Methods of Artificial Intelligence

Planning

UNIVERSITÄT OSNABRÜCK

Probabilistic Planning:

Markov Decision Processes
- Basics-

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 (e.g. normally distributed)

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 loose a lot of money later
  - Extending maintenance intervals may decrease spendings now
    - but may result in business interruptions due to technical failures later

Markov Decision Processes (MDPs)

- Markov Decision Processes take account of both problems

- Intuitively, MDPs describe
  - probability of state changes that correspond to 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

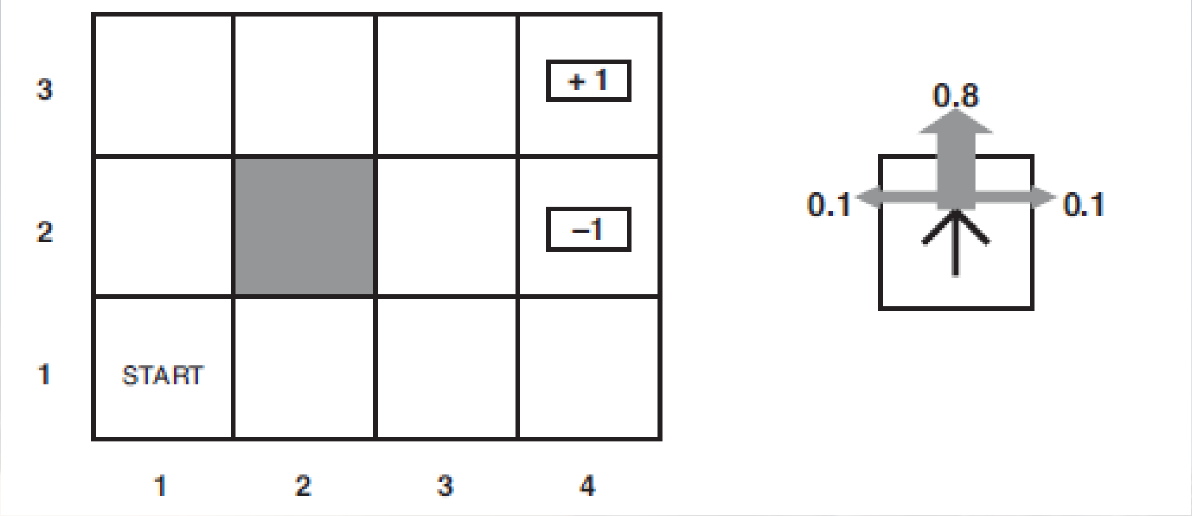
MDPs: Formal Definition

- 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

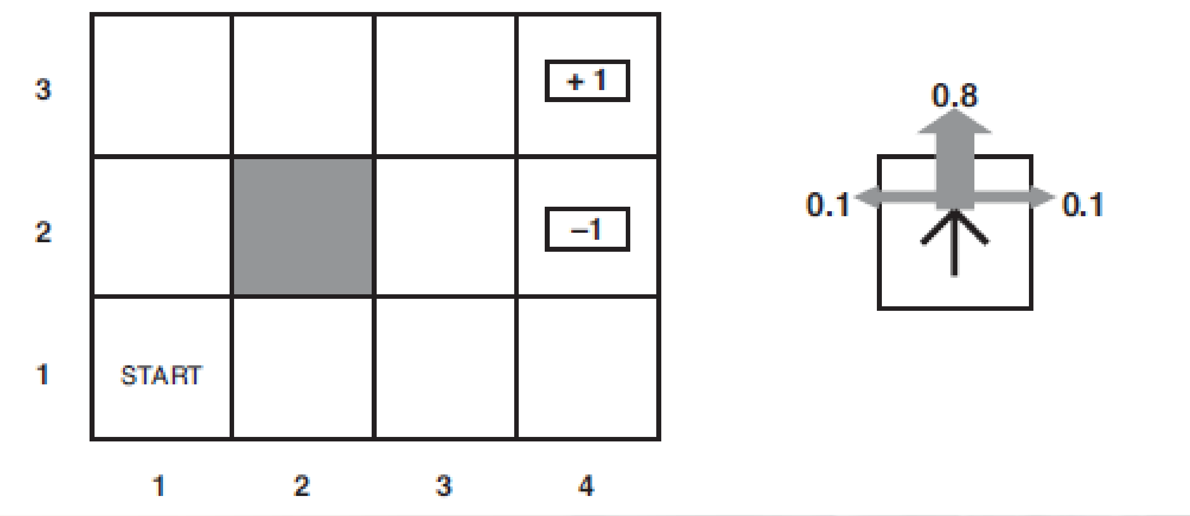
    r(s, a): short-term benefit of performing action a in state s

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 grids (special terminal states)

Example: States and Actions



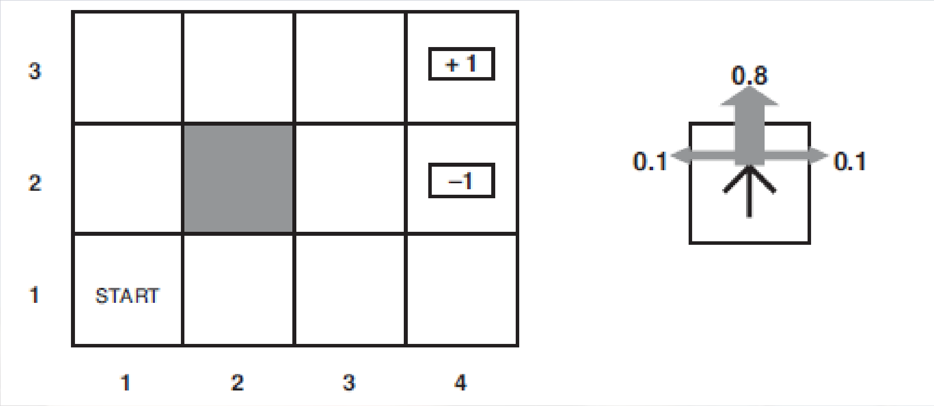
- **States** encode agent's current position

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

- **Actions** encode agent's possible actions

A = { up, down, left, right}

Example: Transition Probabilities

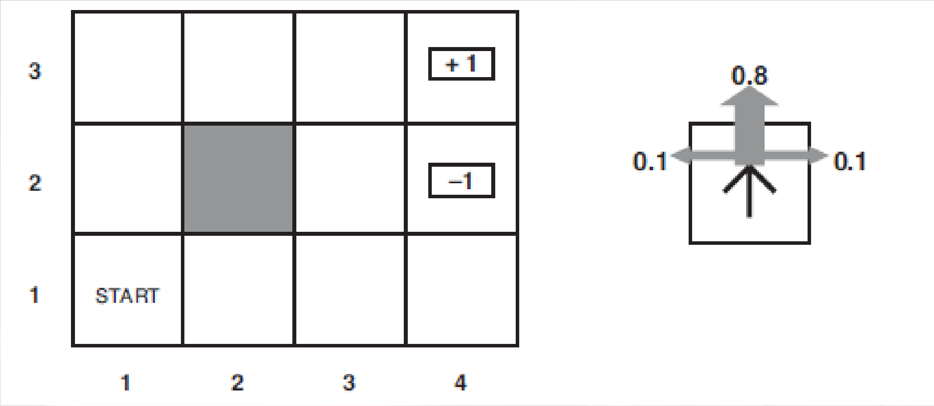


- Transition function p: when performing action a, with probability
  - 0.8, we will move in the intended direction
  - 0.1, we will move in direction 90° clockwise to intended direction
  - 0.1, we will move in direction 90° counterclockwise to intended direction

Unless action leads out of grid world or to an obstacle

– in this case, nothing happens

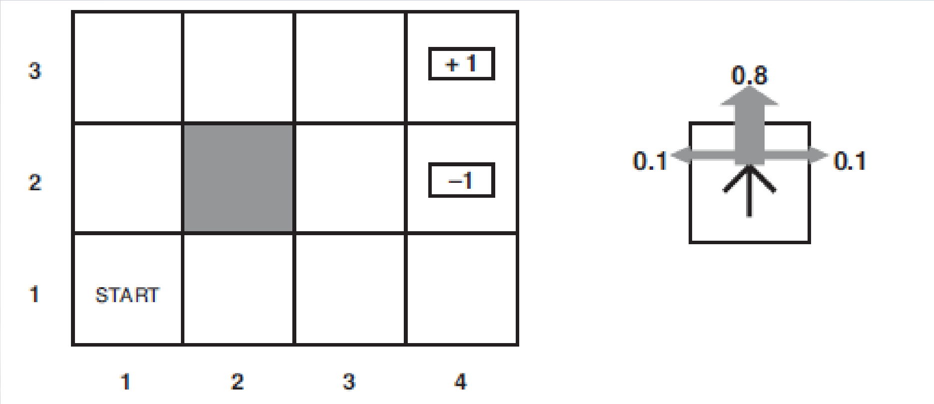
Example: Transition Probabilities



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

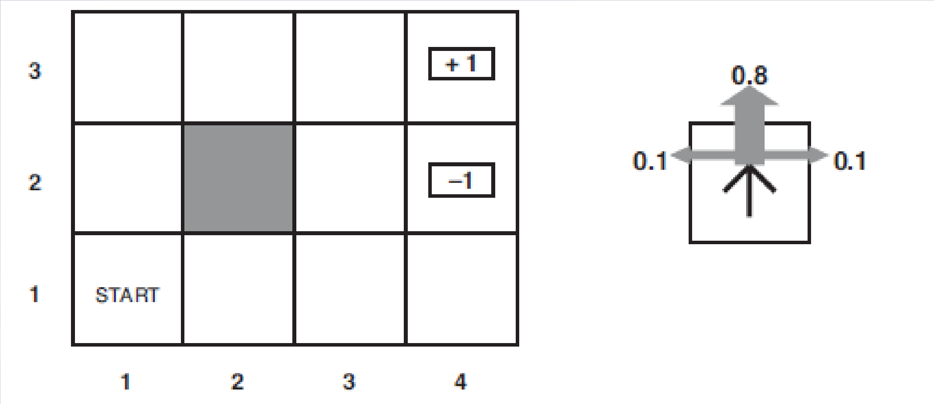
Unless action leads out of grid world – in this case, nothing happens

Example: Transition Probabilities



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

Example: Rewards

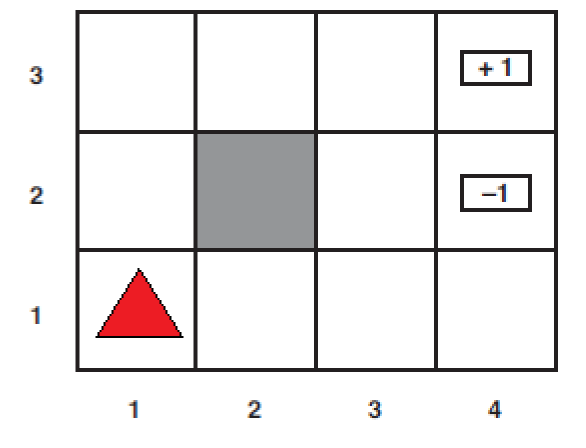


- Rewards give incentive to reach desired goal state
- r(s, a) = -0.04 for all actions a and all states s ≠ (2,4), s ≠ (3,4)

(each action that does not finish game, results in penalty)

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

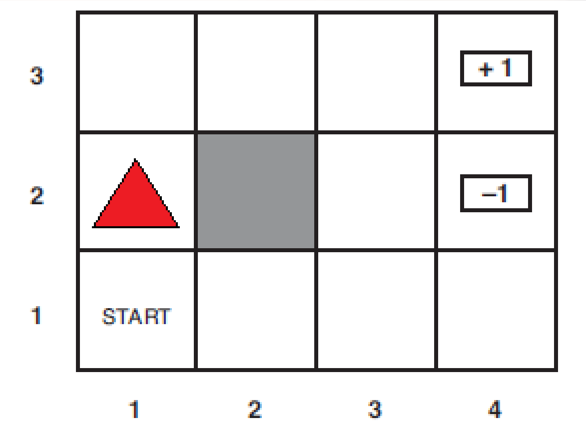
Example: Sample Run



Cumulative Reward: 0

Example: Sample Run

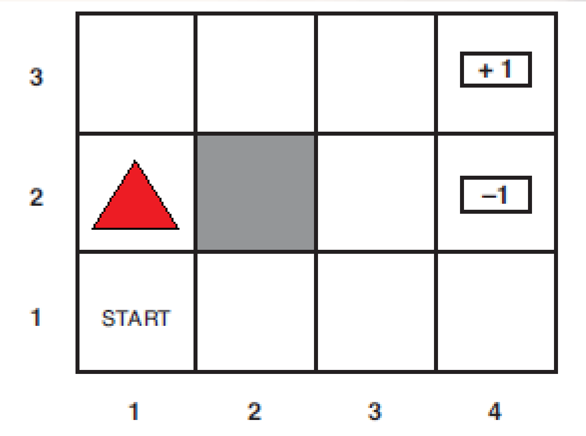
move up (works as intended)



Cumulative Reward: -0.04

Example: Sample Run

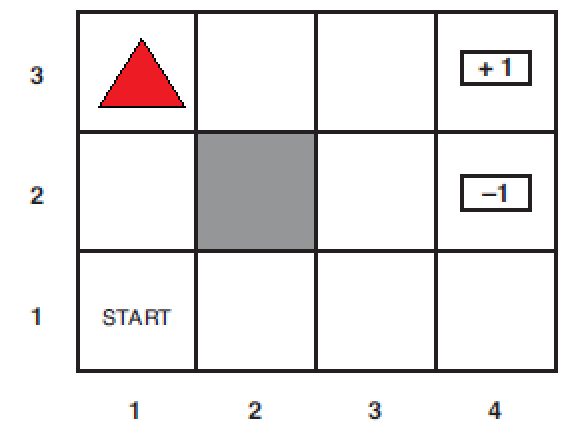
move up (move right instead)



Cumulative Reward: -0.08

Example: Sample Run

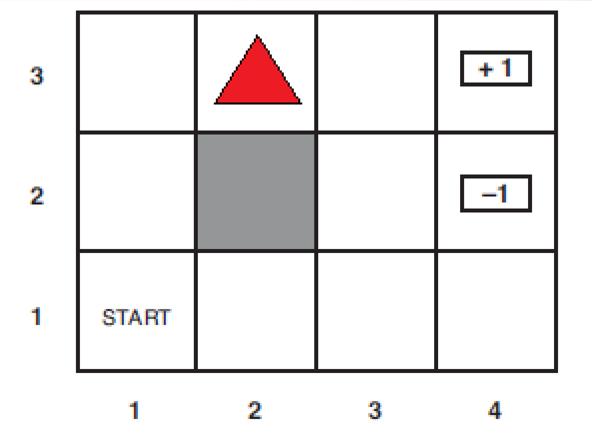
move up (works as intended)



Cumulative Reward: -0.12

Example: Sample Run

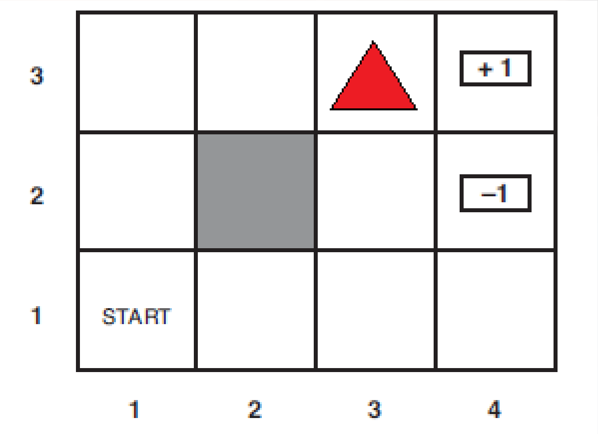
move right (works as intended)



Cumulative Reward: -0.16

Example: Sample Run

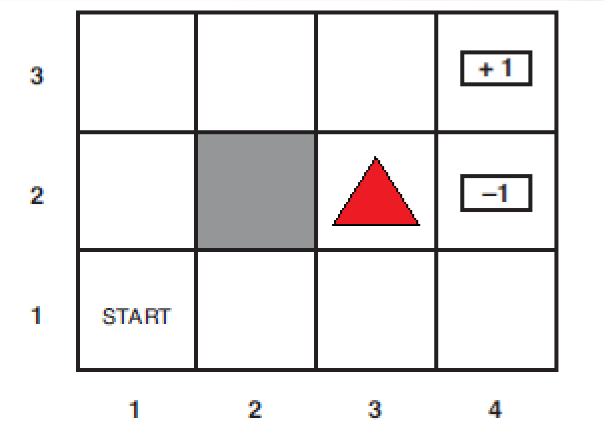
move right (works as intended)



Cumulative Reward: -0.2

Example: Sample Run

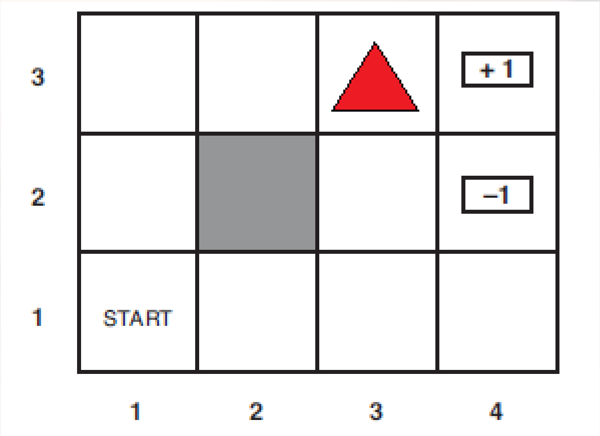
move right (move down instead)



Cumulative Reward: -0.24

Example: Sample Run

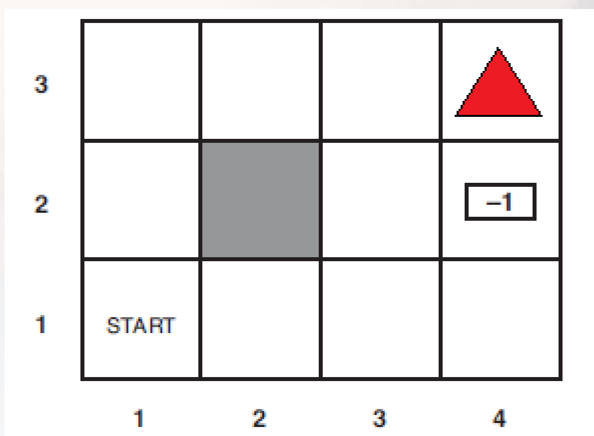
move up (works as intended)



Cumulative Reward: -0.28

Example: Sample Run

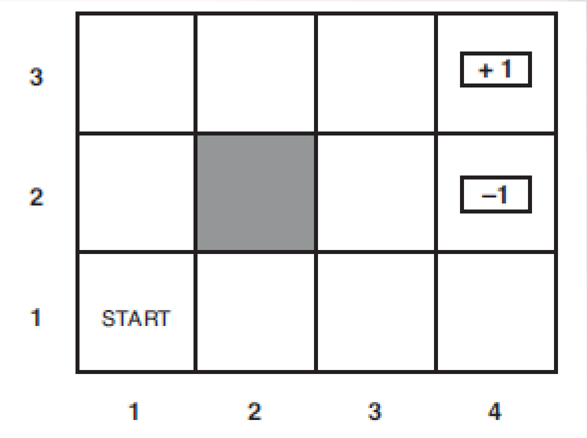
move right (works as intended)



Cumulative Reward: -0.32

Example: Sample Run

move right (effect does not matter – game ends)



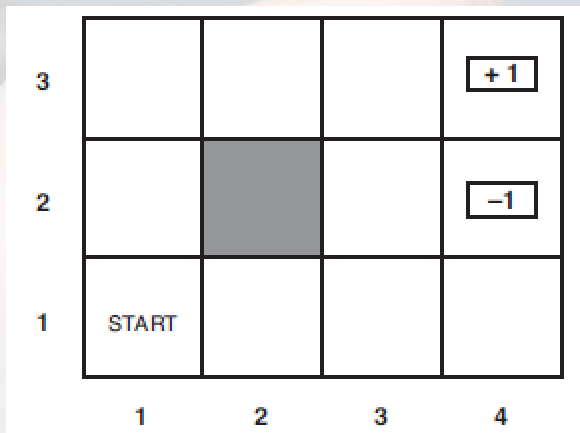
Cumulative Reward: **0.68**

Probabilistic Planning:

Markov Decision Processes
- Policies and Expected Rewards -

Policies

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



- π((1,1)) = up
- π((2,1)) = 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 in k-th step is

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

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

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

γ-discounted Rewards

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

  $E[r_1] + E[r_2] + E[r_3] + E[r_4] + \dots$ (possibly infinite series)

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

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

  $E[r_1] + \gamma * E[r_2] + \gamma^2 * E[r_3] + \gamma^3 * E[r_4] + \dots$

  where γ is a real number between 0 and 1

Exercise: Special Cases

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

  $r_1 + \gamma * r_2 + \gamma^2 * r_3 + \gamma^3 r_4 + \ldots$

  where γ is a real number between 0 and 1

- What happens if we let γ=0?
- What happens if we let γ=1?
- Write down the first four terms for γ=0.5

Exercise: Special Cases

- What happens if we let γ=0?

  0-discounted reward is r_1 (we ignore long-term rewards)

- What happens if we let γ=1?

  1-discounted reward equals expected reward

- Write down 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)

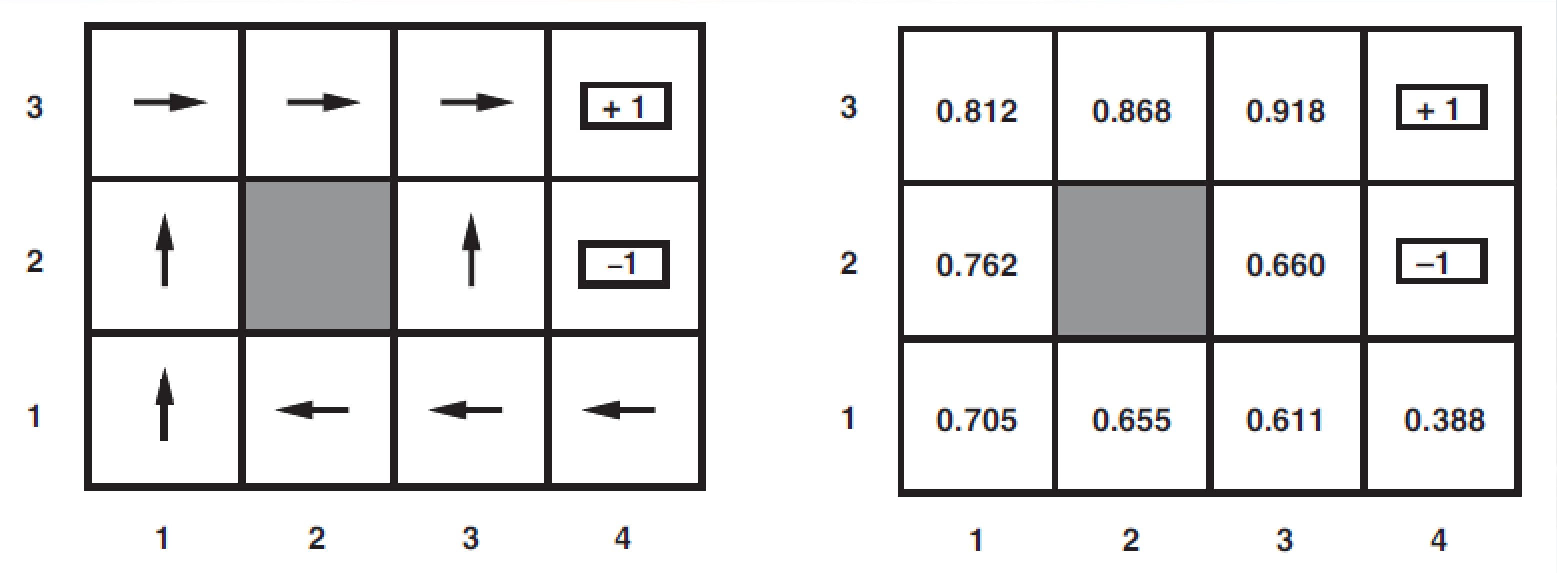
Computing γ-discounted Rewards

- We can compute an optimal policy with respect to the γ-discounted reward using different approaches
  - Policy Iteration
  - Value Iteration
  - Linear Programming

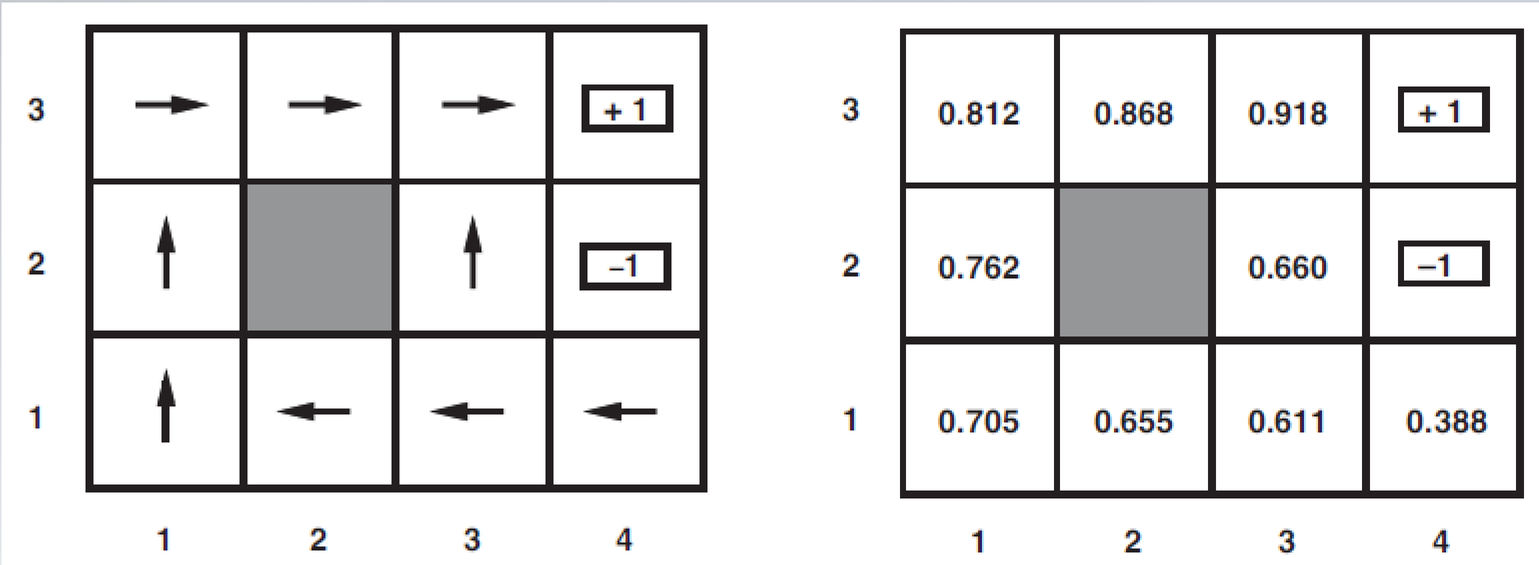
- Worst-case runtime of all approaches is polynomial in the number of states and actions

Example: Grid World

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

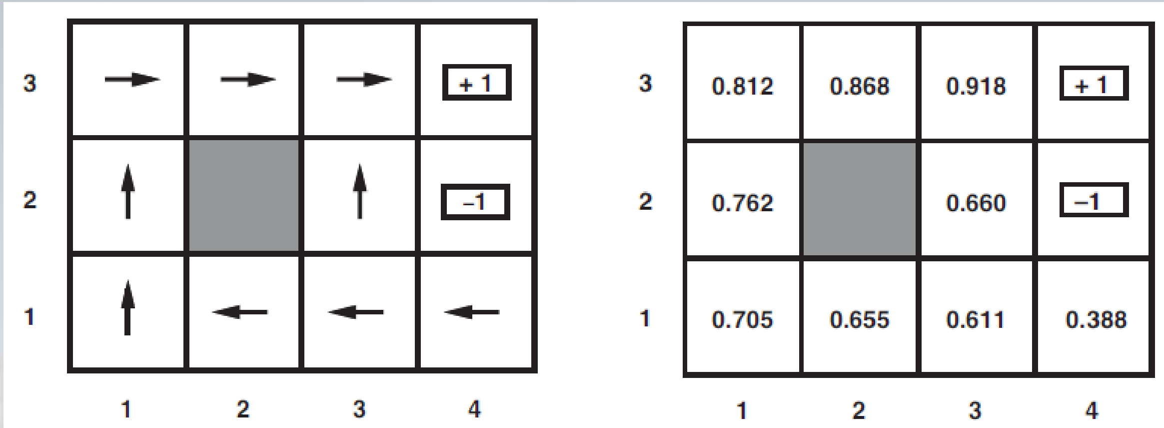


Exercise



- 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
- Why does going to (2,1) have a higher value than going to (1,2)?
- Why does optimal policy recommend going back to start from (1,2)?

Solution



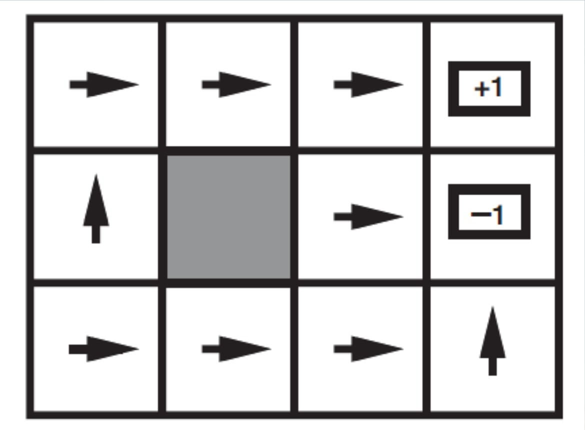
- Why does (2,1) have a higher value than (1,2)?

  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 (1,2)?

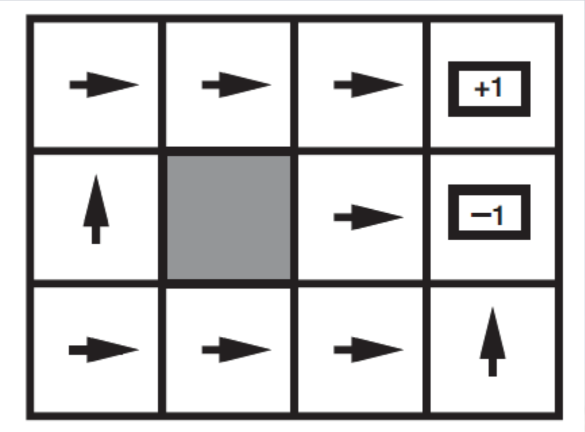
  Since the expected value from start is 0.094 higher than from (1,3) it is worth going back.

Exercise



- The picture above shows optimal policy when steps from non-goal states are rewarded with -2 (penalty)
- Why is it more reasonable to move right from the start state now?

Solution



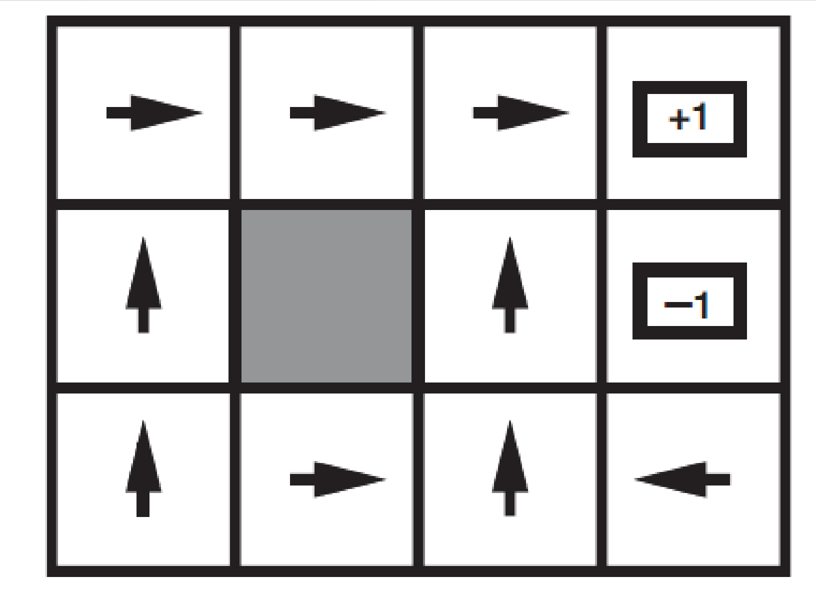
- Why is it more reasonable to move right from the start state now?

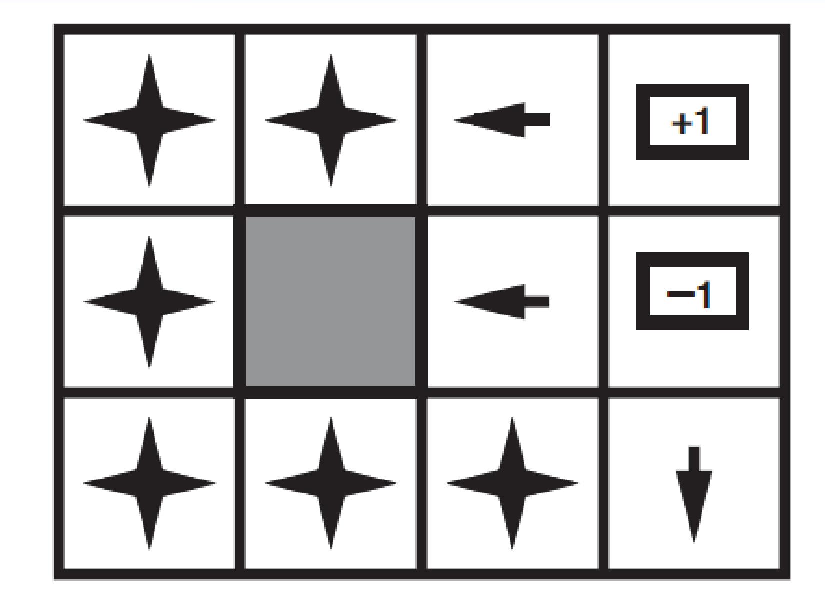
Each step is significantly more expensive than the reward that we can get from any goal state.

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
- Reward +1

Probabilistic Planning:

Markov Decision Processes
- Policy Evaluation and Improvement-

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)

Evaluating Policies Idea

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

$$v_\pi(s) = E[r_1 + \gamma * r_2 + \gamma^2 * r_3 + \gamma^3 * r_4 + \ldots \mid S_o = s]$$

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

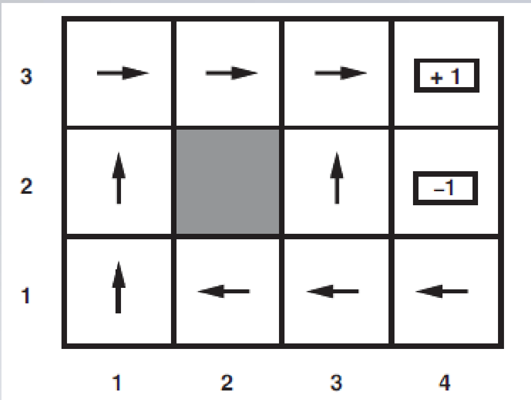
for each s in S

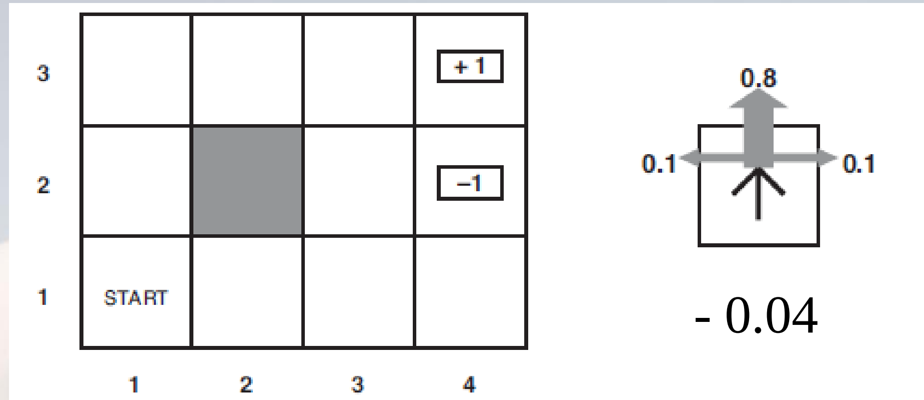
$$v(s) \leftarrow r(s, \pi(s)) + \gamma * \sum_{s' \, in \, S} p(s' | \, s, \pi(s)) * v(s')$$

until change in v 'is negligible'

return v

Iterative Policy Evaluation Example (γ=1)

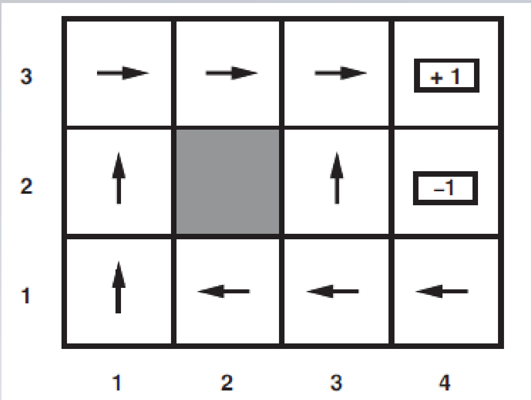


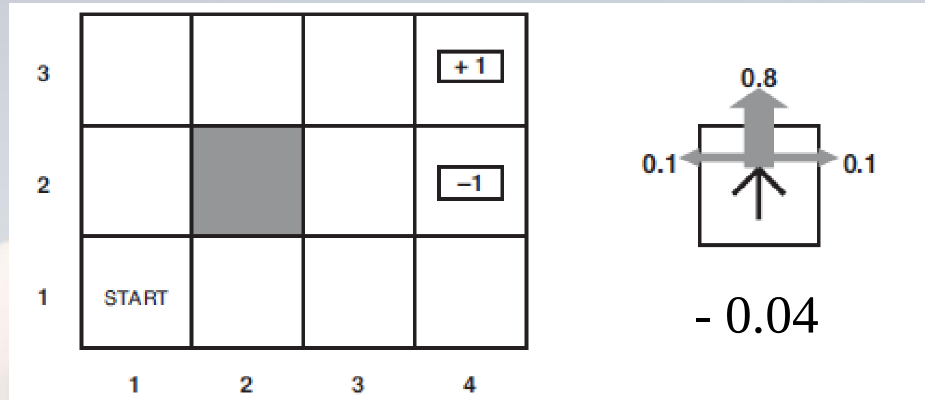


Initialization

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

Iterative Policy Evaluation Example (γ=1)

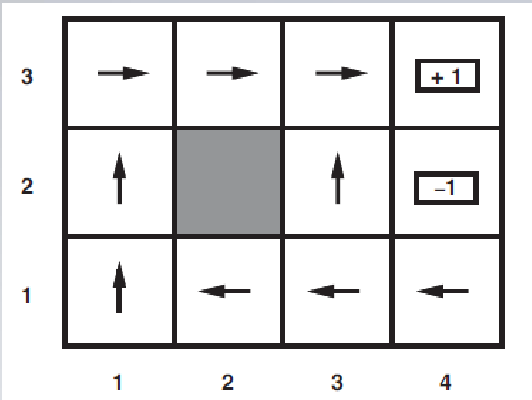


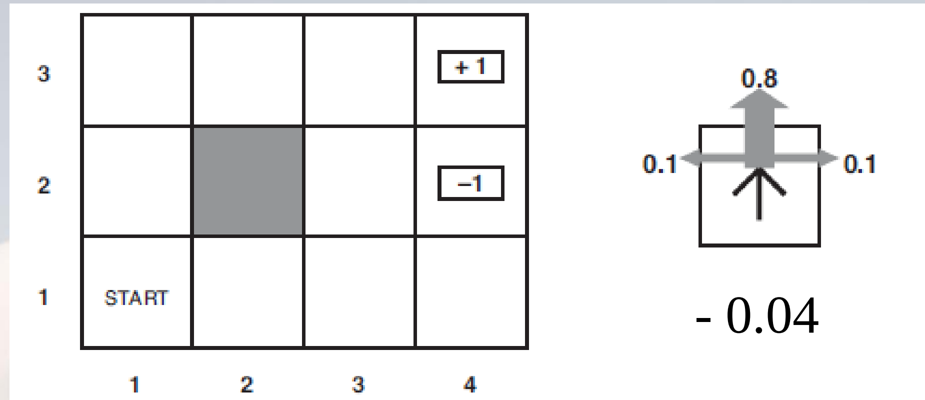


After Round 1

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)





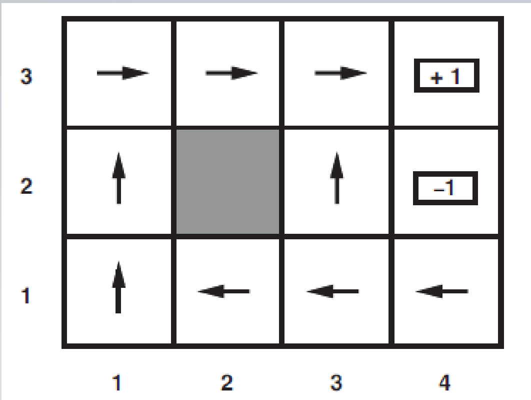
Round 2

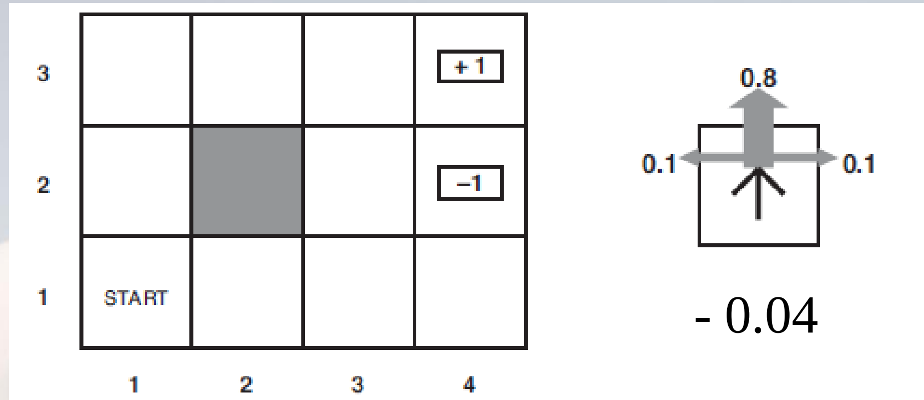
3	-0.04	-0.04	-0.04	1
2	-0.04		-0.04	-1
1		-0.04	-0.04	-0.04

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

= -0.08

Iterative Policy Evaluation Example (γ=1)





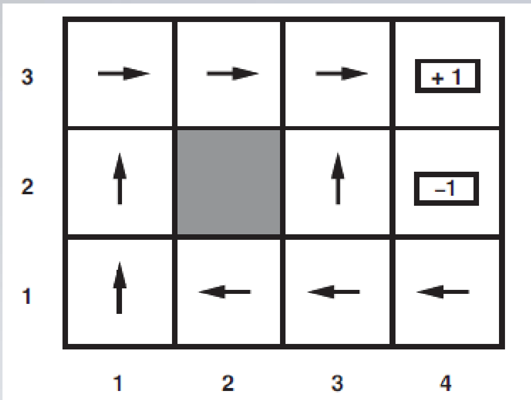
Round 2

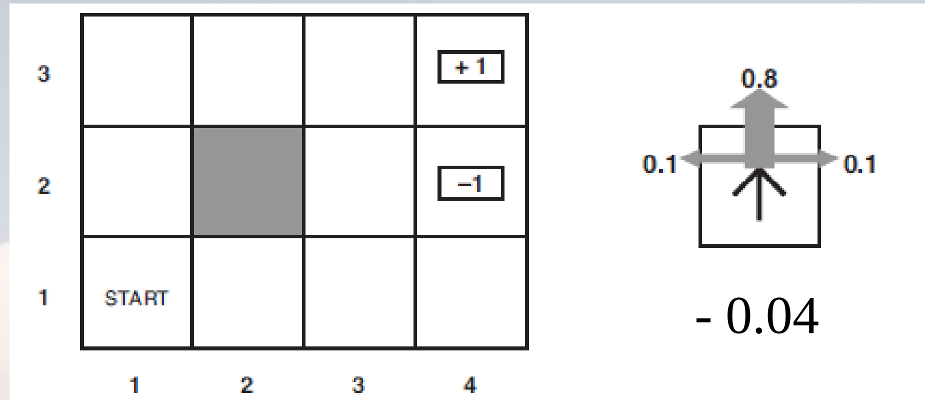
3	-0.04	-0.04		1
2	-0.04		-0.04	-1
1	-0.04	-0.04	-0.04	-0.04

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

= 0.752

Iterative Policy Evaluation Example (γ=1)





Round 2

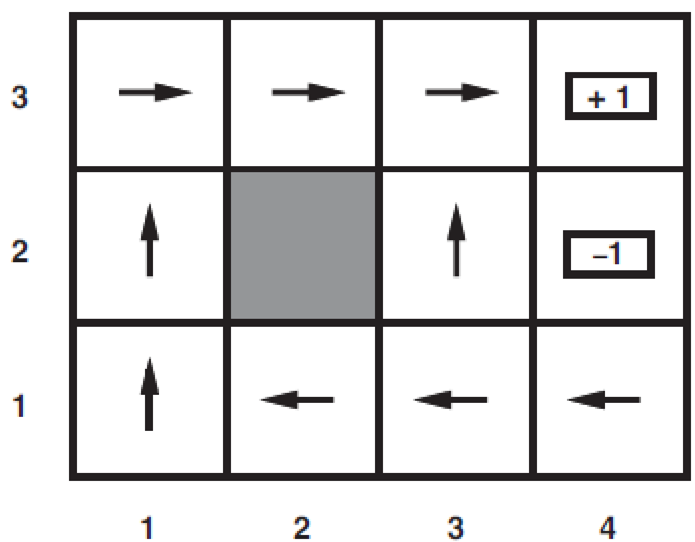
3	-0.04	-0.04	-0.04	1
2	-0.04			-1
1	-0.04	-0.04	-0.04	-0.04

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

= -0.176

Iterative Policy Evaluation Example (γ=1)

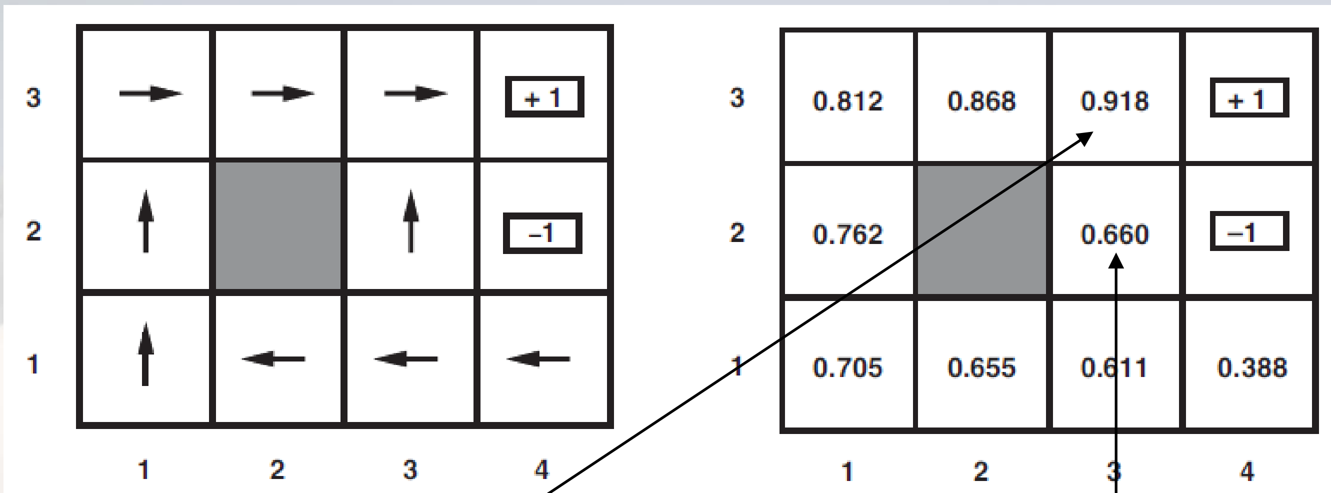
- Here are again the final values of the (optimal) policy



	1	2	3	4
3	0.812	0.868	0.918	+1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388

Policy Evaluation Example (γ=1)

- Value function is stable under updates (satisfies Bellman Equation)



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

= 0.9178

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

= 0.6604

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, 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) ← greedy action with respect to v

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 $\gamma=1$, 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 $\gamma=1$

Reinforcement Learning

- Finally, we want to find optimal policies for unknown MDPs
  - we may not know a reasonable transition and/ or reward model
  - or MDP may just be tedious to define
- We can do this by means of Reinforcement Learning
  - We basically learn MDP and optimal policy from observations
- We will discuss this in the Machine Learning sessions

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

Further Readings

Most topics can be found in:

Russell, S., Norvig, P. Artificial Intelligence - A modern approach. Pearson Education: 2010.

More detailed information can be found in:

LaValle, S. M. Planning algorithms.
Cambridge university press: 2006

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

Sutton, R. S., & Barto, A. G. Reinforcement learning: An introduction. Cambridge: MIT press: 1998.