Methods of Artificial Intelligence

5. Planning II: Probabilistic Planning

Nohayr Muhammad

Winter Term 2022/2023

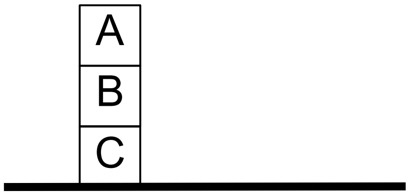
November 30th, 2022

Last time..

- Classical Planning
- Find sequence of actions that transforms given state into goal state
- An action “guarantees” reaching certain states
- Situation Calculus
- STRIPS
- PDDL

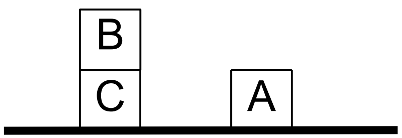
Example

S1:



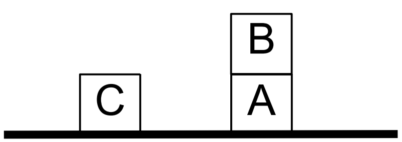
- onTable(C, S1)
- on(B,C, S1)
- on(A,B, S1)
- clear(A, S1)

put_on_table(A,S1)



- onTable(C, put_on_table(A,S1))
- on(B,C, put_on_table(A,S1))
- onTable(A, put_on_table(A,S1))
- clear(A, put_on_table(A,S1))
- clear(B, put_on_table(A,S1))

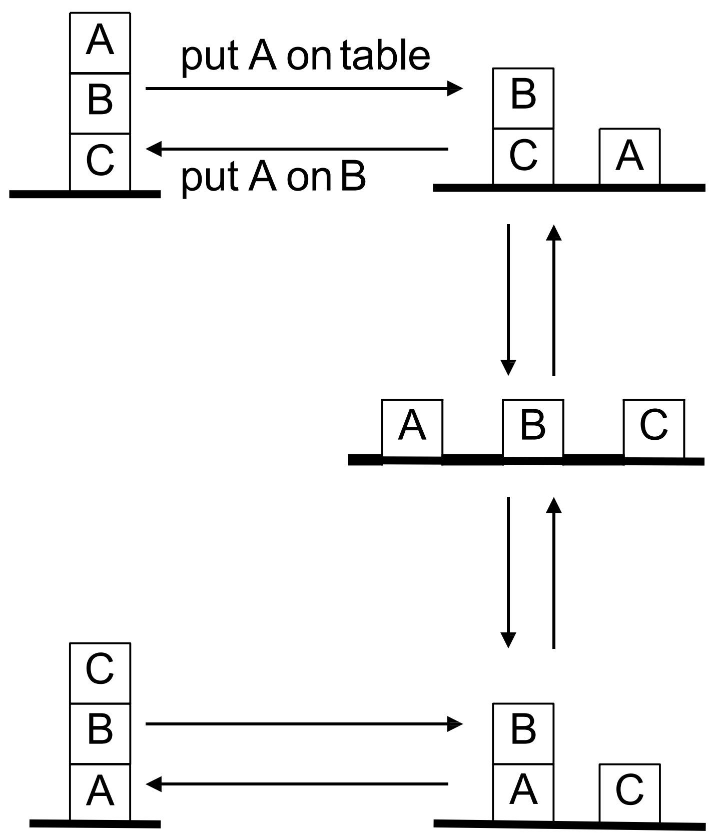
put(B,A, put_on_table(A,S1))



- onTable(C, put(B,A, put_on_table(A,S1)))
- on(B,A, put(B,A, put_on_table(A,S1)))
- onTable(A, put(B,A, put_on_table(A,S1)))
- clear(B, put(B,A, put_on_table(A,S1)))
- clear(C, put(B,A, put_on_table(A,S1)))

Example: State-Space Model

- Operator application spans a search space
- s_i is connected to s_j iff s_j results from s_i when applying a single operator

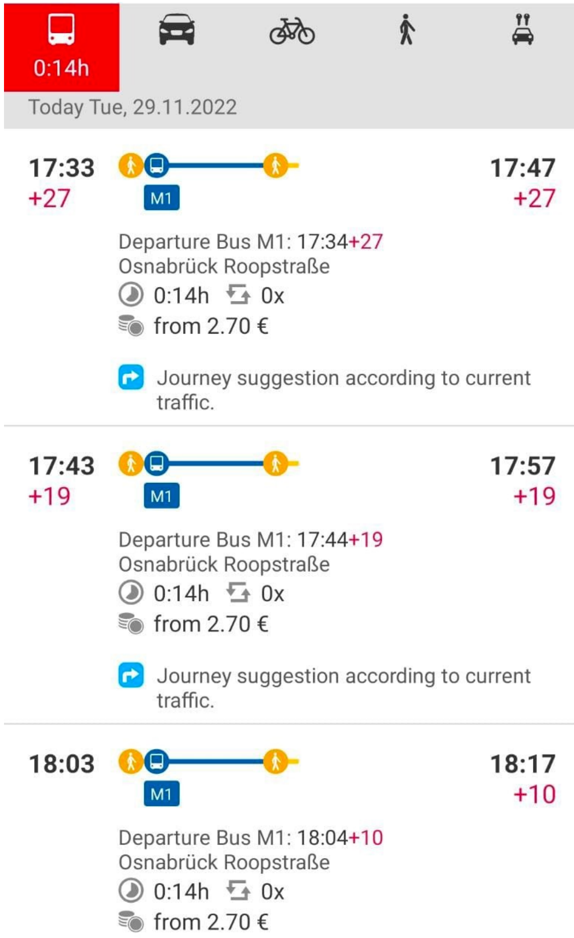


Today..

- Probabilistic Planning
- Dealing with uncertainty
- Dealing with long-term consequences
- Markov Decision Processes (MDPs)

Introduction

Uncertainty in Planning



Uncertainty in Planning

- In many situations, we have to deal with uncertainty:
  - Taking a particular road is often fast, but sometimes there is a traffic jam
  - An order is usually delivered within two days, but can be delayed due to labor strike or logistical mistakes
  - When navigating a robot, sensor information is inherently uncertain

Long-term Consequences

- Furthermore, actions often have long-term consequences
  - Not recharging the battery saves time now
    - but we may run out of energy later
  - Canceling insurance increases our budget now
    - but we may lose a lot of money later
  - Extending maintenance intervals may decrease spendings now
    - but may result in business interruptions due to technical failures later

Overview: Probabilistic Planning

- Markov Decision Processes (MDP)
  - Basics
  - The Grid World Example
    - A Sample Run
  - Policies and Rewards
  - Computing Discounted Rewards
  - Playing with the Rewards
- Policy Iteration Algorithm
  - Policy Evaluation
  - An Example of Policy Evaluation
  - Policy Improvement

Markov Decision Processes

Basics

What is Markov about MDPs?

"Markov" generally means that given the present state, the future and the past are independent

For Markov decision processes, "Markov" means action outcomes depend only on the current state

Andrey Markov
(1856-1922)

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

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

Markov Decision Processes

- Markov Decision Processes (MDPs) take account of both mentioned problems:
  - uncertainty
  - long-term consequences
- Intuitively, MDPs describe
  - probability of state changes caused by actions
  - short-term rewards of actions in particular states
- A policy determines in what state we choose what action
- We evaluate policies by their expected reward with respect to
  - uncertainty of outcome of actions and
  - future rewards

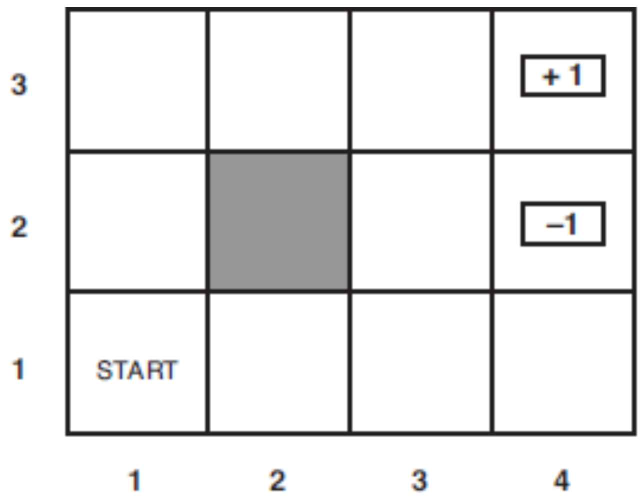
MDPs: Formal Definition

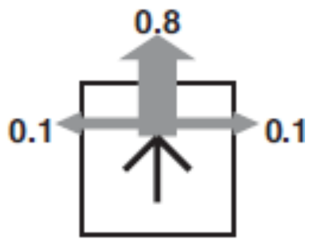
- a Markov Decision Process is a tuple (S, A, p, r), where
  - S is a set of states
  - A is a set of actions
  - p is the state transition probability function

    p(s' | s, a): probability that performing action a in state s yields state s'
  - r is the reward function

    r(s, a): short-term benefit of performing action a in state s

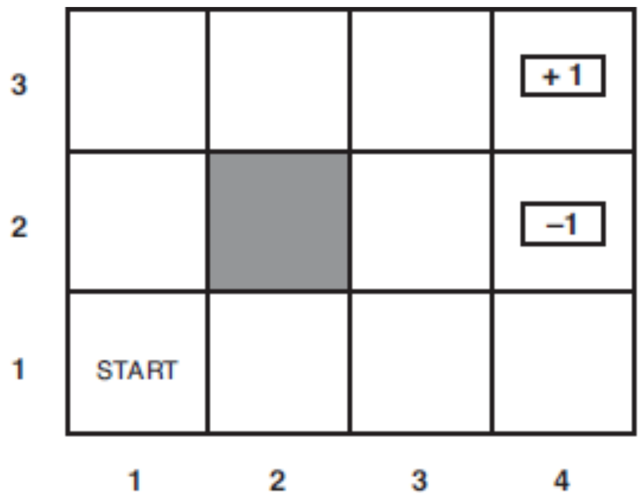
Example: Grid World

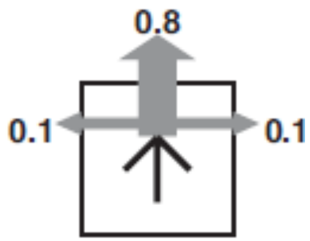




- Move agent from Start to one of the goal fields
- Agent can move in four directions with uncertain outcome
- Game ends when agent performs an arbitrary move to one of the goal fields (special terminal states)

Example: States and Actions





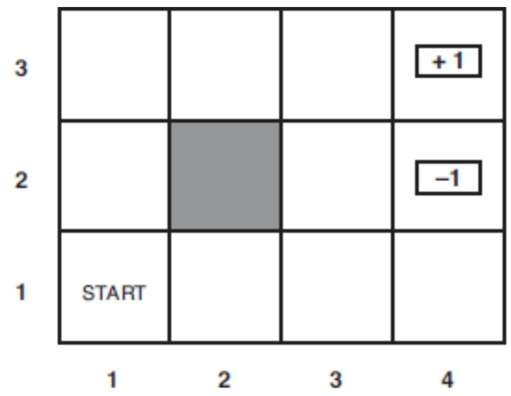
- States encode agent's current position

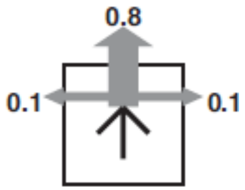
  S = { (1,1), …, (4,1), (1,2), …, (4,2), (1,3), …, (4,3) }

- Actions encode agent's possible actions

  A = { up, down, left, right}

Example: Transition Probabilities

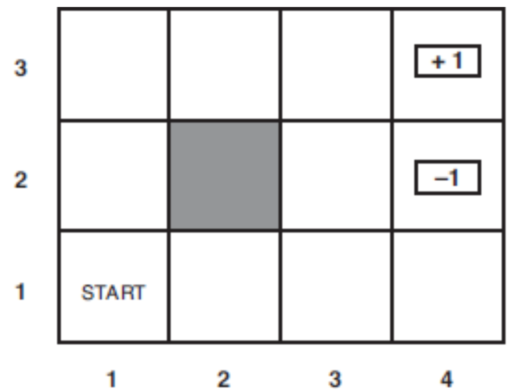


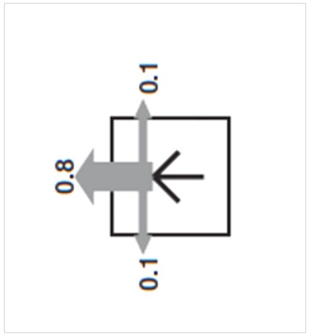


- 20% chance for a transmission error that will drive robot in wrong direction
- Transition function p: when performing **action a**, with probability
  - 0.8, we will move in the intended direction
  - 0.1, we will move in direction 90° clockwise to intended direction
  - 0.1, we will move in direction 90° counterclockwise to intended direction

Unless action leads out of grid world or to an obstacle
– in this case, nothing happens (robot is blocked by wall)

Example: Transition Probabilities

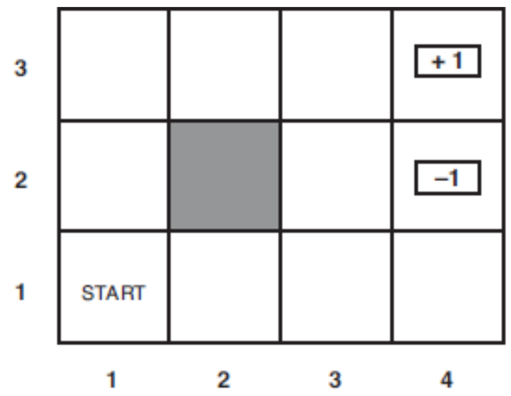


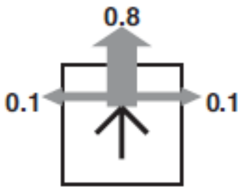


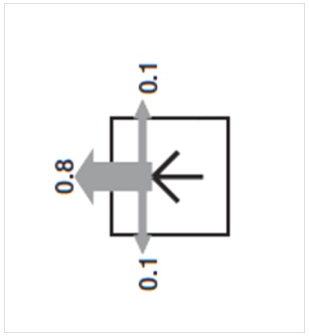
- p: when performing **left**, with probability
  - 0.8, we will move left
  - 0.1, we will move up
  - 0.1, we will move down

Unless action leads out of grid world – in this case, nothing happens

Example: Transition Probabilities

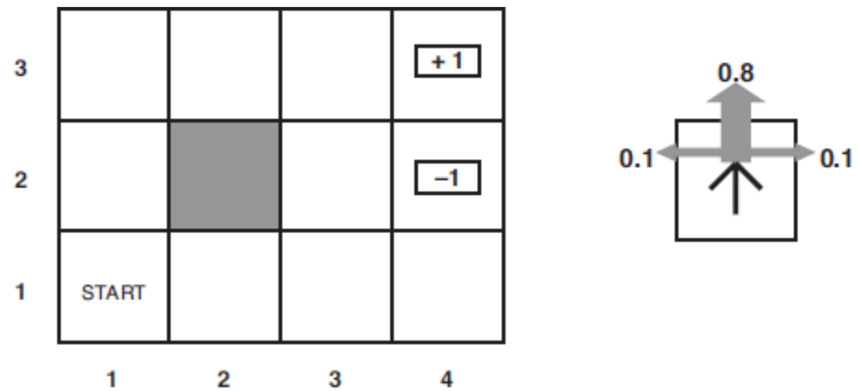






- p((1,2) | (1,1), up) = 0.8
- p((1,1) | (1,1), left) = 0.8 (left) + 0.1 (down) = 0.9
- p((1,3) | (1,1), up) = 0
- p((2,1) | (1,1), up) = 0.1

Example: Rewards



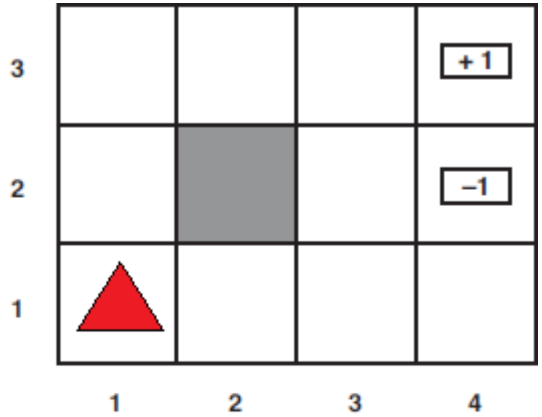
- Rewards give incentive to reach desired goal state
- r(s, a) = -0.04 for all actions a and all states s ≠ (4,2), s ≠ (4,3)

  (each action that does not finish game, results inpenalty)

- r((4,2), a) = -1 for all actions a
- r((4,3), a) = 1 for all actions a

Grid World Example: A sample run

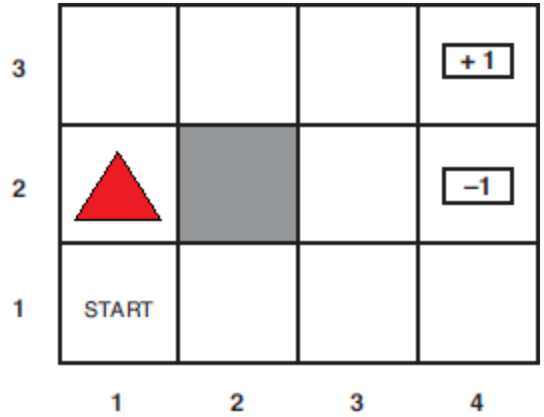
Example: Sample Run



Cumulative Reward: 0

Example: Sample Run

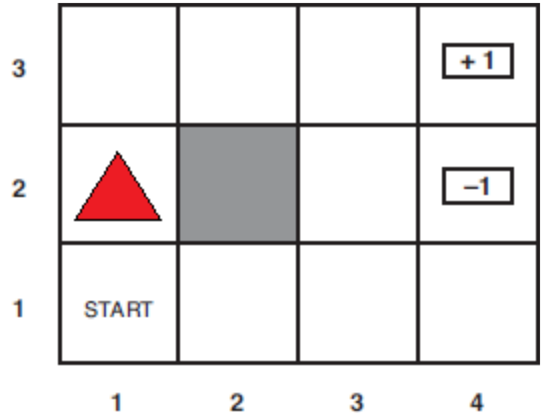
move up (works as intended)



Cumulative Reward: -0.04

Example: Sample Run

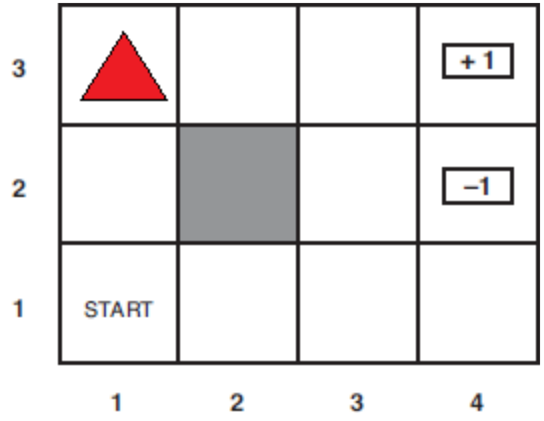
move up (robot moves right instead)



Cumulative Reward: -0.08

Example: Sample Run

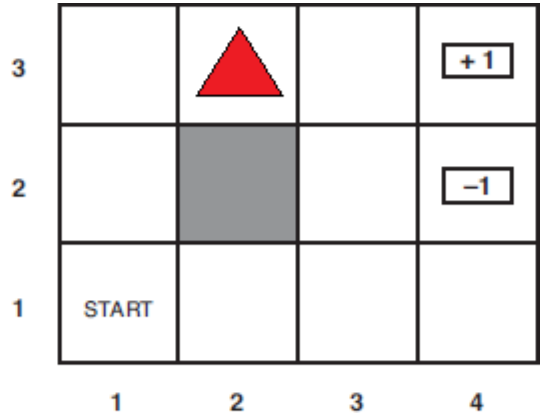
move up (works as intended)



Cumulative Reward: -0.12

Example: Sample Run

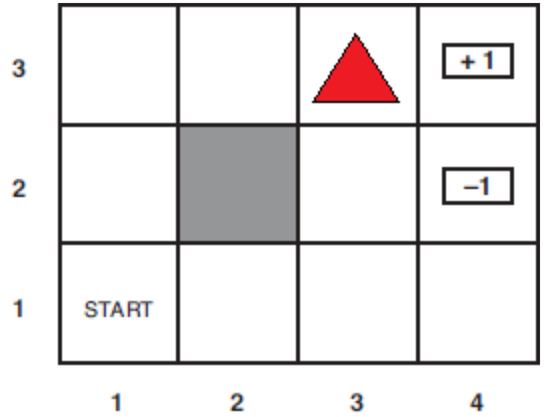
move right (works as intended)



Cumulative Reward: -0.16

Example: Sample Run

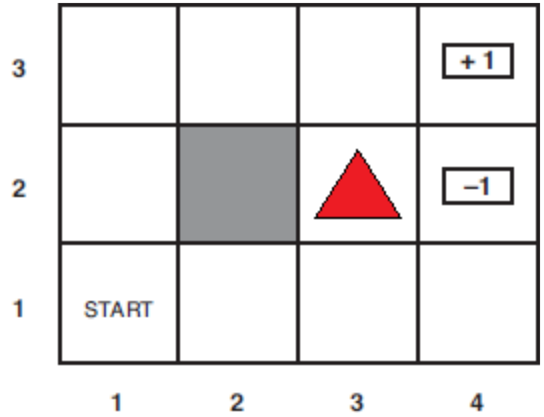
move right (works as intended)



Cumulative Reward: -0.2

Example: Sample Run

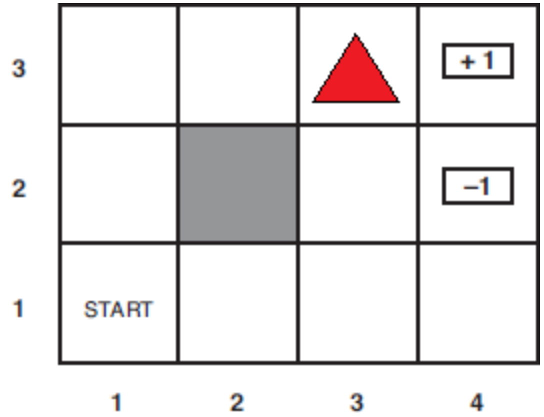
move right (robot moves down instead)



Cumulative Reward: -0.24

Example: Sample Run

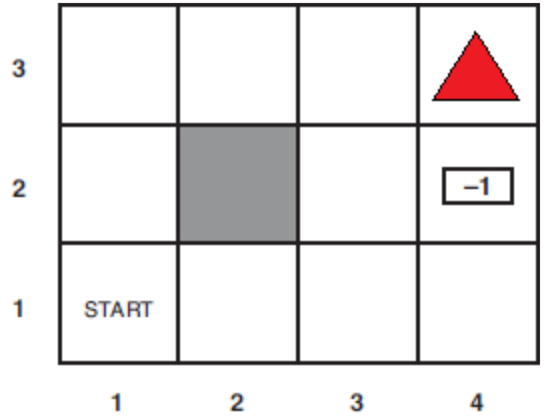
move up (works as intended)



Cumulative Reward: -0.28

Example: Sample Run

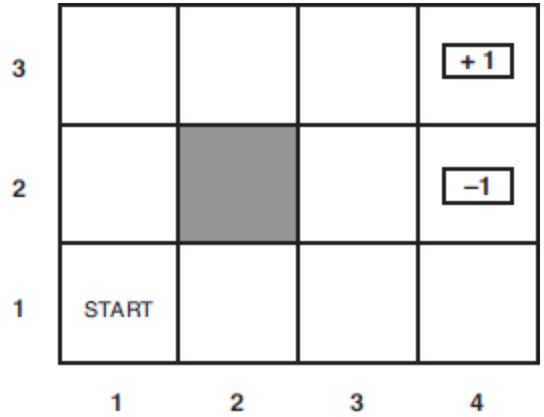
move right (works as intended)



Cumulative Reward: -0.32

Example: Sample Run

move right (action does not matter – game ends)



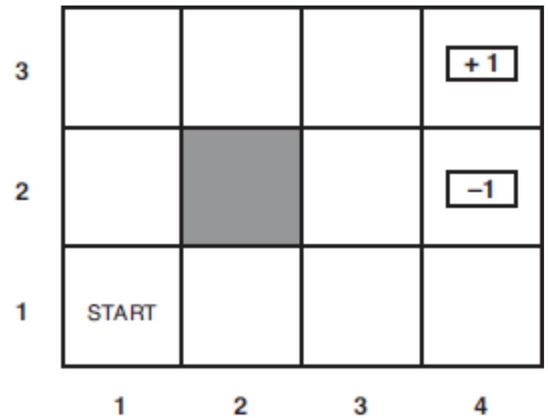
Cumulative Reward: **0.68**

Markov Decision Processes

Policies and Rewards

Policies

- a (deterministic Markov) Policy assigns an action to each state
- Formally, a policy is a mapping π from S to A



- $\pi(\ (1,1)\ ) = up$
- $\pi(\ (1,2)\ ) = up$
- …

Expected Rewards

- When following policy π, we can compute the probability $P(S_k = s)$ of being in state s in the k-th step

- If we have states $s_1, \ldots, s_n$, the expected reward is

$$E_\pi[r_k] = P(S_k = s_1) * r(s_1, \pi(s_1)) + \ldots + P(S_k = s_n) * r(s_n, \pi(s_n))$$

- The expected reward of policy π is given by the series

$$E_\pi[r_1] + E_\pi[r_2] + E_\pi[r_3] + E_\pi[r_4] + \ldots$$ (possibly infinite series)

γ-discounted Rewards

- The expected reward of policy π is given by the series

  $E_\pi[r_1] + E_\pi[r_2] + E_\pi[r_3] + E_\pi[r_4] + \ldots$ (possibly infinite series)

- It may be reasonable to discount future rewards
  - because future rewards may be less valuable (economics) or
  - to guarantee convergence of the series

- The γ-discounted reward of policy π is given by the series

  $E_\pi[r_1] + \gamma * E_\pi[r_2] + \gamma^2 * E_\pi[r_3] + \gamma^3 * E_\pi[r_4] + \ldots$ where γ is a real number between 0 and 1

An Example

- The γ-discounted reward of policy π is given by the series

  $E_\pi[r_1 + \gamma * r_2 + \gamma^2 * r_3 + \gamma^3 r_4 + \ldots]$

  where γ is a real number between 0 and 1

- **Which are the first four terms for γ=0.5?**

  r_1 , $0.5 * r_2$, $0.25 * r_3$, $0.125 * r_4$

  (weight of rewards halves with every step in the future)

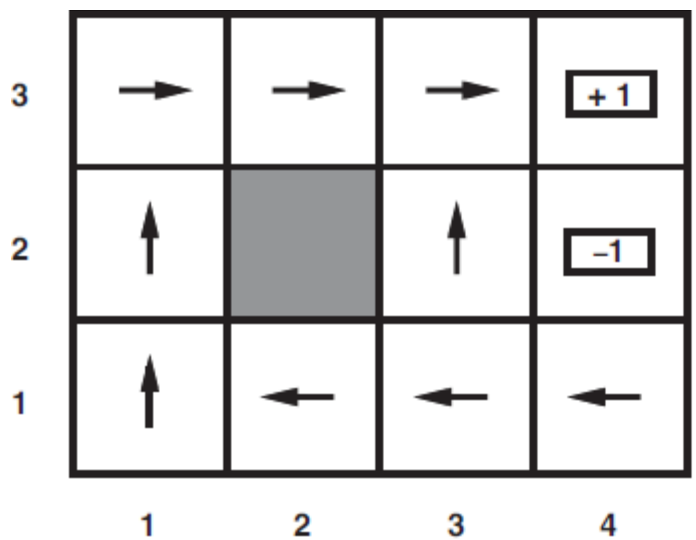
Playing with the Rewards

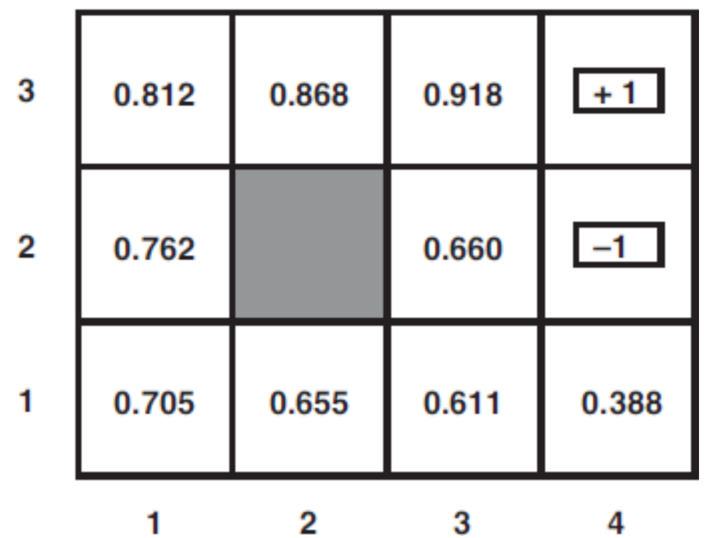
Computing γ-discounted Rewards

- We can compute an optimal policy with respect to the γ-discounted rewards using different approaches
  - Policy Iteration
  - Value Iteration
  - Linear Programming
- Worst-case runtime of all approaches is polynomial in the number of states and actions

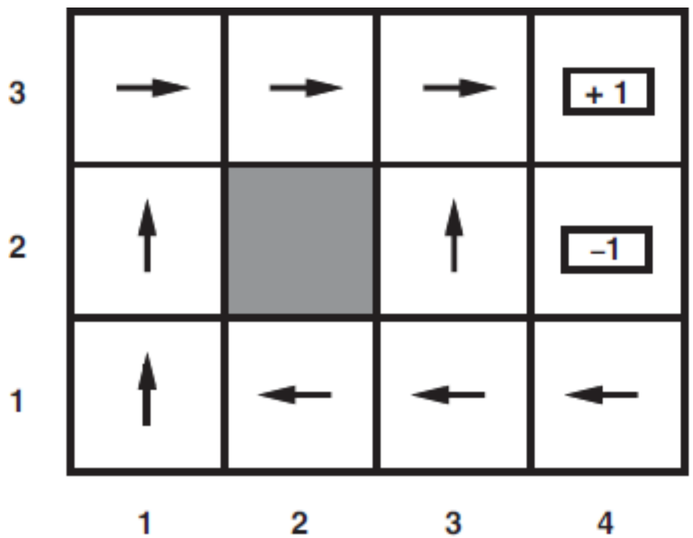
Example: Grid World

- The following picture shows optimal policy and expected values (γ=1) when starting from different states





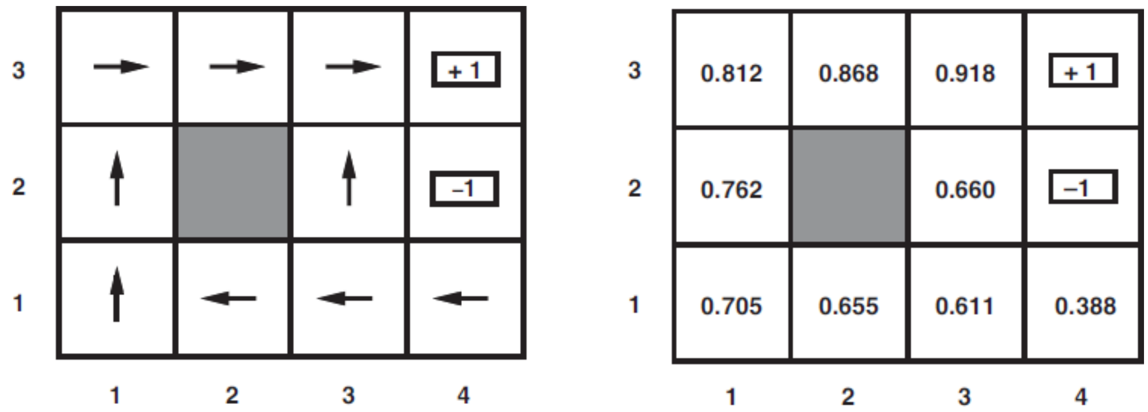
Example



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

- Note that to get from start to positive goal, we need at least 5 steps, no matter whether we go up or right in the first step

Example



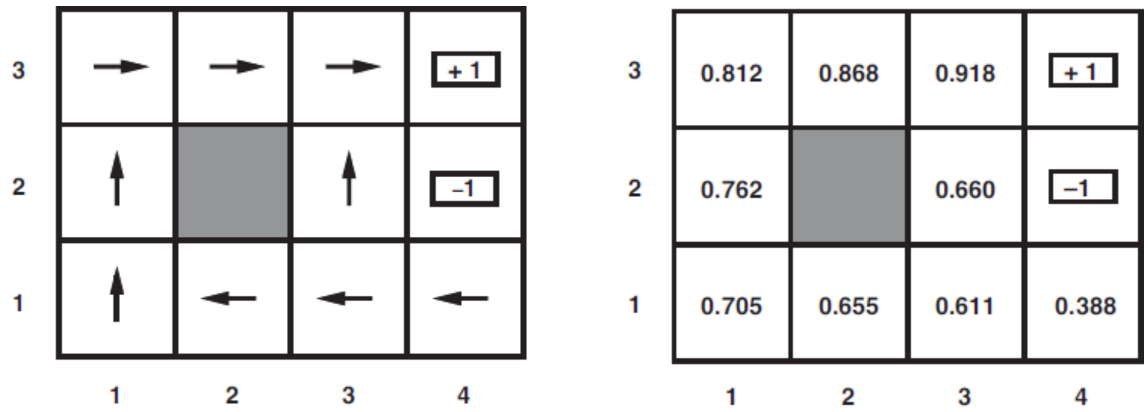
- **Why does (1,2) have a higher value than (2,1)?**

  When going right, there is a higher risk of ending up in negative goal due to uncertainty in moves

- **Why does optimal policy recommend going back to start from (2,1)?**

  Since the expected value from start is 0.094 higher than from (3,1) it is worth going back.

Example

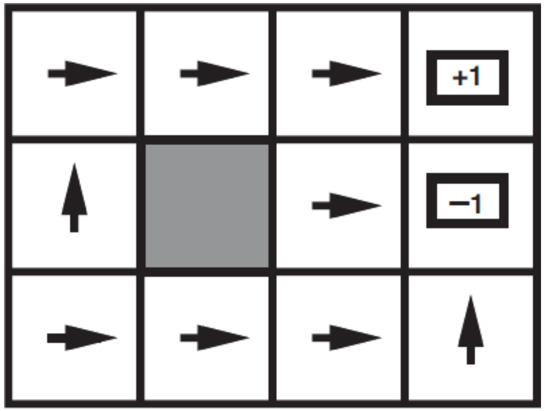


- **Why does optimal policy recommend going left from (3,1)?**

  Because one should consider also unintended effects of actions: choosing *up* might result in a move to the right, and (4,1) has a bad expected value.

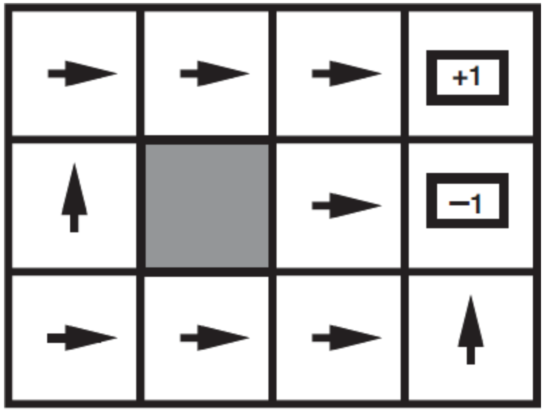
  We'll see this more formally in a moment.

Example



- The picture above shows optimal policy when steps from non-goal states are rewarded with -2 (penalty)

Example



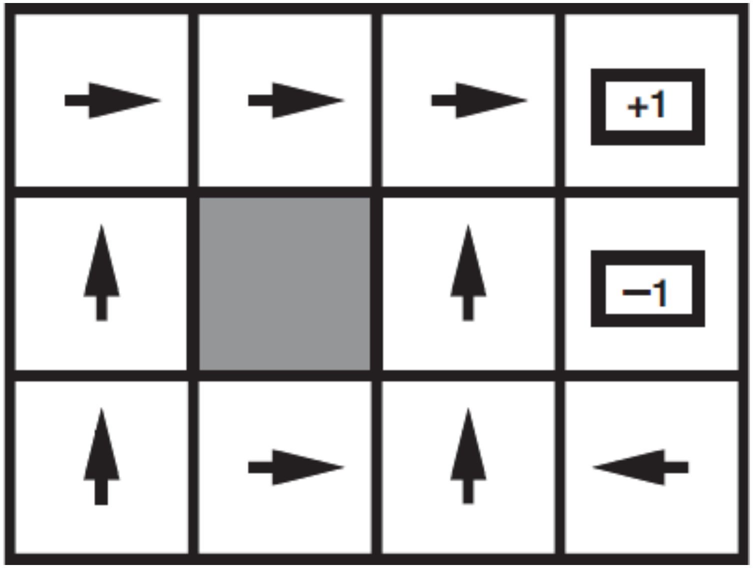
- **Why is it more reasonable to move right from the start state now?**

  Each step is significantly more expensive than the reward that we can get from any goal state.

  Therefore, the agent should try to end the game as soon as possible.

  The negative goal state can be reached before the positive one.

Other rewards



- Reward -0.4

Policy Iteration Algorithm

Reasoning about Policies

- Our main problem is finding an optimal policy (plan)
- We can consider subproblems of increasing difficulty
  - Given MDP and policy, evaluate policy
  - Given MDP, find optimal policy
  - Find optimal policy for unknown MDP (reinforcement learning)

Policy Iteration Overview

- Initialize policies π(s) to random actions
- Repeat
  - Step 1: Policy evaluation: calculate value Vπ(s) for each s
  - Step 2: Policy improvement: update policy using one-step look-ahead
- Until policy doesn't change

Policy Evaluation

Evaluating Policies Idea

- How can we evaluate deterministic policy π?
- Consider value function $v: S \rightarrow \mathbb{R}$ that maps states to values
- We let $v_\pi(s)$ be the expected discounted reward when starting in s and following policy π

$$v_\pi(s) = \mathrm{E}_\pi[r_1 + \gamma * r_2 + \gamma^2 * r_3 + \gamma^3 * r_4 + \dots \mid S_o = s]$$

Iterative Policy Evaluation Idea

- We represent v by an array

  [$v(s_1)$, $v(s_2)$, $v(s_3)$, …, $v(s_n)$]

- We initialize all values with 0

  [0, 0, 0, …, 0]

- We then update values by adding up the reward for the next action (given by policy) and the current expected value of the next state
- One can show that this approach converges to the true values of π
- However, for γ=1, values may be unbounded (termination problem)

Iterative Policy Evaluation Algorithm

- **Input**: - MDP (S, A, p, r)
  - deterministic Policy π
  - discount factor γ

  Output: - value function v_π

Initialize v(s) = 0 for all s in S

do

for each s in S

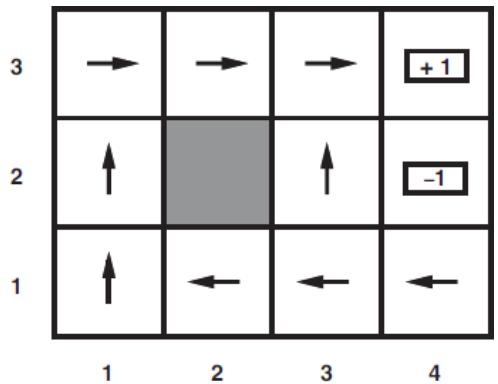
$$v(s) \leftarrow r(s, \pi(s)) + \gamma * \sum_{s'\, in\, S} p(s' \mid s, \pi(s)) * v(s')$$

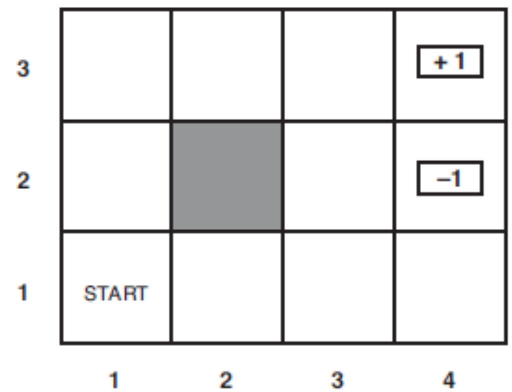
until change in v 'is negligible'

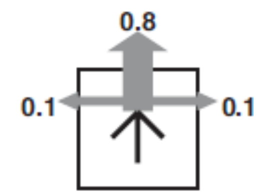
return v

Iterative Policy Evaluation Example

(γ=1)







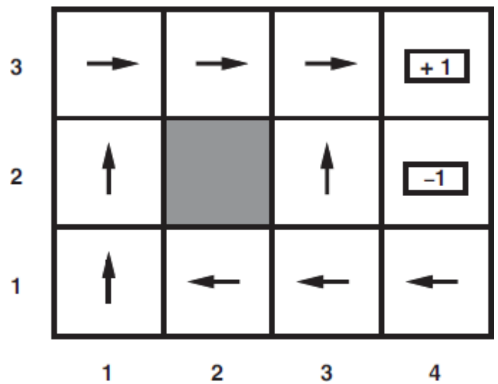
- 0.04

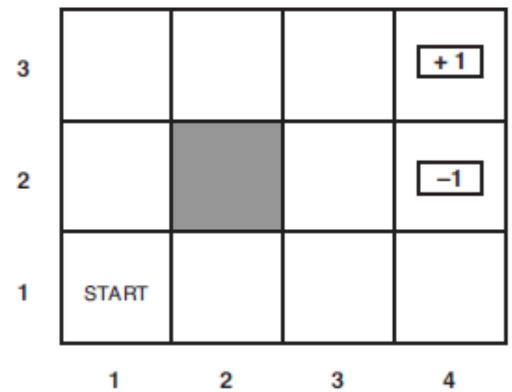
Initialization

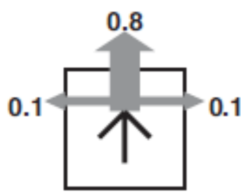
3	0	0	0	0
2	0		0	0
1	0	0	0	0

Iterative Policy Evaluation Example

(γ=1)







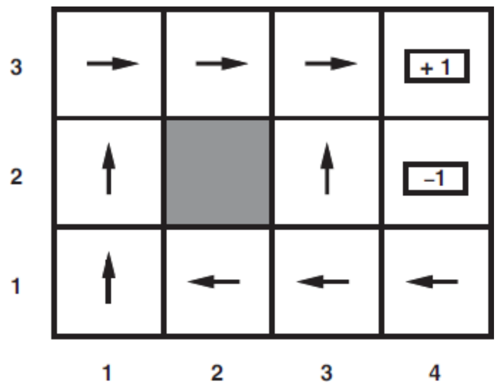
- 0.04

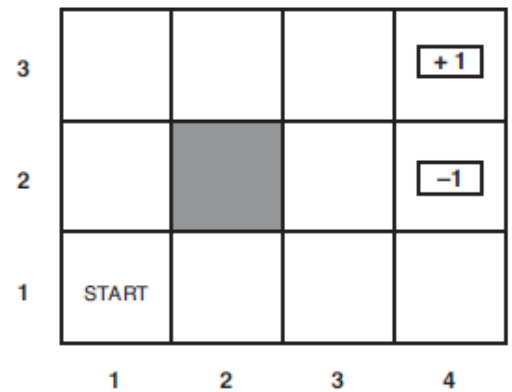
After first iteration

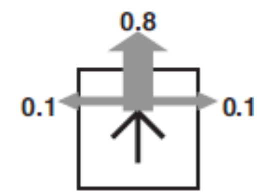
3	-0.04	-0.04	-0.04	1
2	-0.04		-0.04	-1
1	-0.04	-0.04	-0.04	-0.04

Iterative Policy Evaluation Example

(γ=1)







- 0.04

Second iteration

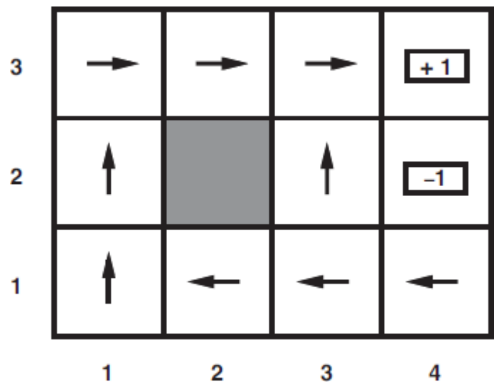
3	-0.04	-0.04	-0.04	1
2	-0.04		-0.04	-1
1		-0.04	-0.04	-0.04

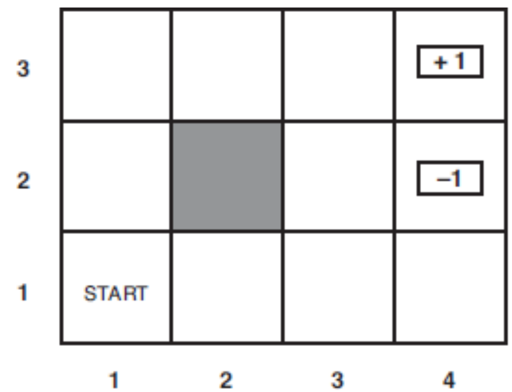
-0.04
+ 0.8 * -0.04
+ 0.1 * -0.04
+ 0.1 * -0.04

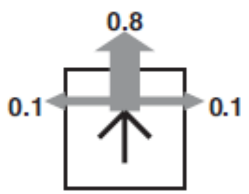
= -0.08

Iterative Policy Evaluation Example

(γ=1)







- 0.04

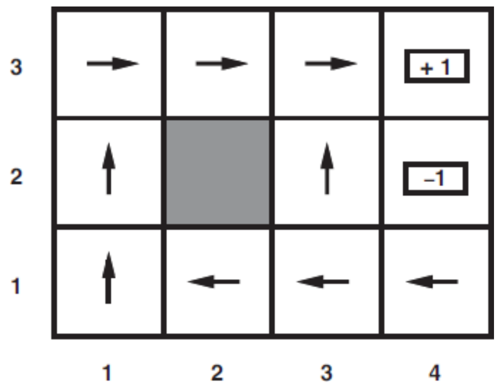
Second iteration

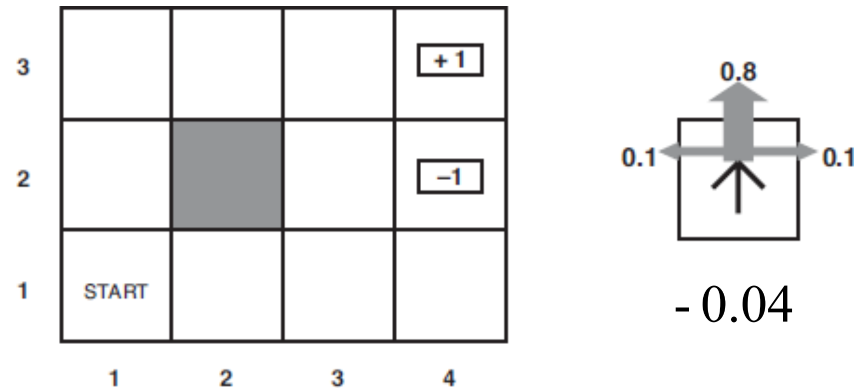
3	-0.04	-0.04		1
2	-0.04		-0.04	-1
1	-0.04	-0.04	-0.04	-0.04

-0.04
+ 0.8 * 1
+ 0.1 * -0.04
+ 0.1 * -0.04

= 0.752

Iterative Policy Evaluation Example (γ=1)





Second iteration

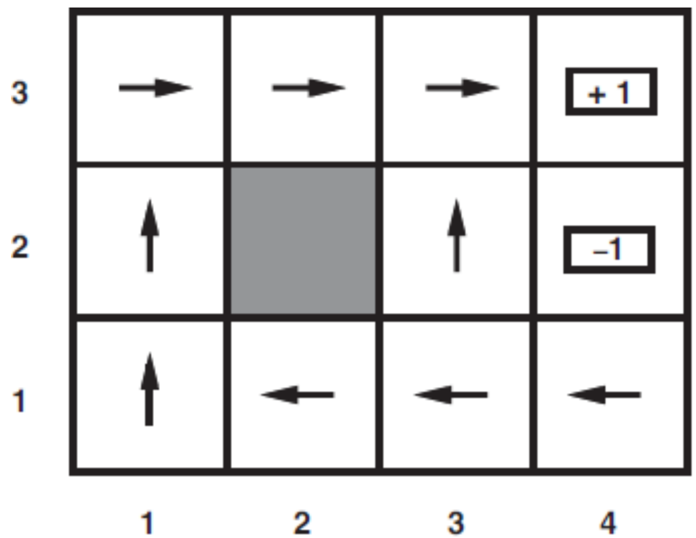
3	-0.04	-0.04	-0.04	1
2	-0.04			-1
1	-0.04	-0.04	-0.04	-0.04

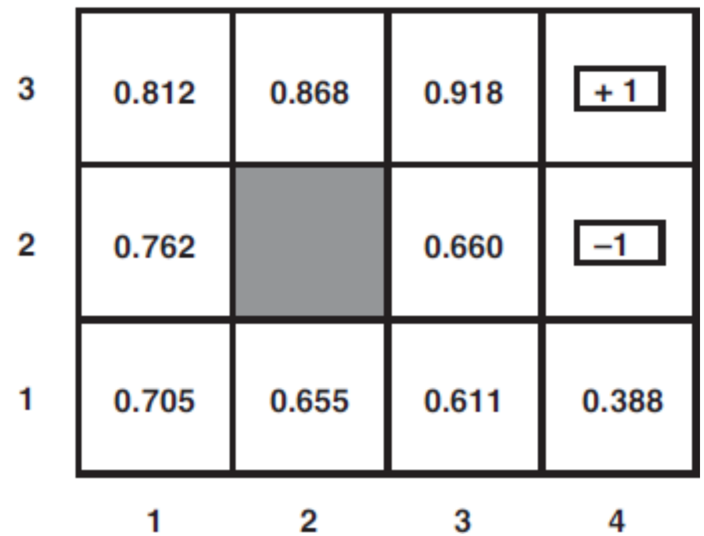
-0.04
+ 0.8 * -0.04
+ 0.1 * -0.04
+ 0.1 * -1

= -0.176

Iterative Policy Evaluation Example (γ=1)

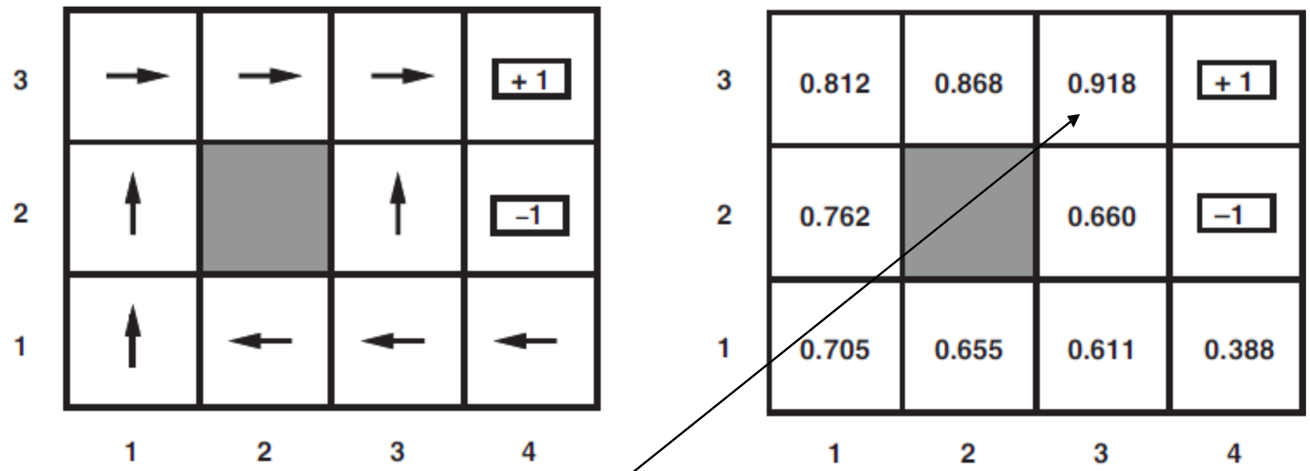
- Here are again the final values of the (optimal) policy





Iterative Policy Evaluation Example (γ=1)

- Value function is stable under updates



-0.04
+ 0.8 * 1
+ 0.1 * 0.918
+ 0.1 * 0.66
- - - - - - - - - - - - - - -
= 0.9178

Policy Improvement

Improving Policies Idea

- Now we know how to evaluate a given deterministic policy
- How can we find an optimal policy?
- Roughly speaking, we can do so by
    - Starting from an arbitrary policy
    - Evaluating the policy
    - Improving the policy

  And iterating until no improvement is possible anymore

- This approach is called Policy Iteration

How to improve Policies

- How can we improve our policy?

- One can show that an optimal policy π must be greedy

$$r(s, \pi(s)) + \gamma * \sum_{s' \, in \, S} p(s' | s, \pi(s)) * v(s')$$

$$= \max_{a \, in \, A} r(s, a) + \gamma * \sum_{s' \, in \, S} p(s' | s, a) * v(s')$$

for all states s

- If π(s) is not greedy, we can improve policy by replacing π(s) with a greedy action (Policy Improvement Theorem)

- If π is greedy, then it is optimal (Bellman Optimality Equation)

Policy Iteration Algorithm

- **Input**: - MDP (S, A, p, r)
  - - discount factor γ
- **Output**: - optimal policy π

Initialize value estimate v and policy π arbritrarily

do

  v ← evaluate π *(perform policy evaluation)*

  for each s in S

    π(s) ← select greedy action with respect to v

until policy is stable (does not change anymore)

return π

Policy Iteration Algorithm

Initialize value estimate v and policy π arbritrarily

do

  v ← evaluate π   *(perform policy evaluation)*

  for each s in S

    π(s) ← action a that maximizes

$$r(s,a) + \gamma * \sum_{s' \, in \, S} p(s' \mid s, a) * v(s')$$

until policy is stable (does not change anymore)

return π

Convergence and Pitfalls

- Policy iteration is guaranteed to converge to an optimal policy under mild assumptions

- However, there are again some subtleties
  - for γ=1, values may be unbounded (evaluation may not terminate)
  - algorithm may cycle between actions that yield equal values
  - (in particular, optimal policy may not be unique)

- In Modified Policy Iteration, we perform only a fixed number of evaluation steps before improving policy
  - may converge faster
  - can avoid divergence problems for γ=1

Summary

Summary

- Markov Decision Processes take account of
  - Uncertainty and
  - Long-term consequences
- An MDP (S, A, p, r) consists of a definition of states, actions, transition probabilities and rewards
- policies describe in what state we choose what action
- policies can be evaluated by their expected reward
- planning comes down to computing an optimal policy

References

The presented slides are mainly Dr. Marco Volpe's slides

Most topics can be found in:

Russell, S., Norvig, P. *Artificial Intelligence - A modern approach.*
Pearson Education: 2010.
(3rd Edition. Sections 17.1-17.3)

More detailed information can be found in:

LaVallle, S. M. *Planning algorithms.*
Cambridge university press: 2006

Thrun, S., Burgard, W., & Fox, D. *Probabilistic robotics.*
MIT press: 2005.