Markov Decision Processes

Luís Macedo

Combining ideas for Stochastic planning

- What is a key limitation of decision networks?

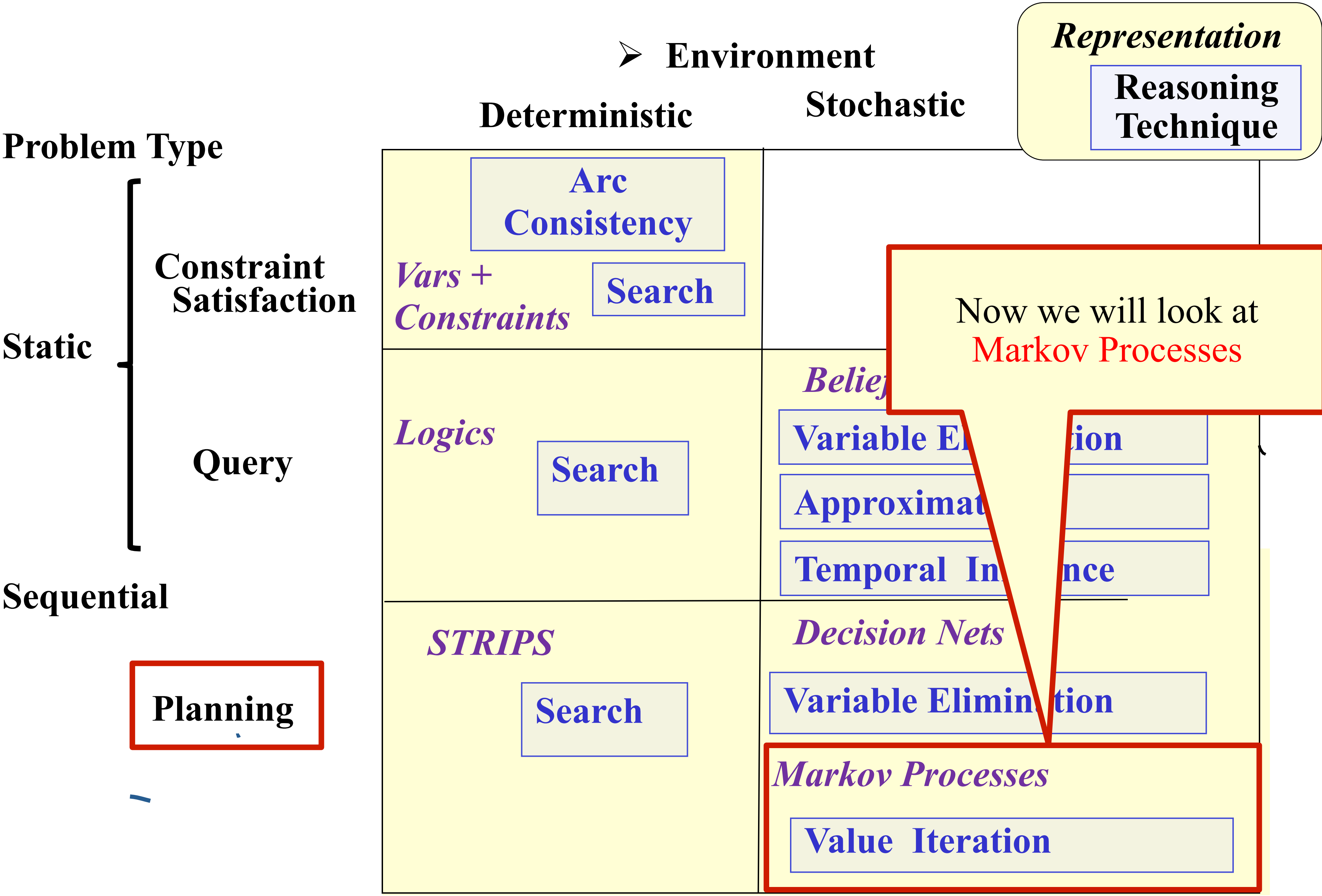
Represent (and optimize) only a fixed number of decisions

- What is an advantage of Markov models?

The network can extend indefinitely

Goal: represent (and optimize) an indefinite sequence of decisions

Representational Dimensions



Now we will look at Markov Processes

Markov Models

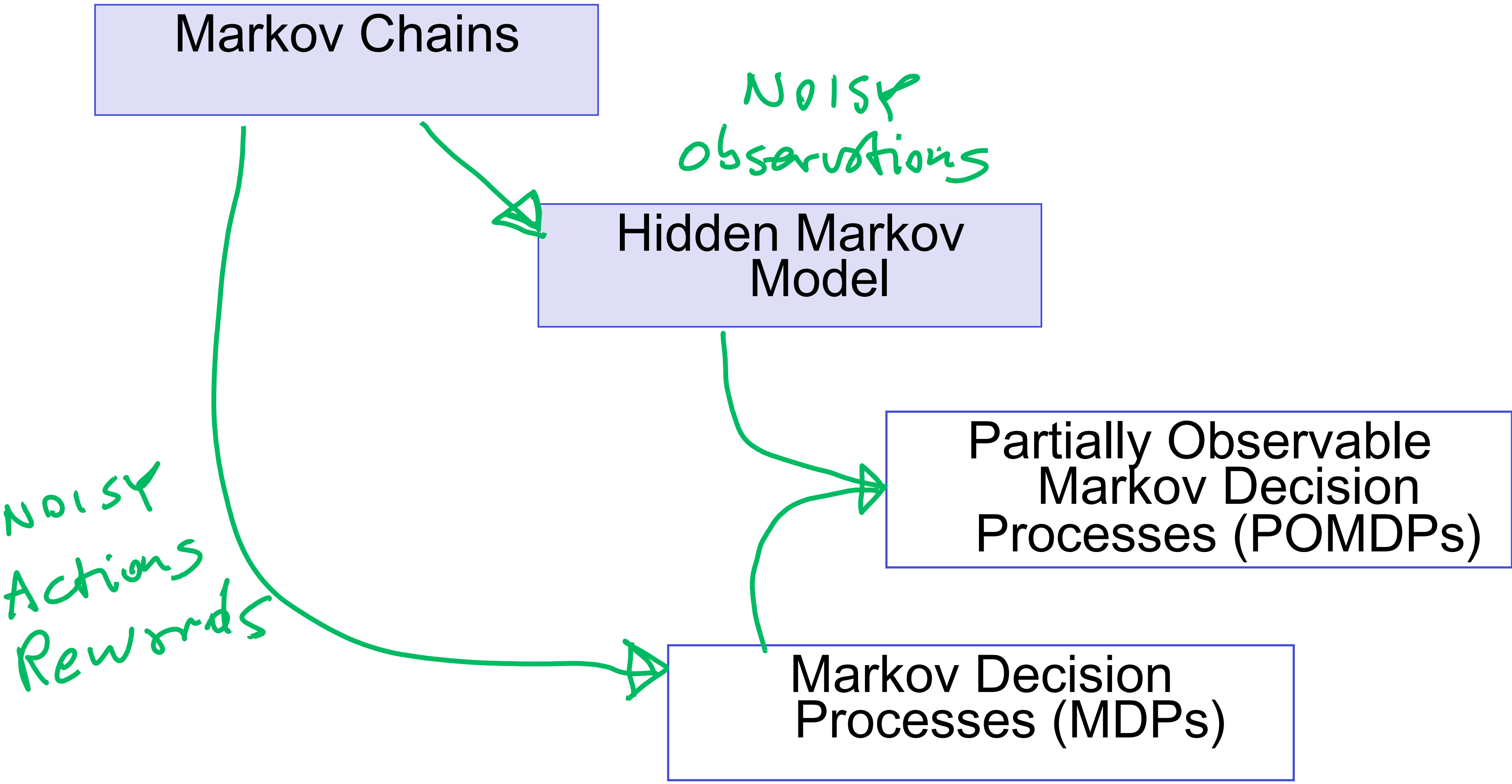
Markov Chains

Hidden Markov Model

Partially Observable Markov Decision Processes (POMDPs)

Markov Decision Processes (MDPs)

Markov Models



Markov Chains

Hidden Markov Model

Partially Observable Markov Decision Processes (POMDPs)

Markov Decision Processes (MDPs)

Overview

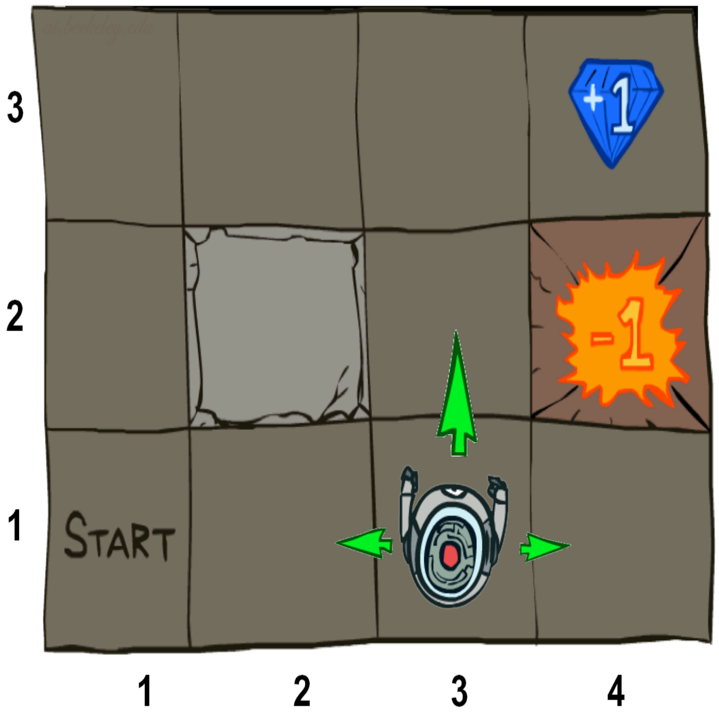
- Decision processes and Markov Decision Processes (MDP)
- Rewards and Optimal Policies
- Defining features of Markov Decision Process
- Solving MDPs

Decision Processes

- Often an agent needs to go beyond a fixed set of decisions
  - Would like to have an ongoing decision process
- **Infinite horizon problems**: process does not stop
- **Indefinite horizon problem:** the agent does not know when the process may stop
- **Finite horizon**: the process must end at a give time N
- At any stage of the process
  - The agent decides which action to perform
  - The new state of the world depends on the previous state as well as the action performed
  - The agent receives rewards or punishments at various points in the process
- Aim: maximize the reward received

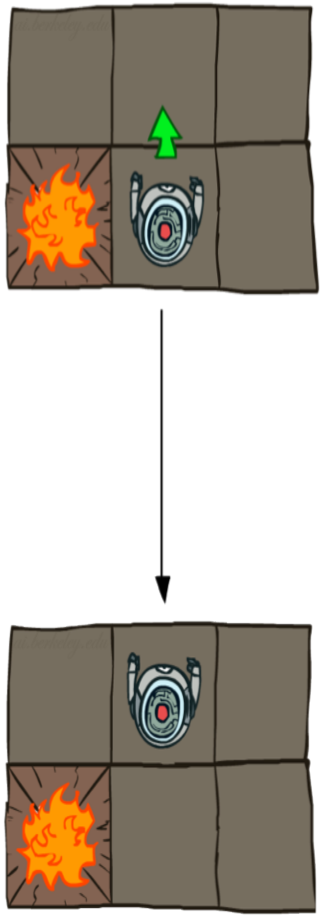
Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: 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
- The agent receives rewards each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards

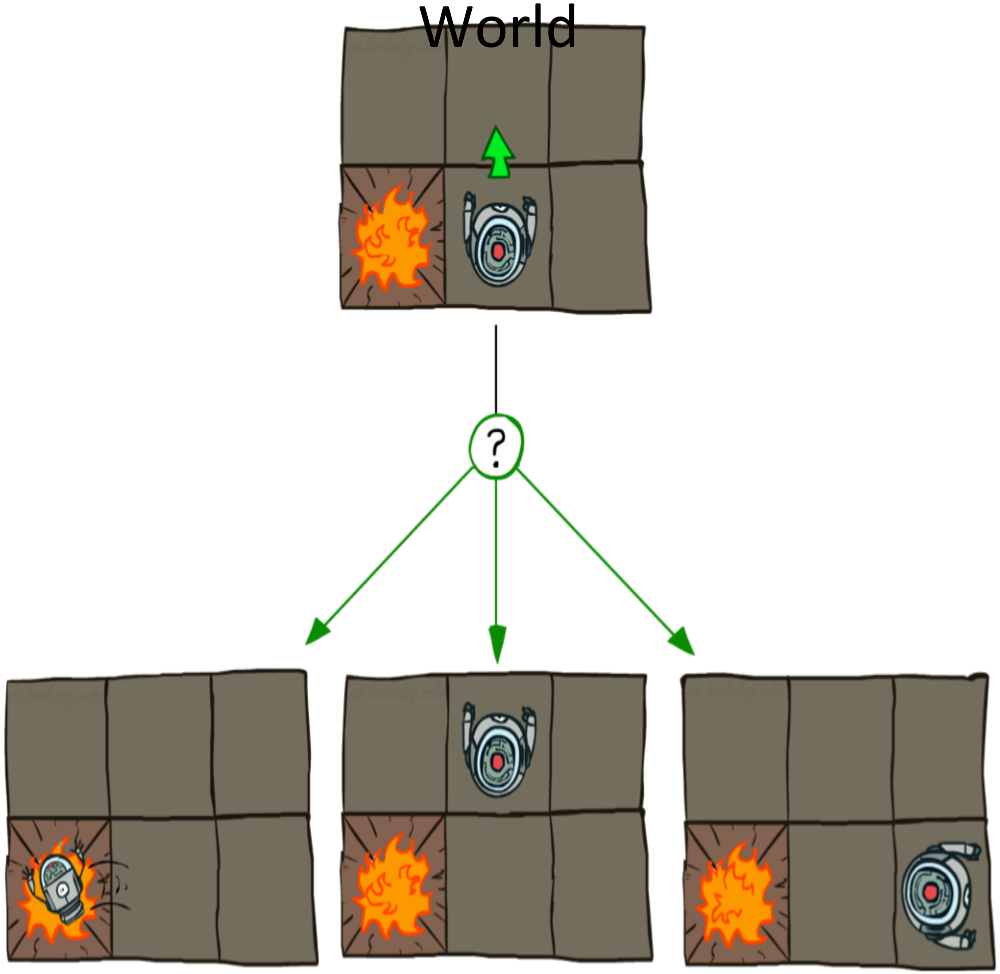


Grid World Actions

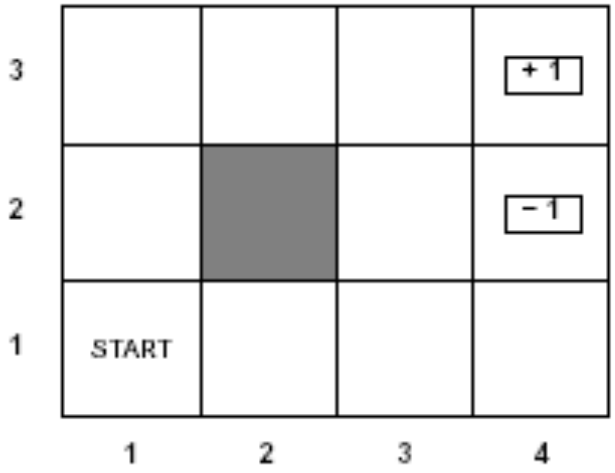
Deterministic Grid World

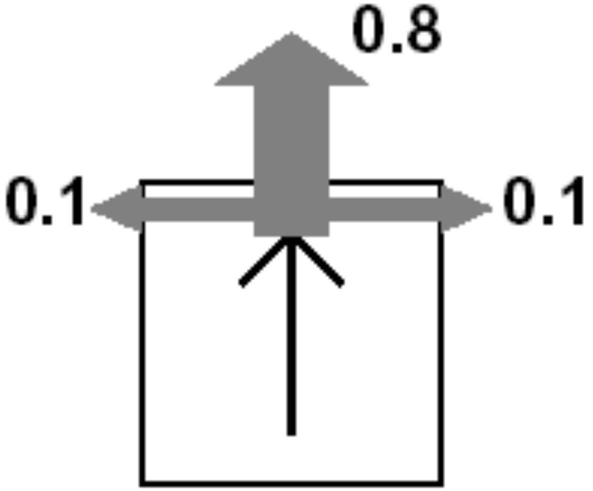


Stochastic Grid World



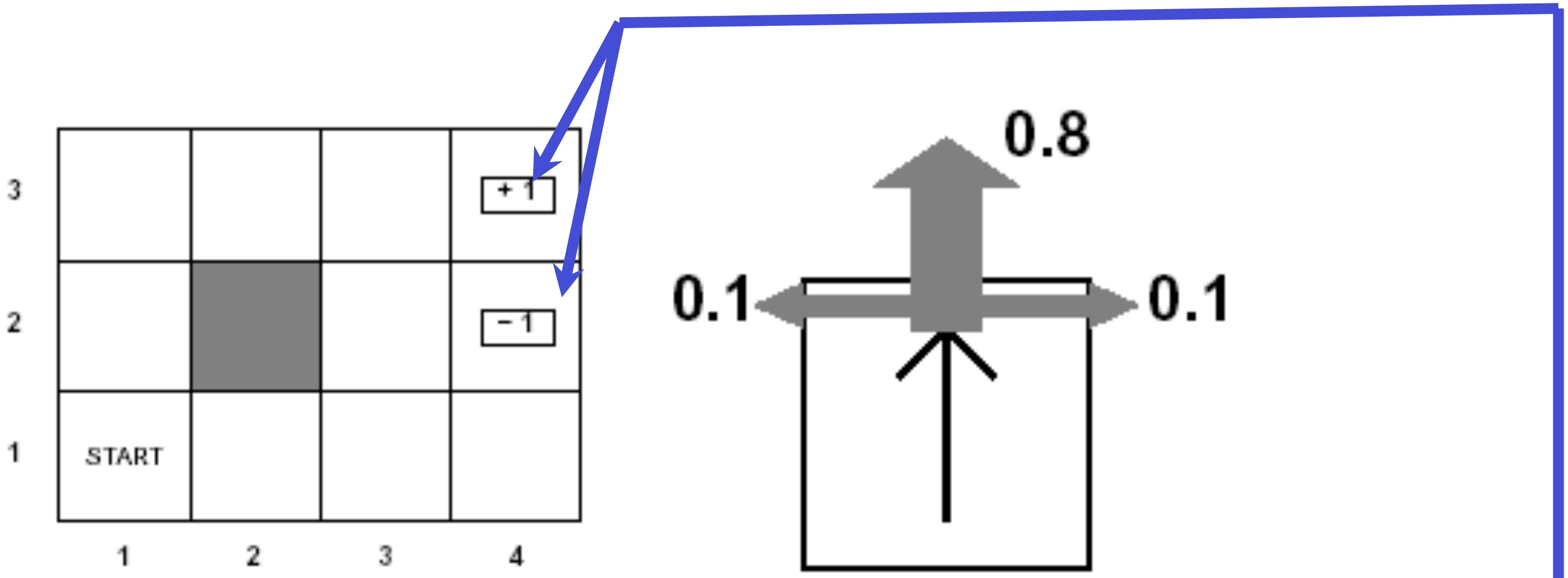
Example





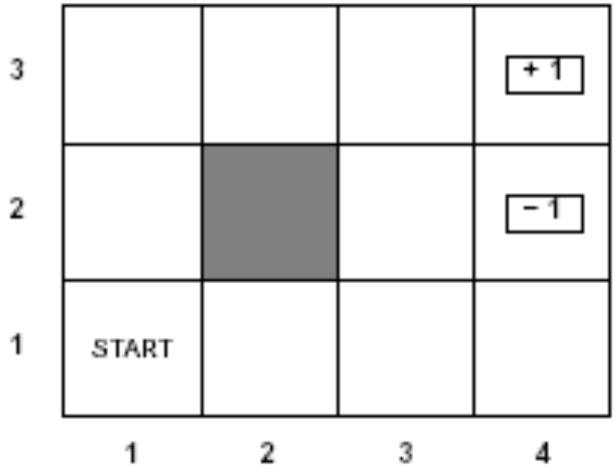
- Agent moves in the above grid via actions *Up, Down, Left, Right*
- Each action has
  - 0.8 probability to reach its intended effect
  - 0.1 probability to move at right angles of the intended direction
  - If the agents bumps into a wall, it stays there

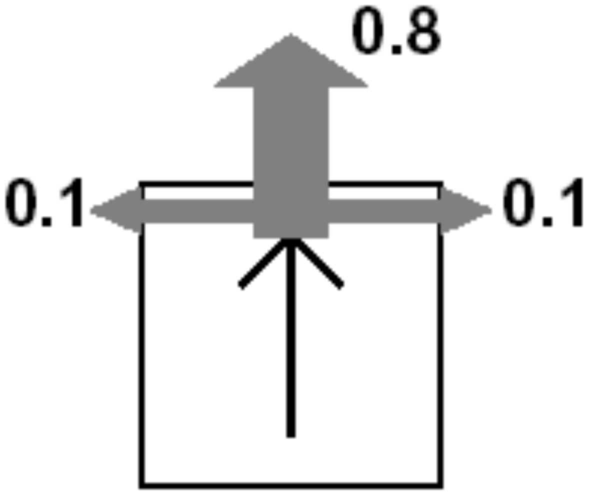
Example



$$R(s) = \begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

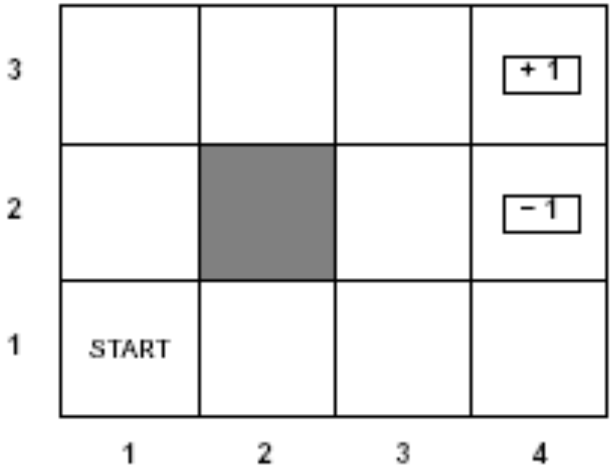
Example

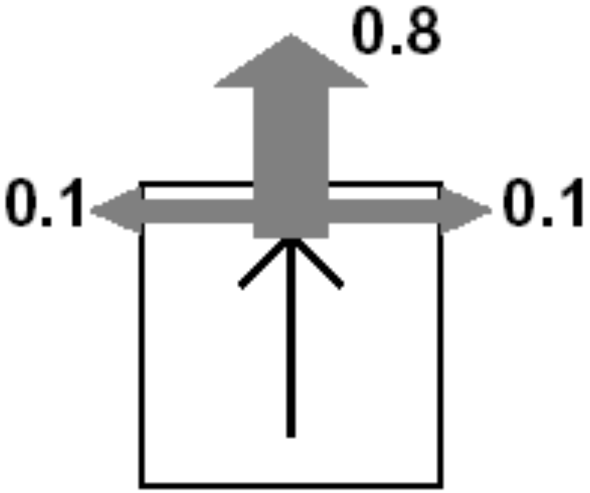




- Can the sequence [*Up, Up, Right, Right, Right*] take the agent in terminal state (4,3)?
- Can the sequence reach the goal in any other way?

Example





- Can the sequence [*Up, Up, Right, Right, Right*] take the agent in terminal state (4,3)?
  - Yes, with probability 0.8^5=0.3278
- Can the sequence reach the goal in any other way?
  - yes, going the other way around with prob. 0.1^4x0.8

Markov Decision Processes (MDP)

- We will focus on decision processes that can be represented as *Markovian* (as in Markov models)
  - Actions have probabilistic outcomes that depend only on the current state
  - Let S_t be the state at time t

$$P(S_{t+1}|S_0, A_0,\ldots,S_t, A_t) = P(S_{t+1}|S_t, A_t)$$

- The process is *stationary* if $P(s'|s , a)$ is the same at each point in time, where $P(s'|s , a)$ is the probability that the agent will be in state s' after doing action a in state s

- We also need to specify the utility function for the agent
  - depends on the sequence of states involved in a decision (***environment history***), rather than a single final state
  - Defined based on a reward $R(s)$ that the agent receives in each state s
    - ✓ can be negative (punishment)

MDP specification

➢ For an MDP you specify:

- set S of states
- set A of actions
- the process' dynamics (or ***transition model***)

  $P(S_{t+1}|S_t, A_t)$

- The reward function

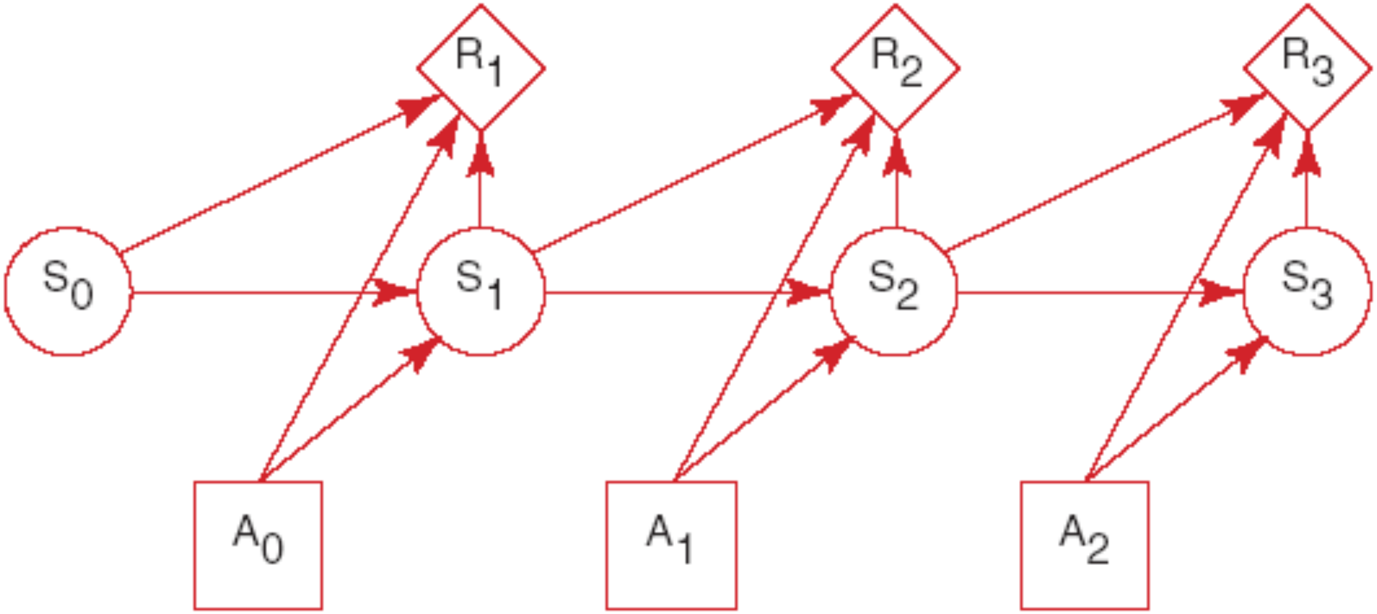
  $R(s, a,_{,}\ s')$

  describing the reward that the agent receives when it performs action *a* in state *s* and ends up in state *s'*

- We will use R(s) when the reward depends only on the state s and not on how the agent got there

MDP specification

- Basically a MDP augments a Markov chain with actions and values

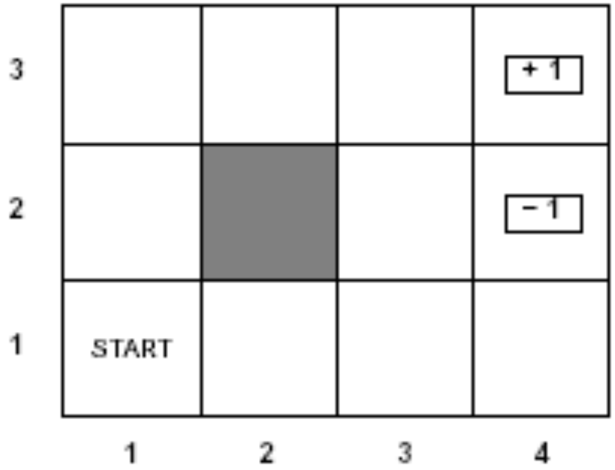


Solving MDPs

- The robot needs to know what to do as the decision process unfolds…
- It starts in a state, selects an action, ends up in another state selects another action….

- Needs to make the same decision over and over: given the current state what should I do?

  - Thus, a policy for an MDP is a single decision function $\pi(s)$ that specifies what the agent should do for each state s



Overview

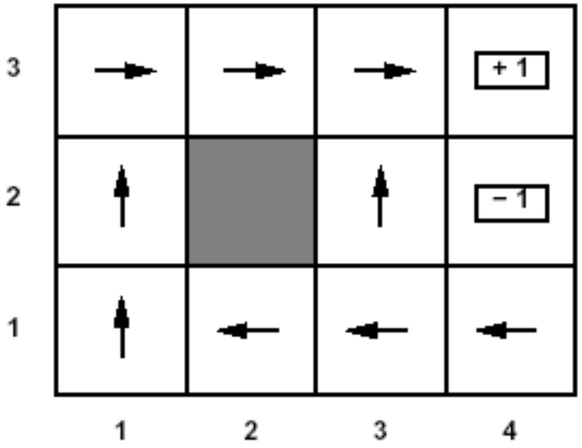
- Decision processes and Markov Decision Processes (MDP)
- Rewards and Optimal Policies
- Defining features of Markov Decision Process
- Solving MDPs
    - Value Iteration
    - Policy Iteration

Solving MDPs

- In MDPs, aim is to find an optimal *policy* π(s)
- Because of the stochastic nature of the environment, a policy can generate a set of environment histories with different probabilities
- Optimal policy maximizes expected total reward, where
  - Each environment history associated with that policy has a given amount of total reward
  - Total reward is a function of the rewards of its individual states (we'll see how)

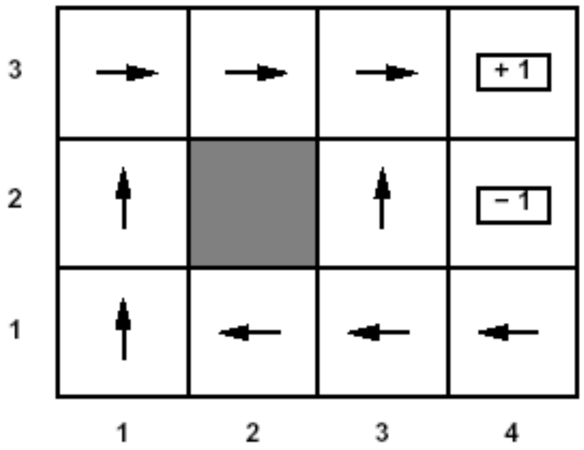
Optimal Policy in our Example

- Let's suppose that, in our example, the total reward of an environment history is simply the sum of the individual rewards
  - For instance, with a penalty of -0.04 in not terminal states, reaching (3,4) in 10 steps gives a total reward of 0.6
  - Penalty designed to make the agent go for short solution paths
- Below is the optimal policy when penalty in non-terminal states is -- 0.04 (we will see later how it is computed)



Rewards and Optimal Policy

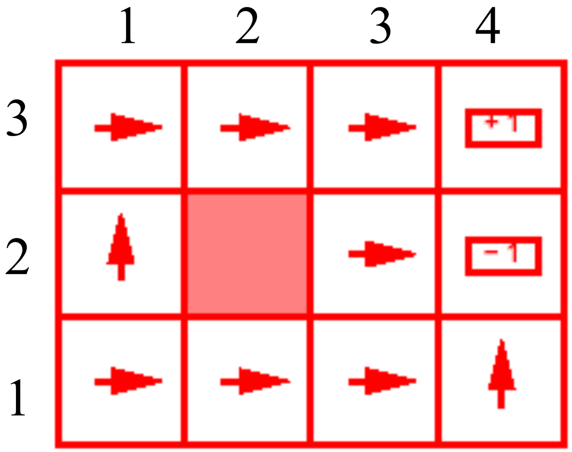
- ➢ Optimal Policy when penalty in non-terminal states is -0.04



- ➢ Note that here the cost of taking steps is small compared to the cost of ending into (2,4)
  - Thus, the optimal policy for state (1,3) is to take the long way around the obstacle rather then risking to fall into (2,4) by taking the shorter way that passes next to it
  - But the optimal policy may change if the reward in the non-terminal states (let’s call it *r*) changes

Rewards and Optimal Policy

- Optimal Policy when r < -1.6284

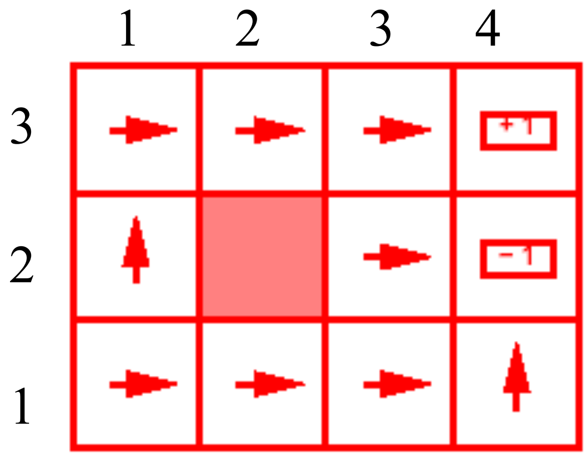


$r = [-\infty : -1.6284]$

- Why is the agent heading straight into (2,4) from it surrounding states?

Rewards and Optimal Policy

- Optimal Policy when r < -1.6284

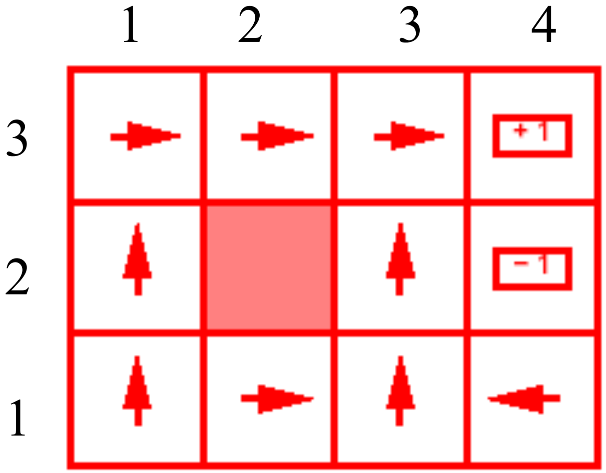


$r = [-\infty : -1.6284]$

- The cost of taking a step is so high that the agent heads straight into the nearest terminal state, even if this is (2,4) (reward -1)

Rewards and Optimal Policy

- Optimal Policy when -0.427 < r < -0.085

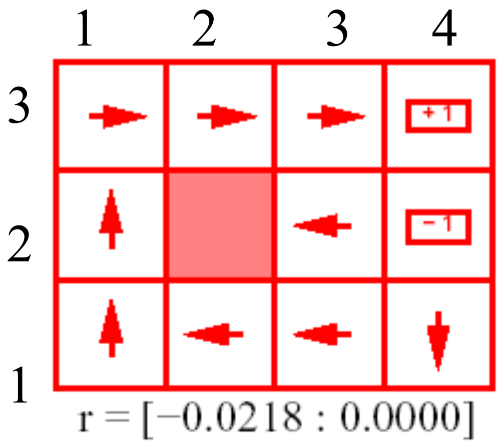


r = [−0.4278 : −0.0850]

- The cost of taking a step is high enough to make the agent take the shortcut to (3,4) from (1,3)

Rewards and Optimal Policy

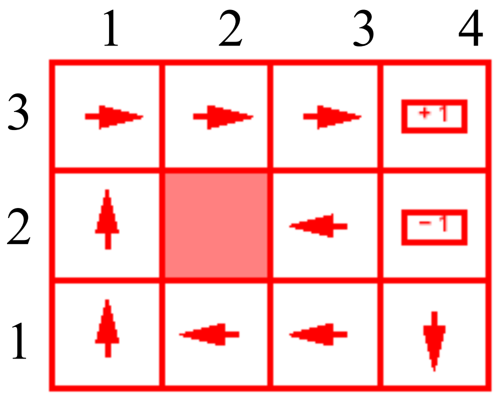
- Optimal Policy when -0.0218 < r < 0



- Why is the agent heading straight into the obstacle from (2,3)?

Rewards and Optimal Policy

- Optimal Policy when $-0.0218 < r < 0$



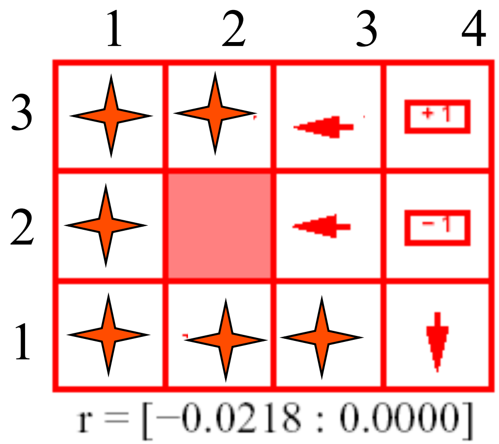
$r = [-0.0218 : 0.0000]$

- Stay longer in the grid is not penalized as much as before. The agent is willing to take longer routes to avoid (2,4)
  - This is true even when it means banging against the obstacle a few times when moving from (2,3)

Rewards and Optimal Policy

- Optimal Policy when r > 0

✦ state where every action belong to an optimal policy

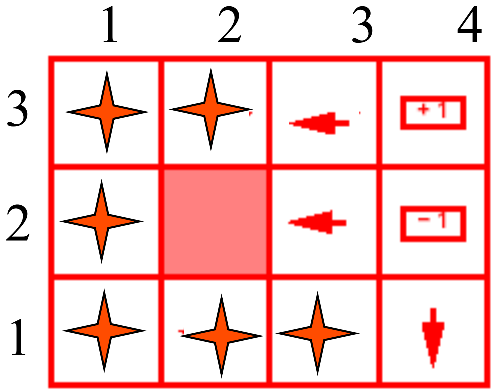


- What happens when the agent is rewarded for every step it takes?

Rewards and Optimal Policy

- Optimal Policy when r > 0

✦ state where every action belong to an optimal policy



r = [−0.0218 : 0.0000]

- What happens when the agent is rewarded for every step it takes?
  - it is basically rewarded for sticking around
  - The only actions that need to be specified are the ones in states that are adjacent to the terminal states: take the agent away from them

Types of Reward Functions

- Suppose the agent receives the sequence of rewards $r_1, r_2,\ldots,r_k$
- Ways to define the reward for this sequence, i.e. its *utility* for the agent (the textbook uses *value*, so will we):

Total (or additive) reward $V = \sum_{i=0}^{\infty} r_i$

Discounted reward

$V = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \ldots..$

where γ is the ***discount factor***, and $0 \le \gamma \le 1$

Note that this becames the total reward for $\gamma = 1$

Discounted Reward

- If we are dealing with infinite horizon, the environment histories can be infinitely long
  - their values will generally be infinite if we use additive rewards -> hard to compare
- The problem disappears if we use a discounted rewards with $\gamma < 1$ and reward for each state bounded by R_{max}

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

$$= \sum_{i=0}^{\infty} \gamma^i r_{i+1} \leq \sum_{i=0}^{\infty} \gamma^i R_{\max} = \frac{R_{\max}}{1-\gamma}$$

  - using the standard formula for an infinite geometric series
- Thus, with discounted reward, the utility of an infinite sequence is *finite*

Discounted Reward

- The discount factor γ determines the preference of the agent for current rewards vs. future rewards

- The closer γ is to 0, the less importance is given to future rewards
  - We could see the resulting policies as being “short sighted”, i.e. not taking much into account the consequences of the agent’s action in the distant future
  - For γ = 1, discounted rewards are equivalent to additive rewards

Property of Discounted Reward

- From now on, we will assume to be working with discounted rewards, possibly allowing γ = 1 if necessary

- Property
  - Be V_k the reward accumulated from time *t* (so V_1 is the reward accumulated from the beginning of the process)

$$V_k = r_k + \gamma r_{k+1} + \gamma^2 r_{k+2} + ... =$$

$$r_k + \gamma(r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + ..) = r_k + \gamma V_{k+1}$$

  - That is, the reward accumulated from time *t* is the immediate reward received at that time plus the discounted reward accumulated from time *t+1*

Overview

- Brief review of simpler decision making problems
  - one-off decisions
  - sequential decisions
- Decision processes and Markov Decision Processes (MDP)
- Rewards and Optimal Policies
- Defining features of Markov Decision Process
- Solving MDPs
  - Value Iteration

Value Iteration

- Algorithm to find an optimal policy and its value for a MDP
- We first need a couple of definitions
  - $V^{\pi}(s)$: the expected value of following policy π in state s
  - $Q^{\pi}(s, a)$, where a is an action: expected value of performing *a* in *s*, and then following policy π.
- We have, by definition

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

states reachable from s by doing a

Probability of getting to *s'* from *s* via *a*

reward of getting to *s'* from *s* via *a*

expected value of following policy *Pi* in s'

Value of a policy

- We can then compute $V^{\pi}(s)$ in terms of $Q^{\pi}(s, a)$

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s))$$

Expected value of following π in s

Expected value of performing the action indicated by π in s, and follow π after that

action indicated by π in s

Value of an optimal policy

- $Q^*(s, a)$: Expected value of performing a in s, and then following the optimal policy π*.

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V^*(s'))$$

- $V^*(s)$: the expected value of following optimal policy π* in state s
  - Obtained by choosing the action in s that maximizes *Q*(s,a)*

$$V^*(s) = \max_a Q^*(s,a)$$

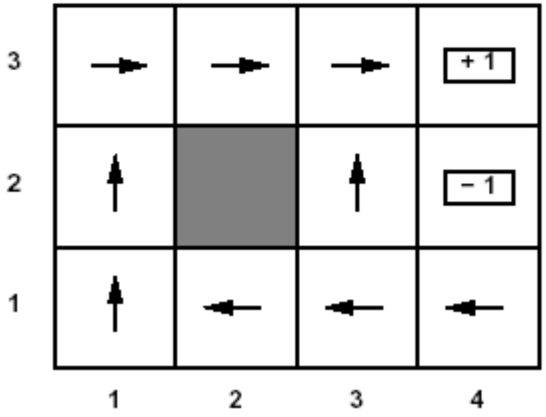
- Optimal policy π* is one that gives the best value for each state

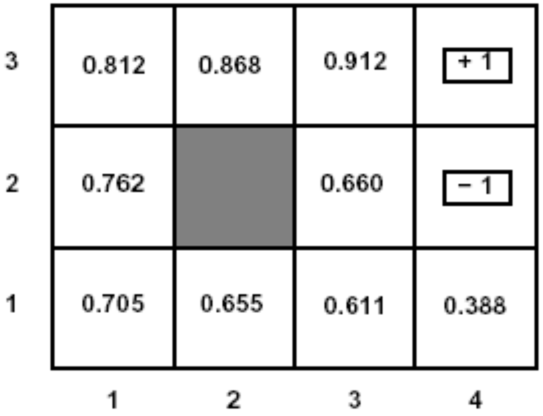
$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

- Note: V*(s) is essentially the Maximum Expected Utility (MEU) of a state

Value of an optimal policy

- Note that V*(s) and R(s) are quite different quantities
  - R(s) is the short term reward related to s
  - V*(s) is the expected long term total reward from s onward if following π*
- Example: V*(s) for the optimal policy we found for the grid problem with *R(s non-terminal)* = -0.04 and γ =1

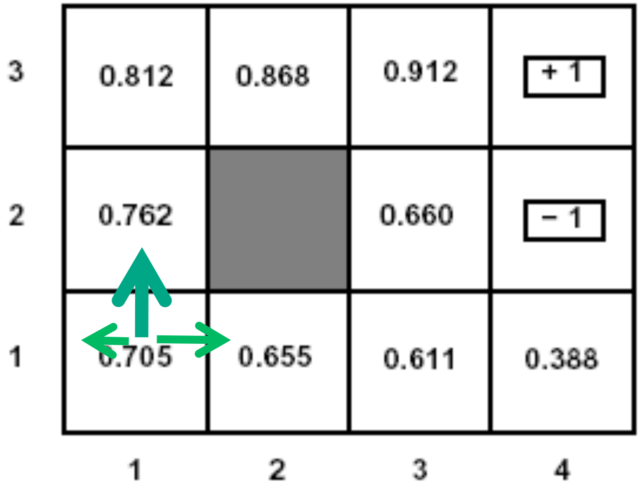




3	0.812	0.868	0.912	+ 1
2	0.762		0.660	− 1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Example

$$V^*(s) = \max_a Q^*(s,a) \;=\; \max_a \sum_{s'} P(s'|\,s,a)(R(s,a,s') + \gamma V^*(s'))$$



	1	2	3	4
3	0.812	0.868	0.912	+1
2	0.762	▨	0.660	−1
1	0.705	0.655	0.611	0.388

- To find the best action in (1,1)

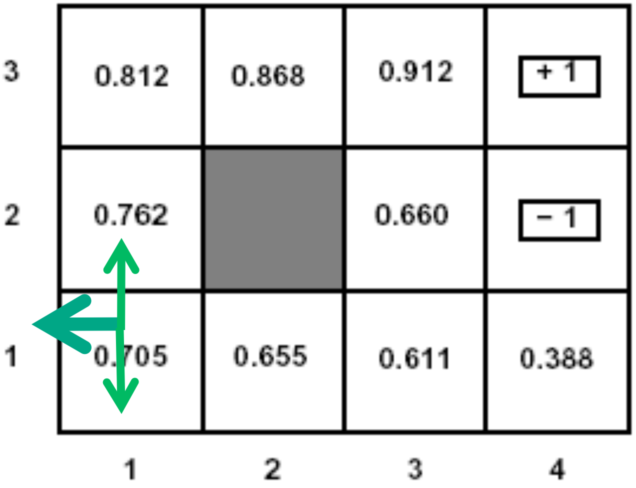
$$\pi^*(1,1) = \arg\max \begin{bmatrix} 0.8[R(2,1) + \gamma V^*(2,1)] + 0.1[(R(1,2) + \gamma V^*(1,2)] + 0.1[(R(1,1) + \gamma V^*(1,1)] & UP \\ 0.9[R(1,1) + \gamma V^*(1,1)] + 0.1[(R(2,1) + \gamma V^*(2,1)] & LEFT \\ 0.9[R(1,1) + \gamma V^*(1,1)] + 0.1[(R(1,2) + \gamma V^*(1,2)] & DOWN \\ 0.8[R(1,2) + \gamma V^*(2,1)] + 0.1[(R(2,1) + \gamma V^*(1,2)] + 0.1[(R(1,1) + \gamma V^*(1,1)] & RIGHT \end{bmatrix}$$

- Plugging in the numbers from the game state above, give *Up* as best action

- And using the value for *UP*, we can update V*(1,1)

Example

$$V^*(s) = \max_a Q^*(s,a) \;=\; \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V^*(s'))$$



	1	2	3	4
3	0.812	0.868	0.912	+ 1
2	0.762	▨	0.660	− 1
1	0.705	0.655	0.611	0.388

- To find the best action in (1,1)

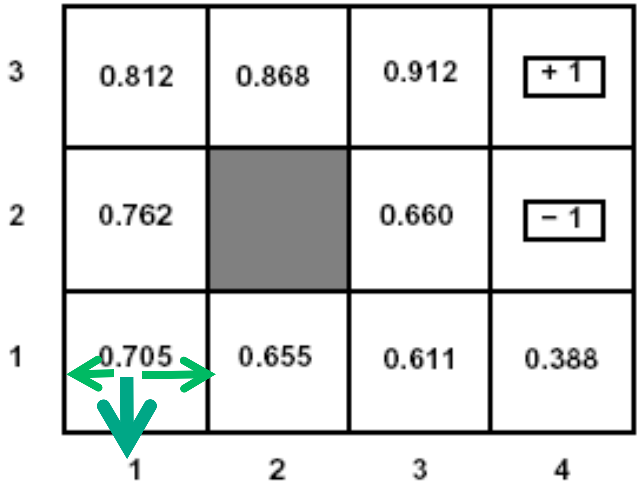
$$\pi^*(1,1) = \arg\max \left[\begin{array}{ll} 0.8[R(2,1) + \gamma V^*(2,1)] + 0.1[(R(1,2) + \gamma V^*(1,2)] + 0.1[(R(1,1) + \gamma V^*(1,1)] & UP \\ 0.9[R(1,1) + \gamma V^*(1,1)] + 0.1[(R(2,1) + \gamma V^*(2,1)] & LEFT \\ 0.9[R(1,1) + \gamma V^*(1,1)] + 0.1[(R(1,2) + \gamma V^*(1,2)] & DOWN \\ 0.8[R(1,2) + \gamma V^*(2,1)] + 0.1[(R(2,1) + \gamma V^*(1,2)] + 0.1[(R(1,1) + \gamma V^*(1,1)] & RIGHT \end{array}\right]$$

- Plugging in the numbers from the game state above, give *Up* as best action

- And using the value for *UP*, we can update V*(1,1)

Example

$$V^*(s) = \max_a Q^*(s,a) \;=\; \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V^*(s'))$$



3	0.812	0.868	0.912	+1
2	0.762	▨	0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

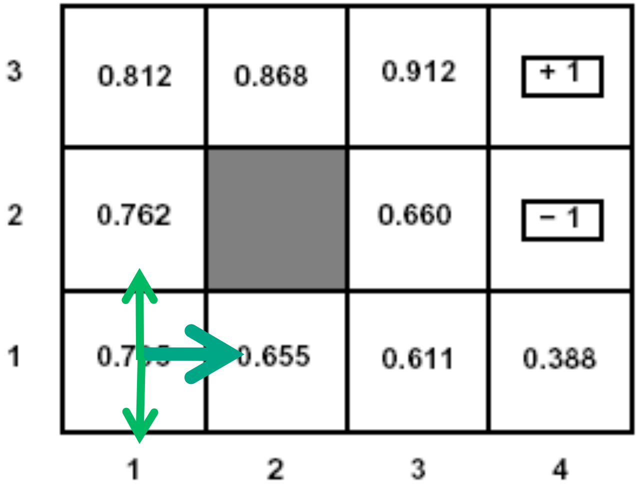
- To find the best action in (1,1)

$$\pi^*(1,1) = \arg\max \begin{bmatrix} 0.8[R(2,1) + \gamma V^*(2,1)] + 0.1[(R(1,2) + \gamma V^*(1,2)] + 0.1[(R(1,1) + \gamma V^*(1,1)] & \quad UP \\ 0.9[R(1,1) + \gamma V^*(1,1)] + 0.1[(R(2,1) + \gamma V^*(2,1)] & \quad LEFT \\ 0.9[R(1,1) + \gamma V^*(1,1)] + 0.1[(R(1,2) + \gamma V^*(1,2)] & \quad DOWN \\ 0.8[R(1,2) + \gamma V^*(2,1)] + 0.1[(R(2,1) + \gamma V^*(1,2)] + 0.1[(R(1,1) + \gamma V^*(1,1)] & \quad RIGHT \end{bmatrix}$$

- Plugging in the numbers from the game state above, give *Up* as best action
- And using the value for *UP*, we can update V*(1,1)

Example

$$V^*(s) = \max_a Q^*(s,a) \; = \; \max_a \sum_{s'} P(s'|\,s,a)(R(s,a,s') + \gamma V^*(s'))$$



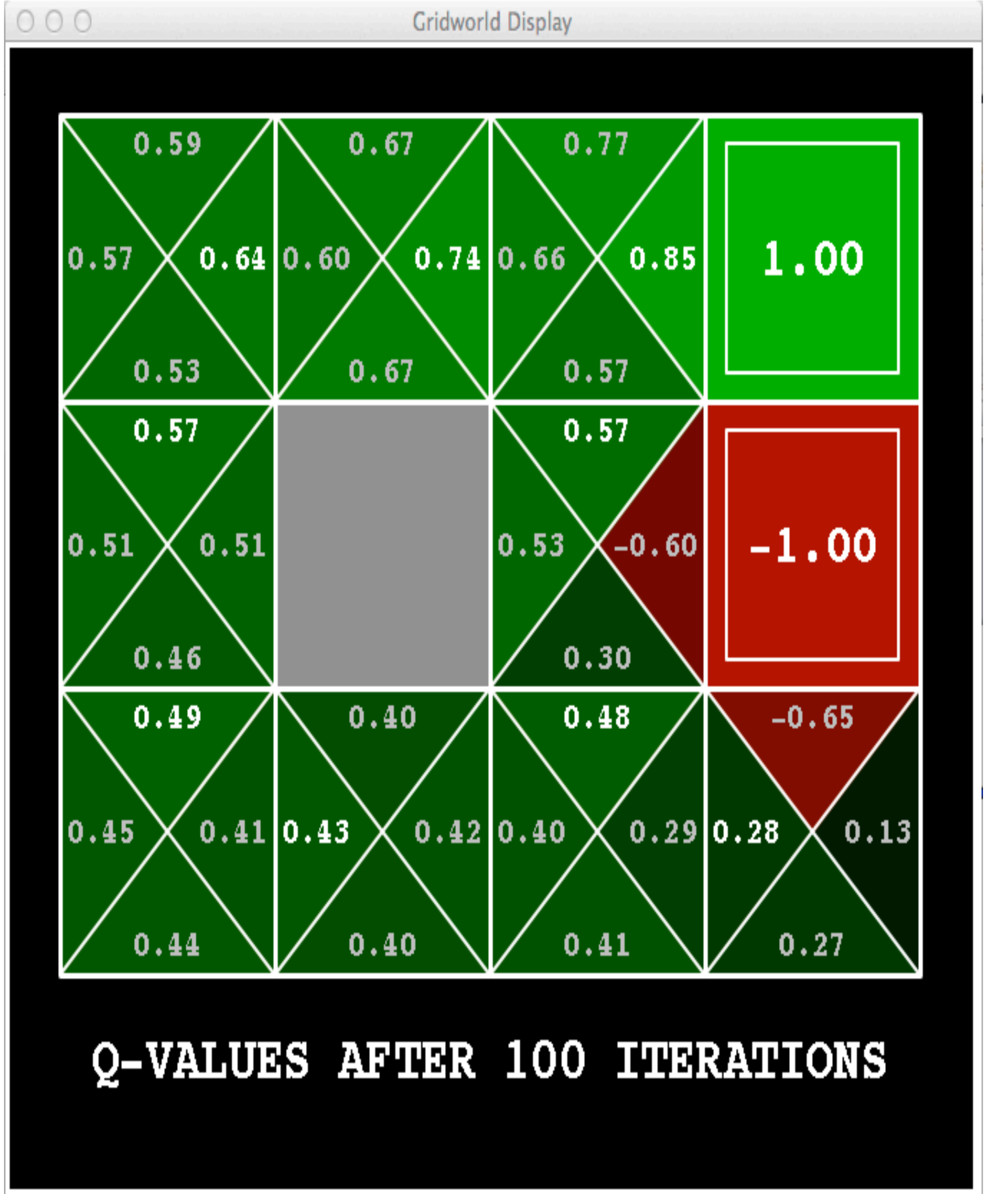
	1	2	3	4
3	0.812	0.868	0.912	+ 1
2	0.762		0.660	− 1
1	0.705	0.655	0.611	0.388

- To find the best action in (1,1)

$$\pi^*(1,1) = \arg\max \begin{bmatrix} 0.8[R(2,1) + \gamma V^*(2,1)] + 0.1[(R(1,2) + \gamma V^*(1,2)] + 0.1[(R(1,1) + \gamma V^*(1,1)] & UP \\ 0.9[R(1,1) + \gamma V^*(1,1)] + 0.1[(R(2,1) + \gamma V^*(2,1)] & LEFT \\ 0.9[R(1,1) + \gamma V^*(1,1)] + 0.1[(R(1,2) + \gamma V^*(1,2)] & DOWN \\ 0.8[R(1,2) + \gamma V^*(2,1)] + 0.1[(R(2,1) + \gamma V^*(1,2)] + 0.1[(R(1,1) + \gamma V^*(1,1)] & RIGHT \end{bmatrix}$$

- Plugging in the numbers from the game state above, give *Up* as best action
- And using the value for *UP*, we can update V*(1,1)

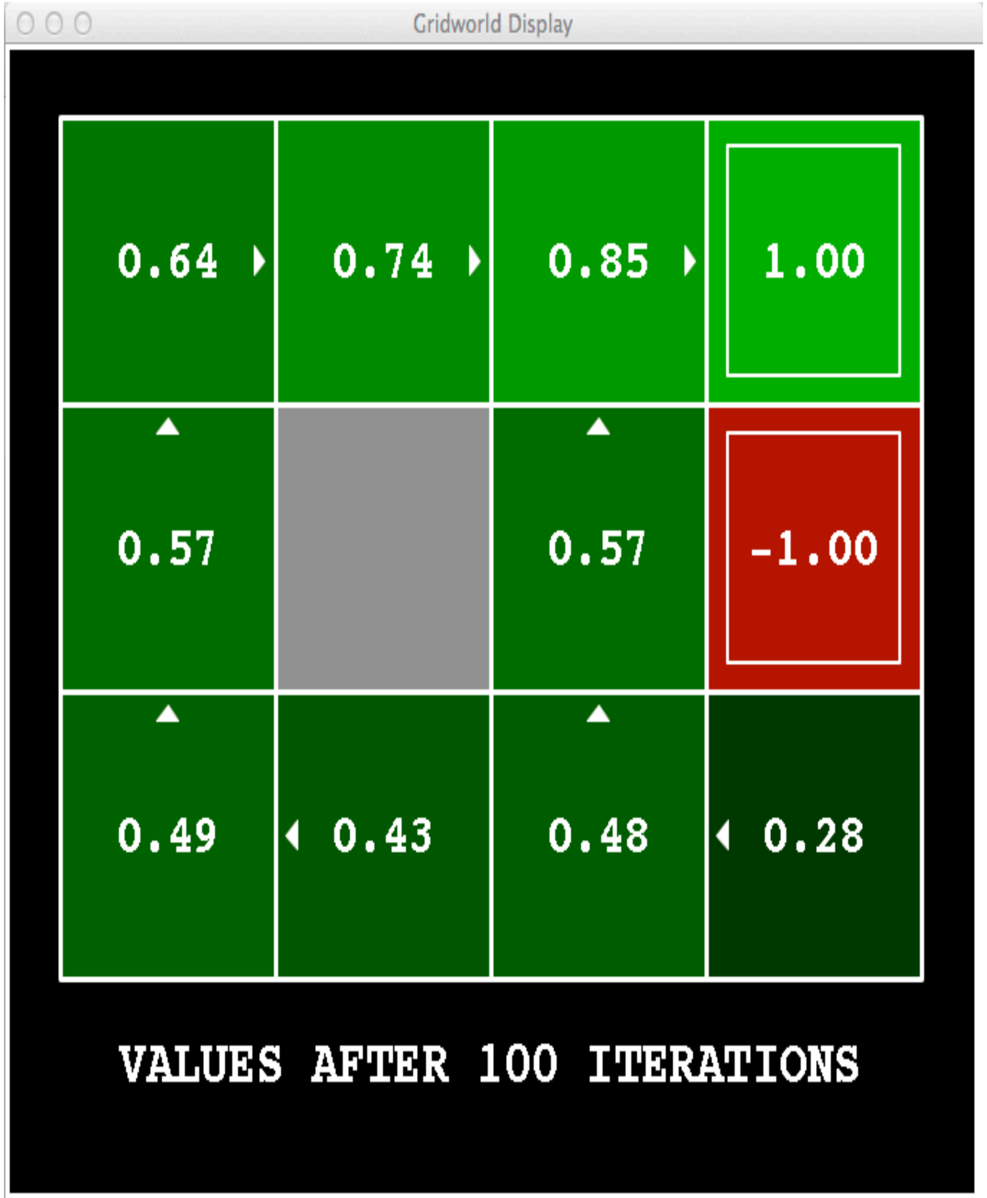
Snapshot of Demo – Gridworld Q Values



0.59 0.57 0.64 0.53	0.67 0.60 0.74 0.67	0.77 0.66 0.85 0.57	1.00
0.57 0.51 0.51 0.46		0.57 0.53 -0.60 0.30	-1.00
0.49 0.45 0.41 0.44	0.40 0.43 0.42 0.40	0.48 0.40 0.29 0.41	-0.65 0.28 0.13 0.27

Q-VALUES AFTER 100 ITERATIONS

Snapshot of Demo – Gridworld V Values



0.64 ▸	0.74 ▸	0.85 ▸	1.00
▲ 0.57		▲ 0.57	-1.00
▲ 0.49	◂ 0.43	▲ 0.48	◂ 0.28

VALUES AFTER 100 ITERATIONS

But How to Find the Utilities?

- Given *N* states, w can write an equation like the one below for each of them

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V^*(s'))$$

- Each equation contains *n* unknowns – the V* values for the *n* states
- N equations in N variables (Bellman equations)
  - It can be shown that they have a unique solution: the values for the optimal policy
- Unfortunately the N equations are non-linear, because of the max operator
  - Cannot be easily solved by using techniques from linear algebra
- Value Iteration Algorithm
  - Iterative approach to find the optimal policy and corresponding values

Value of Iteration: General Idea

- Start with arbitrary expected rewards on each states $V^{(0)}(s)$
- Repeat simultaneously for every *s* until there is "no change"
  - Calculate the right end side of the equation using $V^{(k)}$ computed at the previous iteration k

$$V^{(k+1)}(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V^{(k)}(s'))$$

  - and use the result as the new value for V, $V^{(k+1)}$
- True "no change" in the values of V(s) from one iteration to the next are guaranteed only if run for infinitely long.
  - In the limit, this process converges to a unique set of solutions for the Bellman equations
  - They are the total expected rewards (utilities) for the optimal policy

Example

- Suppose, for instance, that we start with values V$^{(0)}$(s) that are all 0, except in the terminal states (V$^{(0)}$(3,4))=+1, V(2,4))=-1

Iteration 0

3	0	0	0	+1
2	0	▨	0	-1
1	0	0	0	0
	1	2	3	4

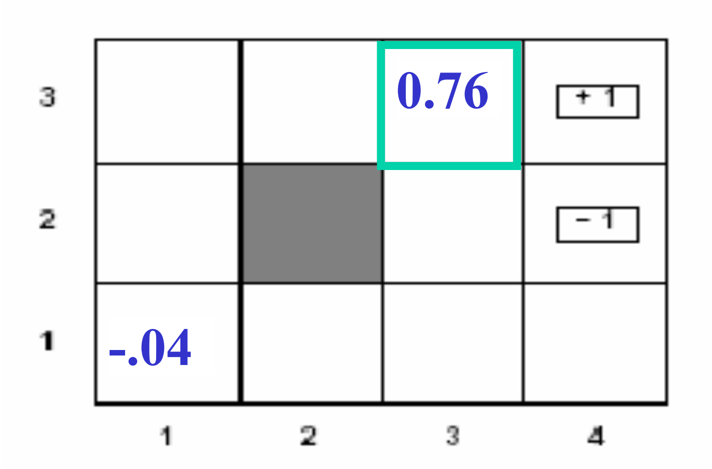
Iteration 1

3	0	0	0	+1
2	0	▨	0	-1
1	-0.04	0	0	0
	1	2	3	4

$$V^{(1)}(1,1) = \max \begin{bmatrix} 0.8[R(2,1)+\gamma V^{(0)}(2,1)] + 0.1[(R(1,2)+\gamma V^{(0)}(1,2)] + 0.1[(R(1,1)+\gamma V^{(0)}(1,1)] & UP \\ 0.9[R(1,1)+\gamma V^{(0)}(1,1)] + 0.1[(R(2,1)+\gamma V^{(0)}(2,1)] & LEFT \\ 0.9[R(1,1)+\gamma V^{(0)}(1,1)] + 0.1[(R(1,2)+\gamma V^{(0)}(1,2)] & DOWN \\ 0.8[R(1,2)+\gamma V^{(0)}(2,1)] + 0.1[(R(2,1)+\gamma V^{(0)}(1,2)] + 0.1[(R(1,1)+\gamma V^{(0)}(1,1)] & RIGHT \end{bmatrix}$$

$$V^{(1)}(1,1) = \max \begin{bmatrix} 0.8R(2,1)+0.1R(1,2)+0.1R(1,1) \quad = -0.04 & UP \\ 0.9R(1,1)+0.1R(2,1) = -0.04 & LEFT \\ 0.9R(1,1)+0.1R(1,2) = -0.04 & DOWN \\ 0.8R(1,2)+0.1R(2,1)+0.1(R(1,1) = -0.04 & RIGHT \end{bmatrix}$$

Example (cont'd)

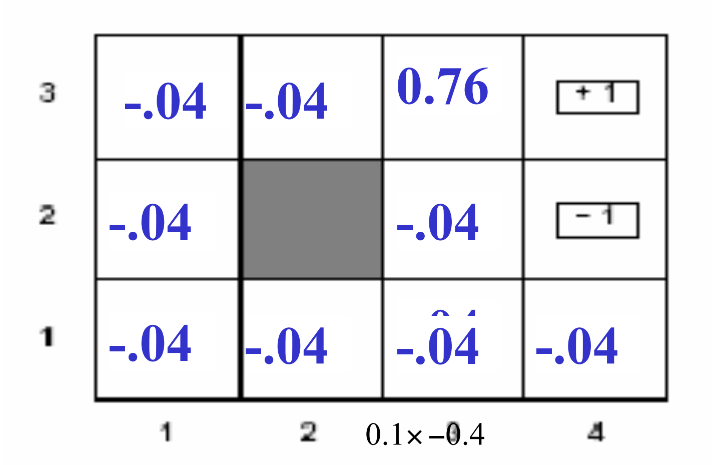


0.1×-0.4

$$V^{(1)}(3,3)=\max\begin{bmatrix} 0.8[R(3,3)+\gamma V^{(0)}(3,3)]+0.1[(R(3,2)+\gamma V^{(0)}(3,2)]+0.1[(R(3,4)+\gamma V^{(0)}(3,4)] & UP \\ 0.8[R(3,2)+\gamma V^{(0)}(3,2)]+0.1[(R(3,3)+\gamma V^{(0)}(3,3)]+0.1[(R(2,3)+\gamma V^{(0)}(2,3)] & LEFT \\ 0.8[R(2,3)+\gamma V^{(0)}(2,3)]+0.1[(R(3,2)+\gamma V^{(0)}(3,2)]+0.1[(R(3,4)+\gamma V^{(0)}(3,4)] & DOWN \\ 0.8[R(3,4)+\gamma V^{(0)}(3,4)]+0.1[(R(1,2)+\gamma V^{(0)}(1,2)+0.1[(R(3,3)+\gamma V^{(0)}(3,3)] & RIGHT \end{bmatrix}$$

$$V^{(1)}(3,3)=\max\begin{bmatrix} 0.8\times -0.04+0.1\times -0.04+0.1\times 0.96 = 0.06 & UP \\ 0.8\times -0.04+0.1\times -0.04+0.1\times -0.04 = -0.04 & LEFT \\ 0.8\times -0.04+0.1\times -0.04+0.1\times 0.96 = 0.06 & DOWN \\ 0.8\times 0.96+0.1\times -0.4+0.1\times -0.4 = 0.76 & RIGHT \end{bmatrix}$$

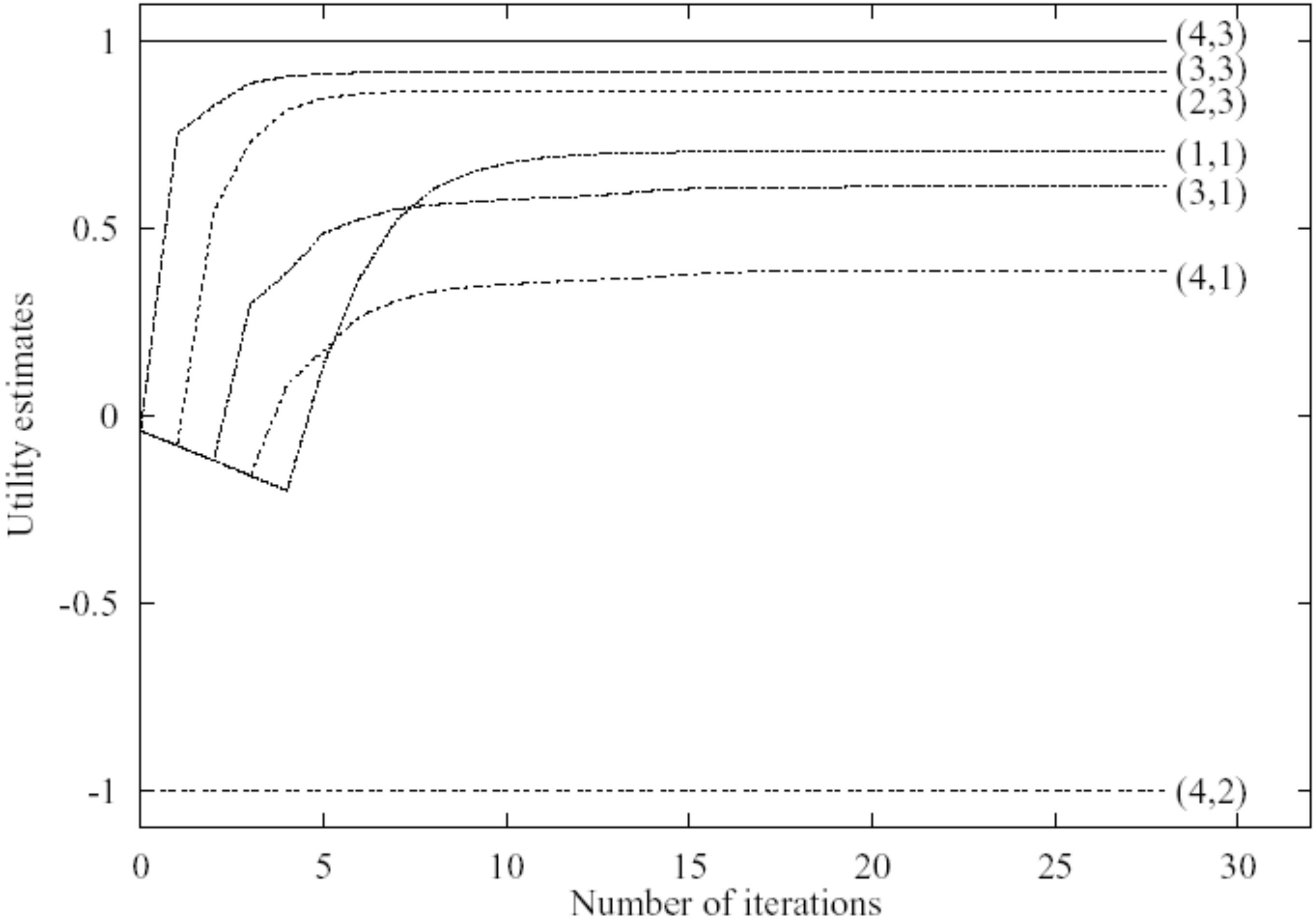
After a Full Iteration



3	-.04	-.04	0.76	+ 1
2	-.04	▨	-.04	− 1
1	-.04	-.04	-.04	-.04
	1	2		4

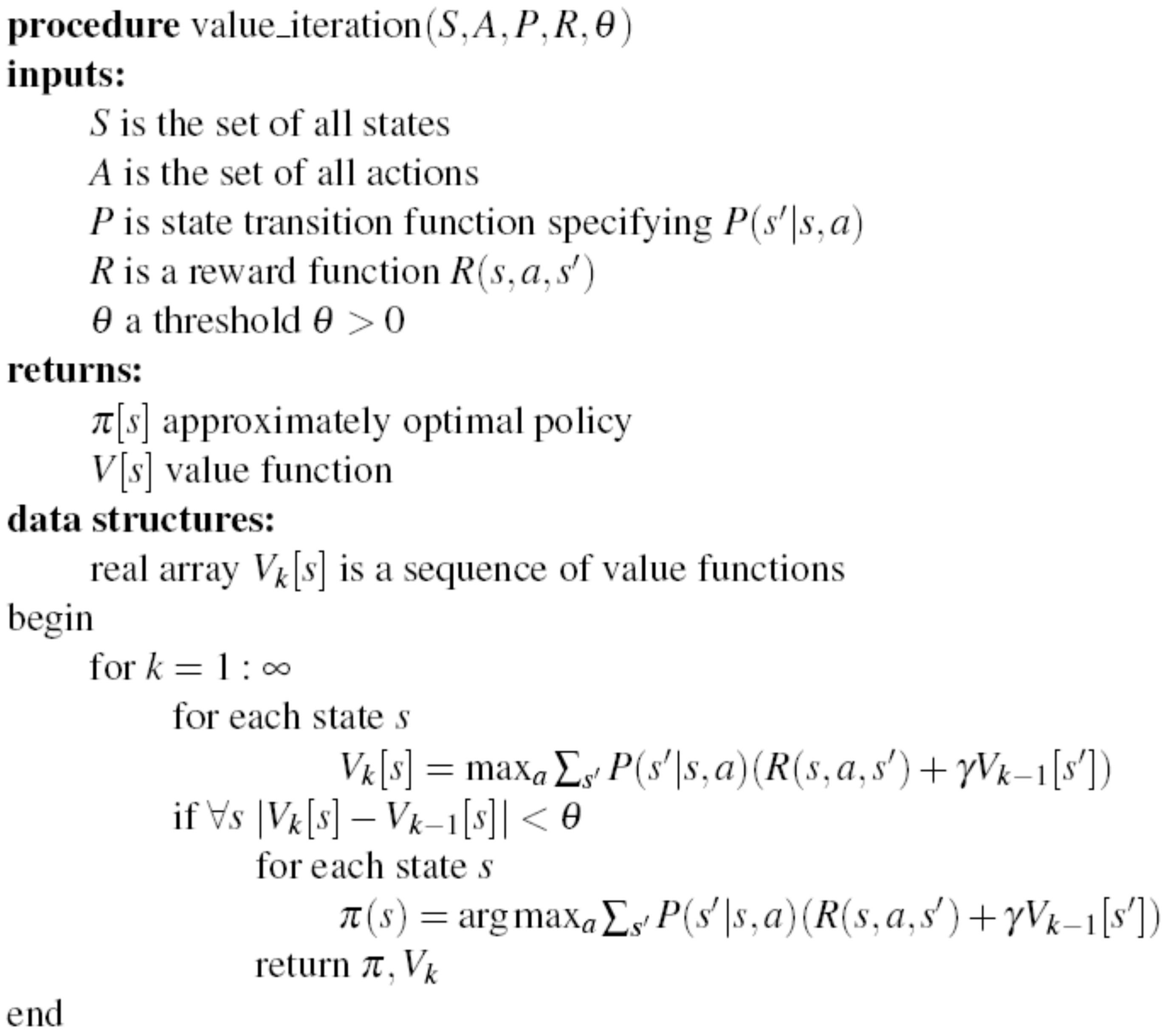
0.1× −0.4

State Utilities as Function of Iteration



- Note that utilities of states at different distances from (3,4) accumulate negative rewards until a path to (3,4) is found

Value Iteration Algorithm (storing V(s))



```
procedure value_iteration($S, A, P, R, \theta$)
inputs:
    $S$ is the set of all states
    $A$ is the set of all actions
    $P$ is state transition function specifying $P(s'|s,a)$
    $R$ is a reward function $R(s,a,s')$
    $\theta$ a threshold $\theta > 0$
returns:
    $\pi[s]$ approximately optimal policy
    $V[s]$ value function
data structures:
    real array $V_k[s]$ is a sequence of value functions
begin
    for $k = 1 : \infty$
        for each state $s$
            $V_k[s] = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_{k-1}[s'])$
        if $\forall s \; |V_k[s] - V_{k-1}[s]| < \theta$
            for each state $s$
                $\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_{k-1}[s'])$
            return $\pi, V_k$
end
```

Video of Demo Q-Learning -- Gridworld

Video of Demo Q-Learning -- Crawler

Example: Learning to Walk

[Video: AIBO W

Example: Learning to Walk

[Video: AIBO W

Example: Learning to Walk

[Video: AIBO W

Video of Demo Crawler Bot

Convergence of Value Iteration

- Need to define Max Norm:
  - Defines length of a vector V as the length of its biggest component

$$\|V\| = \max_s |V(s)| \qquad \text{and}$$

$\|V\text{-}V'\|$ is maximum distance between any two corresponding elements

- Be $V^{(k+1)}$ the value function on all states at the *k+1* iteration of the algorithm. It can be shown (see R&N, p621-623) that

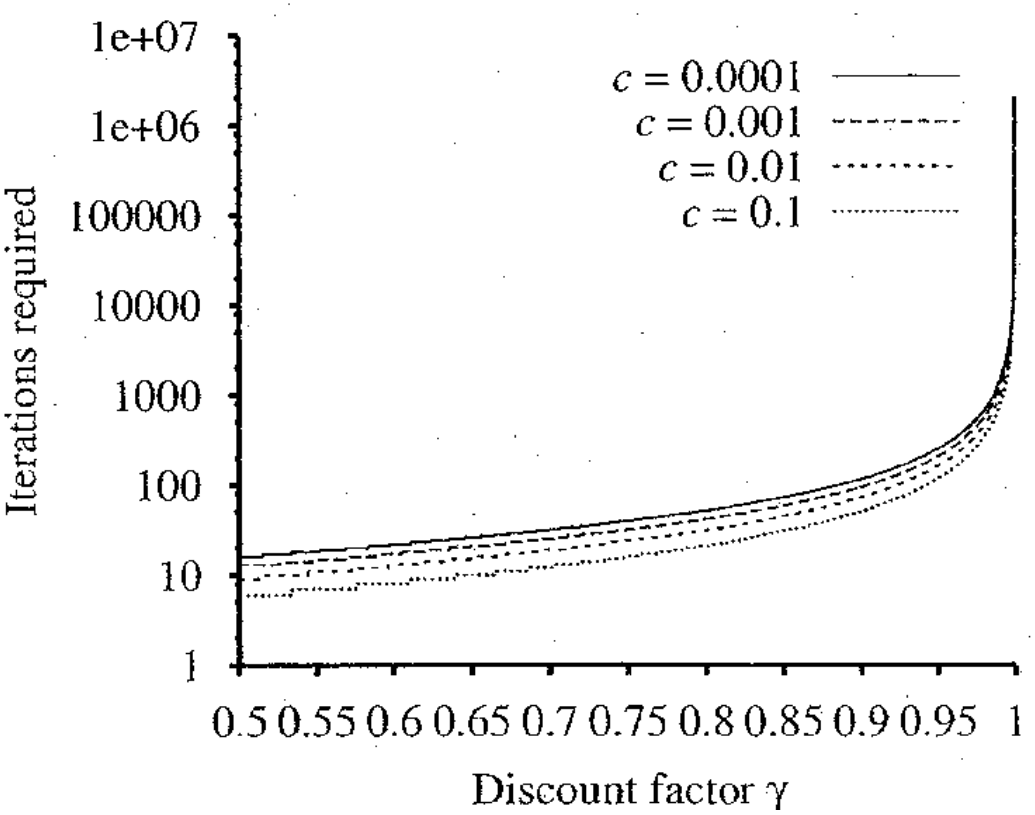
$$\|V^{(k+1)} \text{-} V^*\| < \gamma \|V^{(k)} \text{-} V^*\|$$

- So if we view the MaxNorm($V^{(k)}$ -V^*) as the error in the estimate of V^* at iteration *k*, Value iteration reduces it by γ at every iteration
  - Value iteration converges exponentially fast in *k*

Convergence of Value Iteration

- It can also be shown that, given an error ε, the number of iterations to reach this error is

$$N = \left[\frac{\log \dfrac{2R_{\max}}{\varepsilon(1-\gamma)}}{\underset{s}{\log(1/\gamma)}} \right]$$



- Picture shows how *N* varies with γ for different values of ε/$R_{\max}$ (*c* in the figure)

Convergence of Value Iteration

- Picture shows how *N* varies with γ for different values of ε/R_{max} (*c* in the figure)
  - Good news is that N does not depend too much on the ratio
  - Bad news is that N grows rapidly as γ gets close to 1
- We can make γ small, but this effectively means reducing how far ahead the agent plan its actions
  - short planning horizon can miss the long term effect of the agent actions

Convergence of Value Iteration

- Alternative way to decide when to stop value iteration
- It can be shown that,

$$if \ \left\| V^{(t+1)} - V^{(t)} \right\| < \frac{\varepsilon(1-\gamma)}{\gamma} \quad \text{then} \left\| V^{(t+1)} - V^{*} \right\| < \varepsilon$$

- The *if* side of the statement above is often used as a termination condition for value iteration

Convergence of Value Iteration

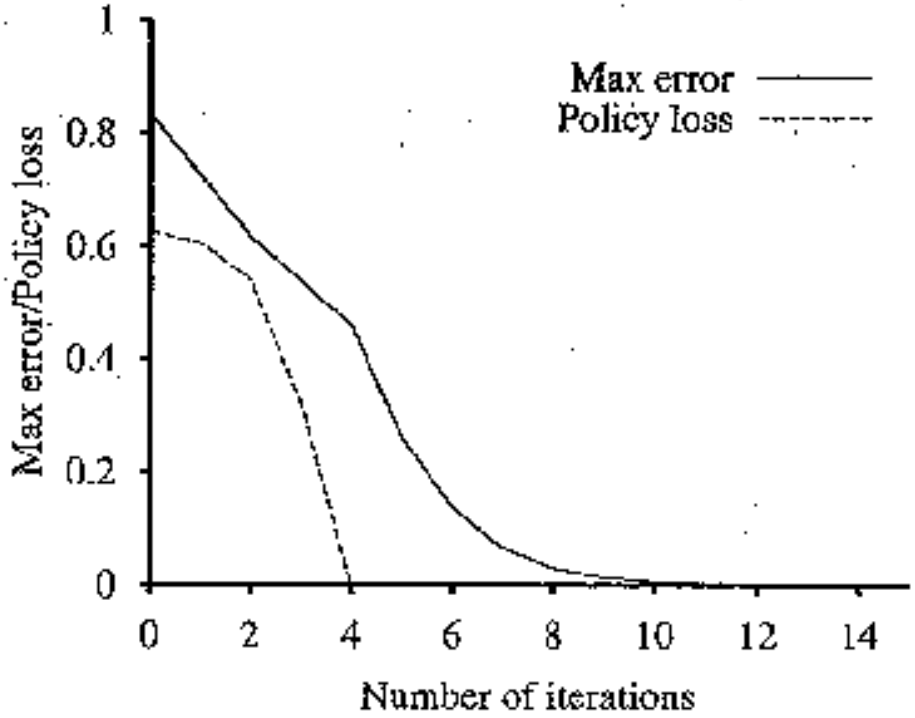
- Remember that after value iteration generates its estimate V*' of V*, one can choose the optimal policy (and corresponding value function) by applying the Bellman equation one more time

$$V^{(final)}(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V^{*\prime}(s'))$$

$$\pi^{(final)}(s) = \arg\max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V^{*\prime}(s'))$$

- In practice, it occurs that $\pi^{(final)}$ becomes optimal long before V*' has converged to V*

Policv Loss



- Max Error is the maximum error made by the estimate V*' as compared V*

  || V*' - V*||

- Policy Loss is the difference in total expected value when following $\pi^{(final)}$ instead of the optimal policy

  || $V^{n(final)}$ - V*||

Learning Goals for MDP

- Describe what is a Markov Decision Process, identify its key features and the components that are needed to fully specify an MDP
- Define policy, optimal policy, reward, reward function and explain how these concepts are related.
- Distinguish different types of planning horizons
- Identify different types of reward functions (additve, discounted) .Compare how different types of functions or different reward values affect the optimal policy when applied to a given decision problem
- Explain/write in pseudocode/implement/trace/debug/ value iteration (VI)
- Demonstrate why VI converges, and derive how convergence varies with gamma and error on optimal policy.