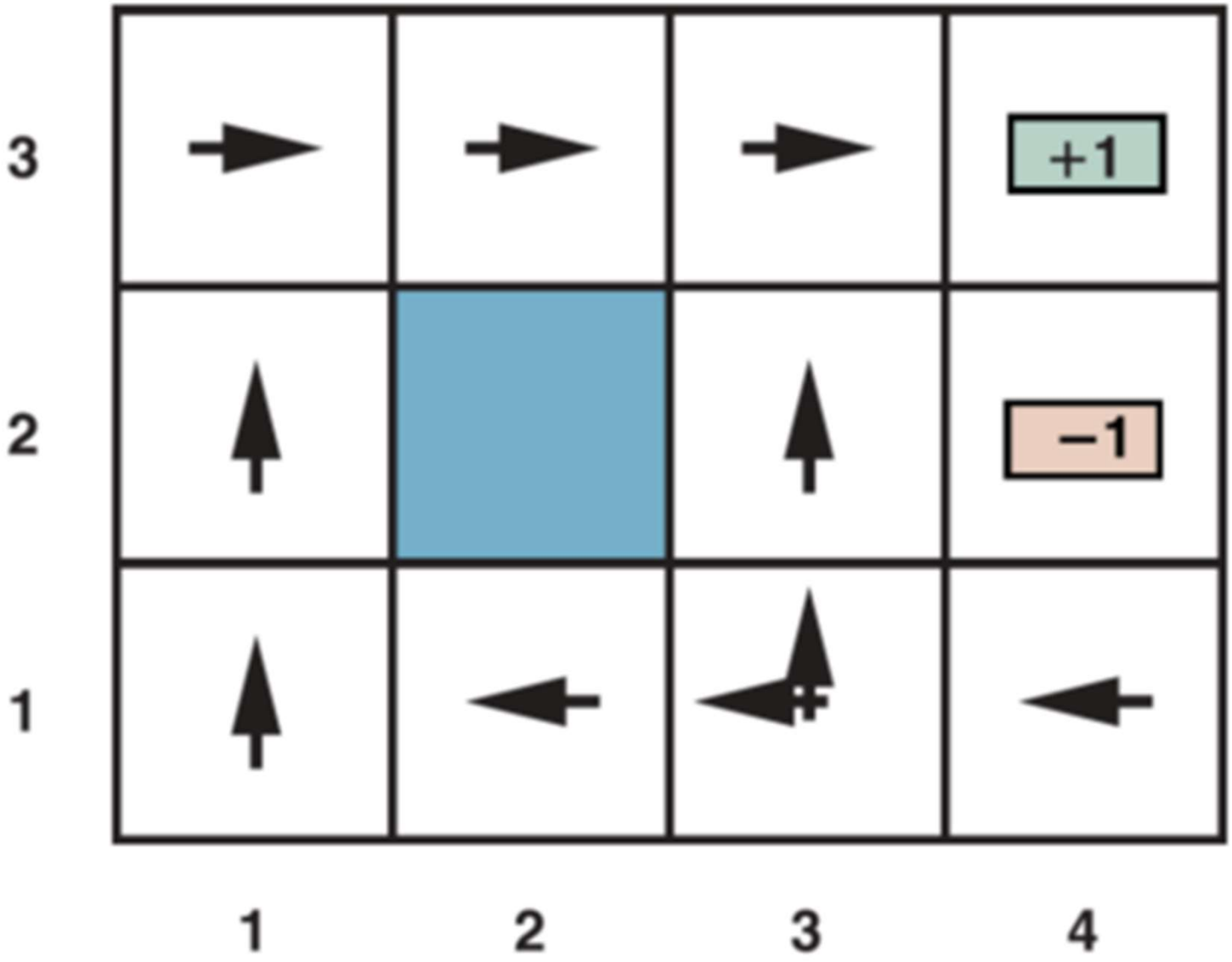
What Is the optimal policy π^*?

It **is** the mapping of actions to all reachable states that maximizes the expected utility

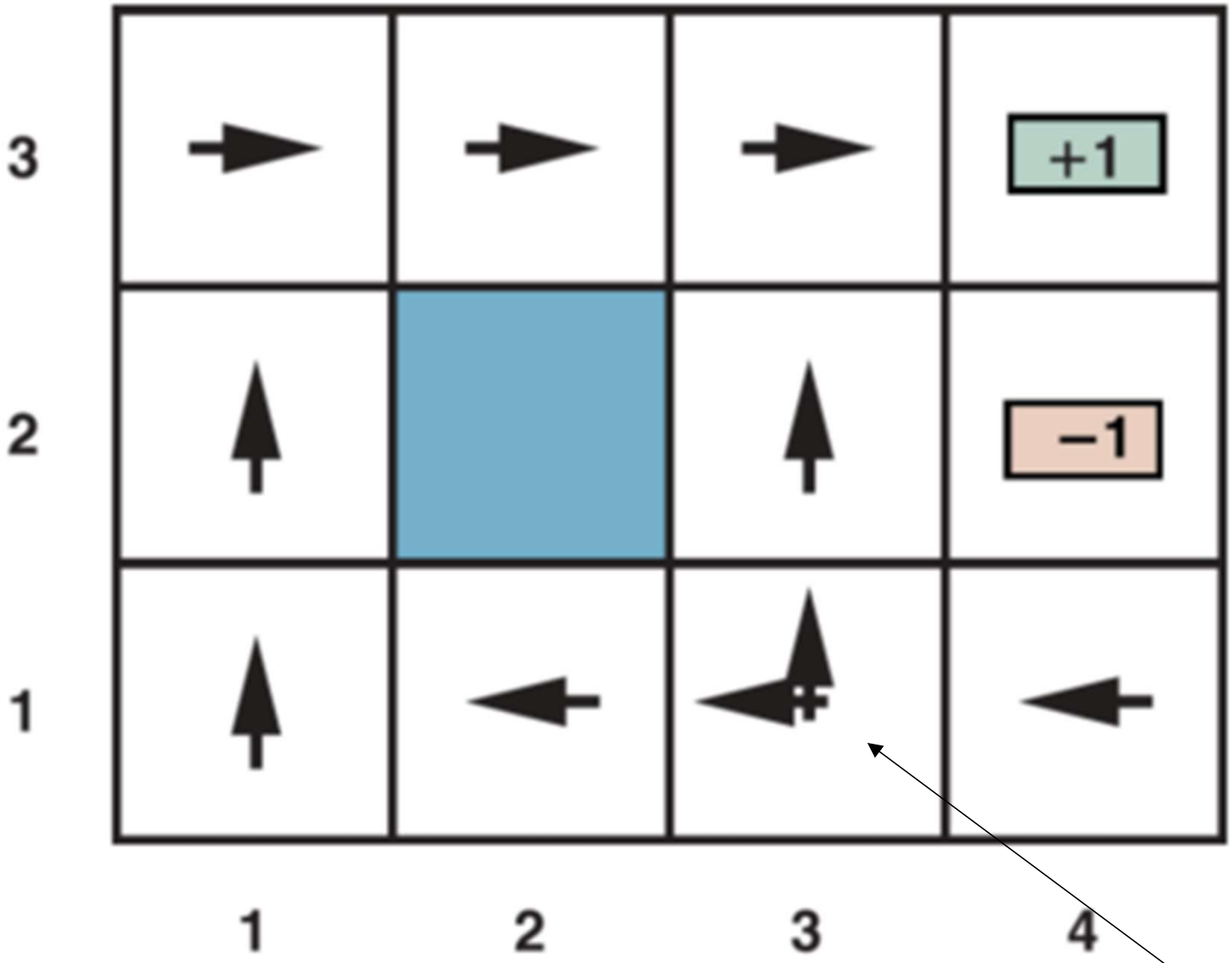
Solving MDPs

- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent

Optimal Policies: r = 0.04

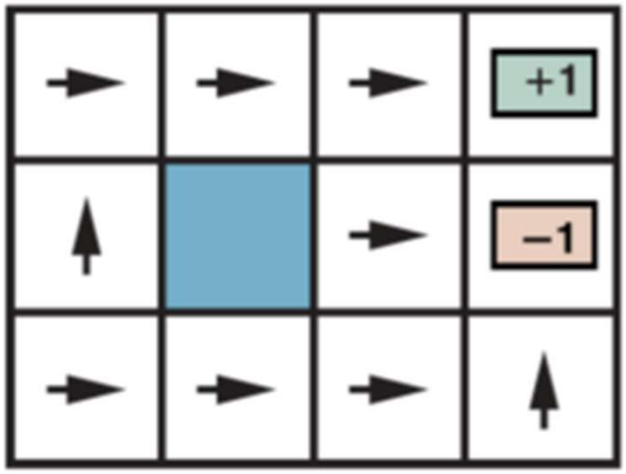


Optimal Policies: r = 0.04

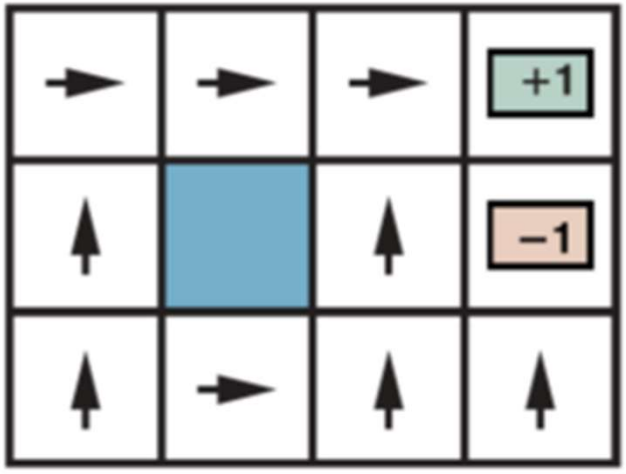


Note two EQUAL policies

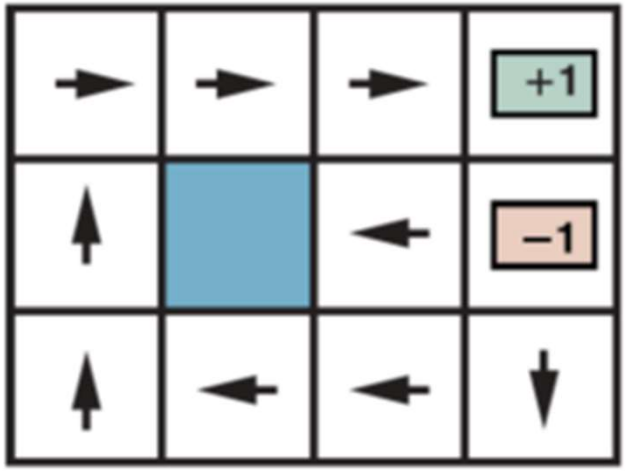
Optimal Policies: Different r Values



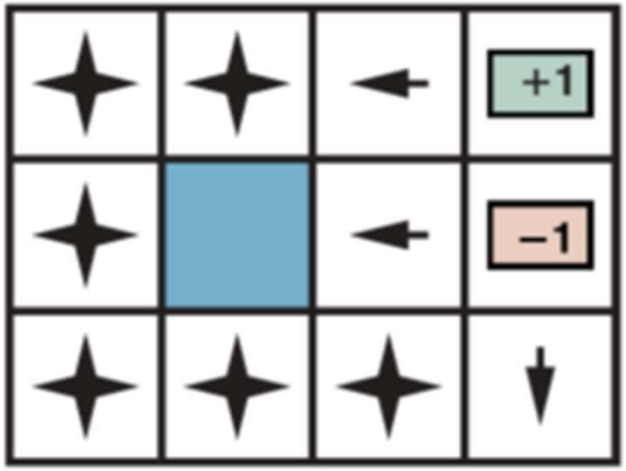
$r < -1.6497$



$-0.7311 < r < -0.4526$



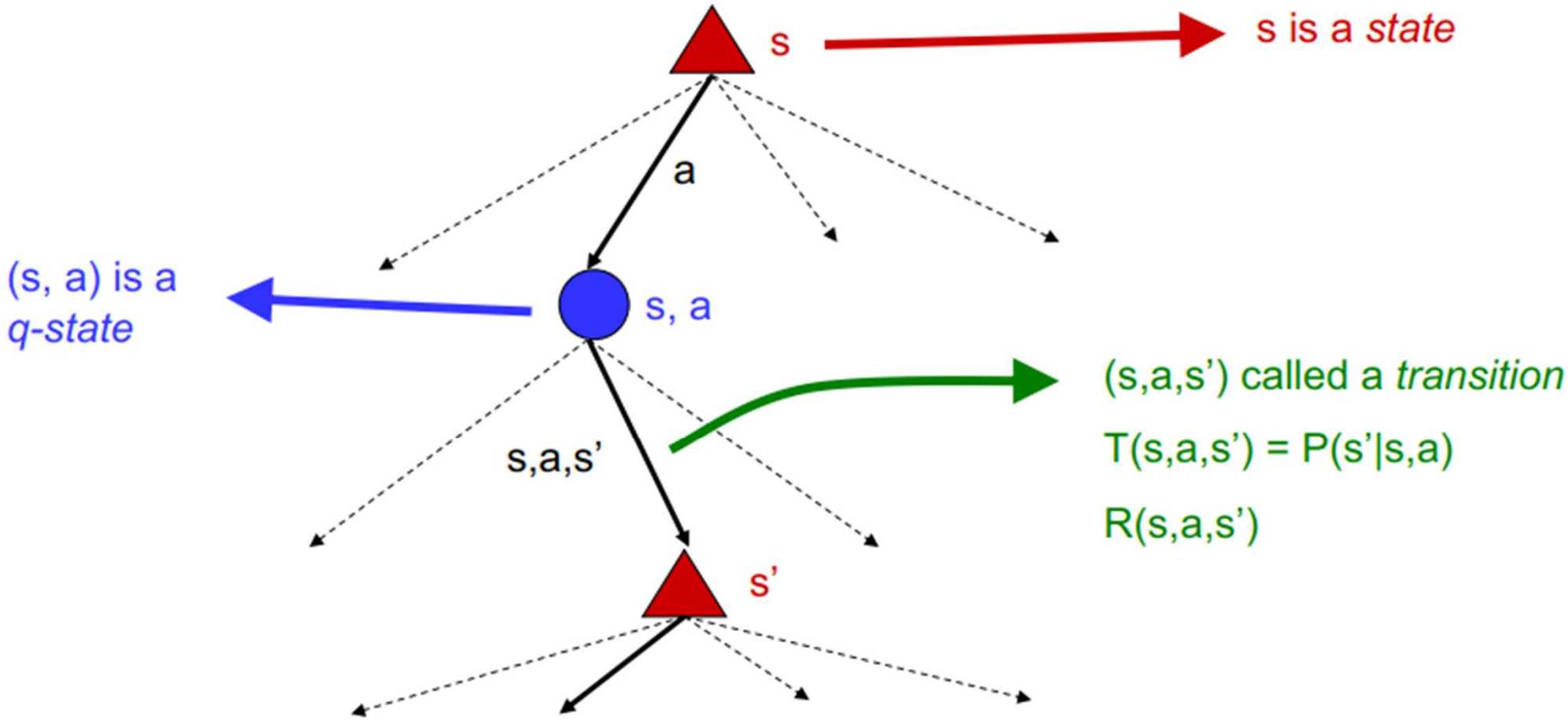
$-0.0274 < r < 0$



$r > 0$

Solve With Searching?

- Each MDP state gives an expectimax-like search tree



Utility of States

- The agent receives a reward at each state
- Utility of a state s given a policy π is the expected reward that the agent will get starting from state s and taking actions according to policy π
- Let S_t denote the state that the agent reaches at time t
- $U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})]$
- The expectation is with respect to the transition probabilities

Utility vs. Reward

- *R(s, a, s')* is the short-term immediate reward the agent receives when it transitions from state s to state s' via action a
- *U(s)* is the long-term cumulative reward from *s* onward
- $U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^{t} R(S_t, \pi(S_t), S_{t+1})]$

Utilities of Sequences of Rewards

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards
- Typically consider stationary preferences:

$$[r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots]$$

$$\Leftrightarrow$$

$$[r_0, r_1, r_2, \ldots] \succ [r'_0, r'_1, r'_2, \ldots]$$

Preference symbol

- Theorem: only two ways to define stationary utilities
  - Additive utility:

$$U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$$

  - Discounted utility:

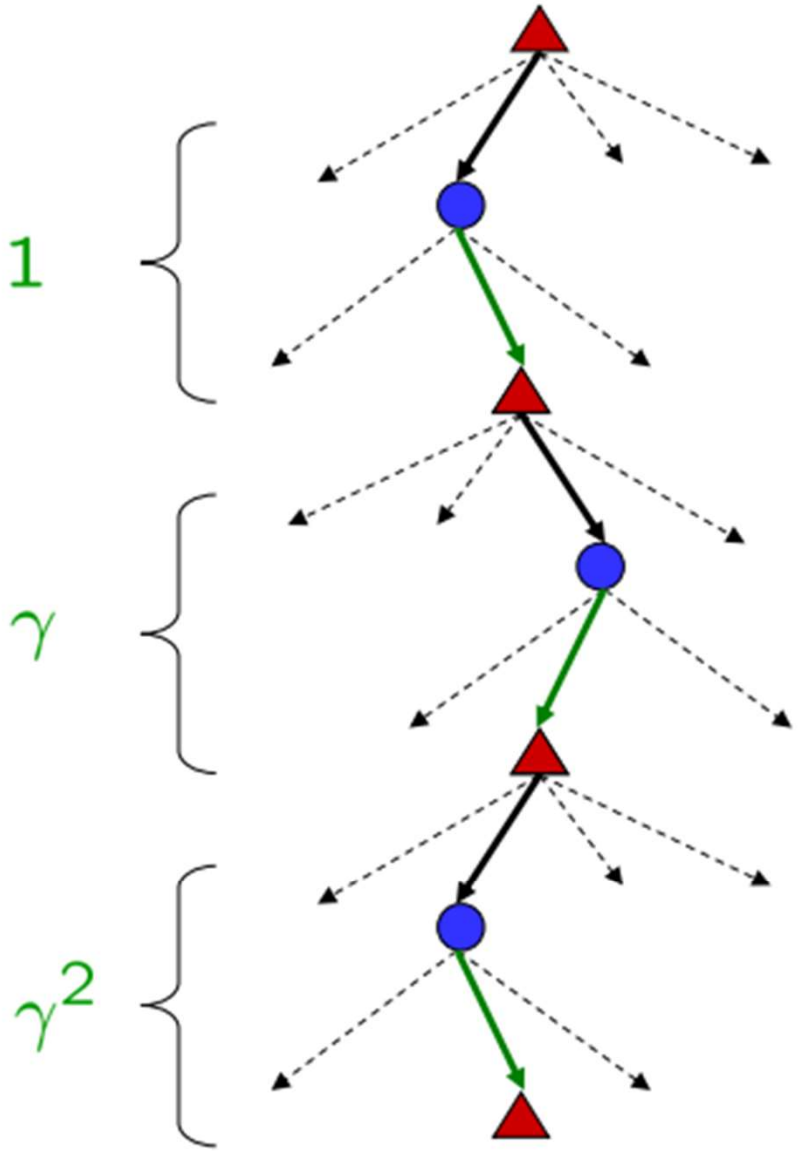
$$U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$$

Discount Factor γ

- The discount factor γ is a number between 0 and 1.
- The discount factor describes the preference of an agent for current rewards over future rewards.
- When γ is close to 0, rewards in the distant future are viewed as insignificant.
- When γ is 1, discounted rewards are exactly equivalent to additive rewards, so additive rewards are a special case of discounted rewards.
- Discounting appears to be a good model of both animal and human preferences over time.

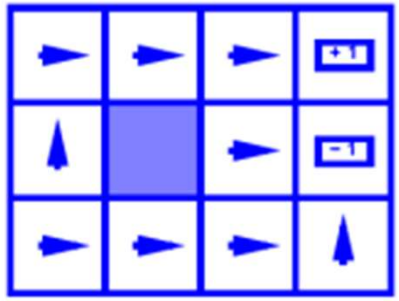
Discounting

- Typically discount rewards by $\gamma < 1$ each time step
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge



Infinite Utilities

- Problem: infinite state sequences have infinite rewards



- Solutions:
  - Finite horizon:
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies (π depends on time left)
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached
  - Discounting: for $0 < \gamma < 1$

$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1-\gamma)$$

    - Smaller γ means smaller "horizon" – shorter term focus

Infinite Discounted Sequence/Finite U

With discounted rewards, the utility of an infinite sequence is finite. In fact, if $\gamma < 1$ and rewards are bounded by ± Rmax, we have

$$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = R_{\max}/(1-\gamma)$$

using the standard formula for the sum of an infinite geometric series

Proper Policy

If the environment contains terminal states and if the agent is guaranteed to get to one eventually, then we will never need to compare infinite sequences.

A policy that is guaranteed to reach a terminal state is called a proper policy. With proper policies, we can use γ = 1 (i.e., additive rewards)

Proper Policy

If the environment contains terminal states and if the agent is guaranteed to get to one eventually, then we will never need to compare infinite sequences.

A policy that is guaranteed to reach a terminal state is called a proper policy. With proper policies, we can use γ = 1 (i.e., additive rewards)

Average Reward [Infinite Sequences]

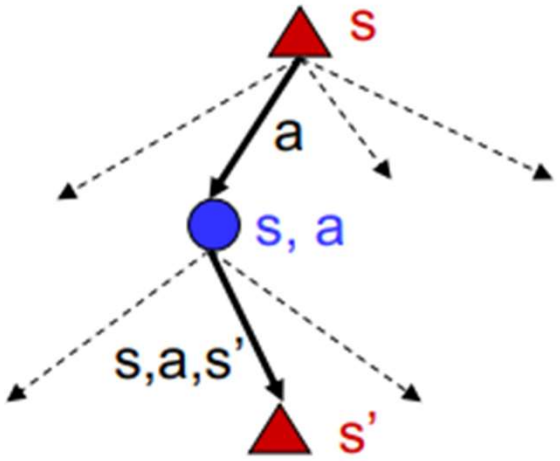
Infinite sequences can be compared in terms of the average reward obtained per time step.

Suppose that square (1,1) in the 4 x 3 world has a reward of 0.1 while the other nonterminal states have a reward of 0.01.

Then a policy that does its best to stay in (1,1) will have higher average reward than one that stays elsewhere.

MDP Recap

- Markov decision processes:
  - States S
  - Start state s_0
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)



- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility (or return) = sum of discounted rewards

Bellman Equation

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

Bellman Optimality Equation

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

Bellman Equation

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