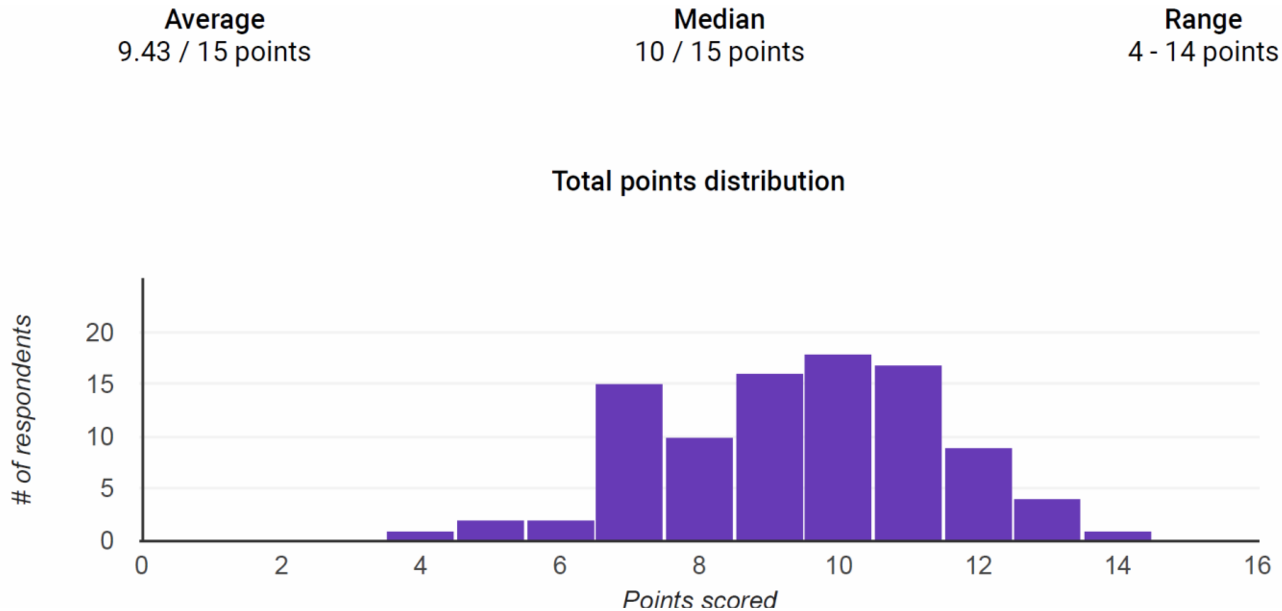
Intelligent Systems (AI-2)

Computer Science cpsc422, Lecture 4

Sep, 15, 2017

Announcements



Office Hours have been posted

- **Giuseppe Carenini** ICICS (CICSR) 105 Wed 1-2
- **Jordon Johnson** ICCS X237, Th 10-12
- **Siddhesh Khandelwal** ICCS X237, W 2-4
- **Dave Johnson** ICCS X237, F 11-12

What to do with readings? In a few lectures we will discuss the first research paper. Instructions on what to do are available on the course webpage.

Last year Conference (program co-chair)

Four papers using

(PO)MDP & Reinforcement Learning!

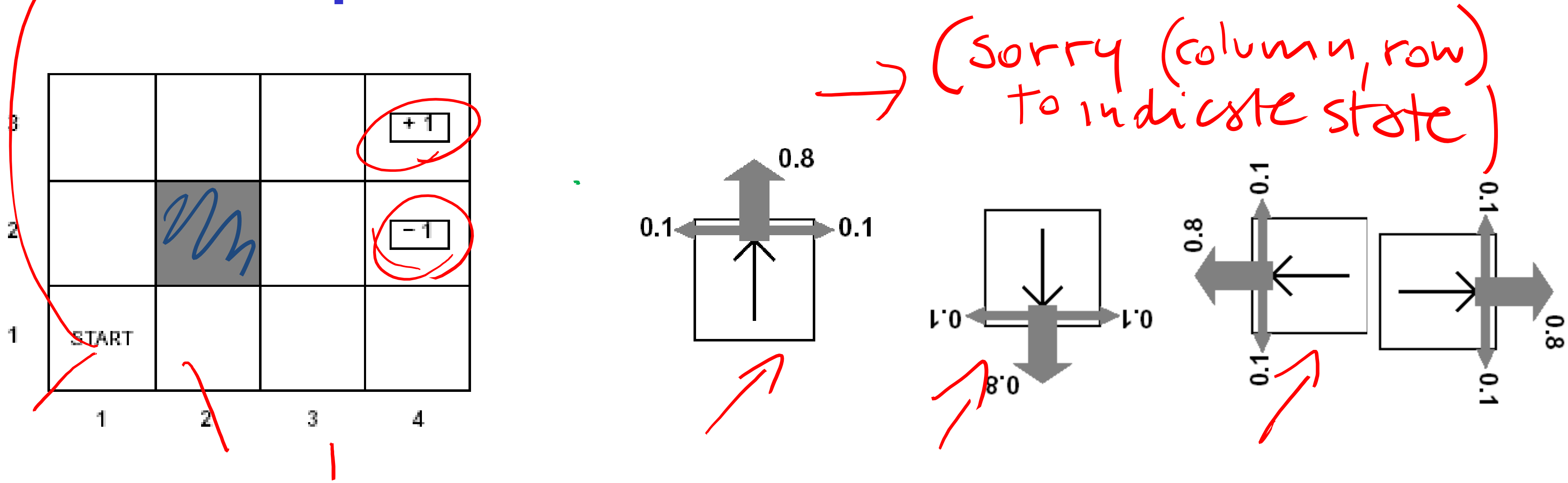
Four papers this year as well……

Lecture Overview

Markov Decision Processes

- Some ideas and notation
- Finding the Optimal Policy
  - Value Iteration
- From Values to the Policy
- Rewards and Optimal Policy

Example MDP: Scenario and Actions



Agent moves in the above grid via actions *Up, Down, Left, Right*

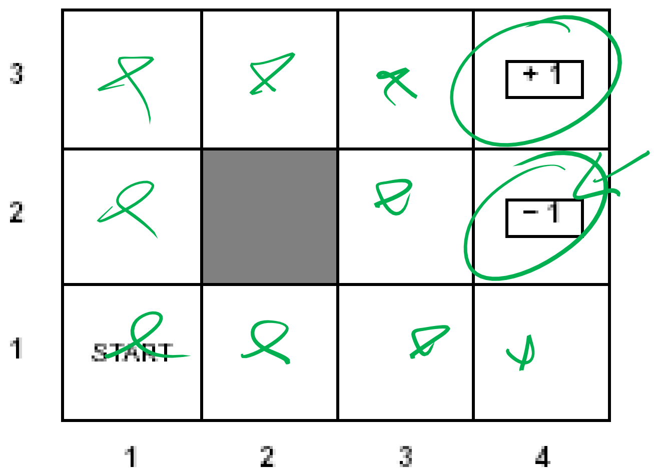
Each action has:

- 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 says there

Eleven states

Two terminal states (4,3) and (4,2)

Example MDP: Rewards



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

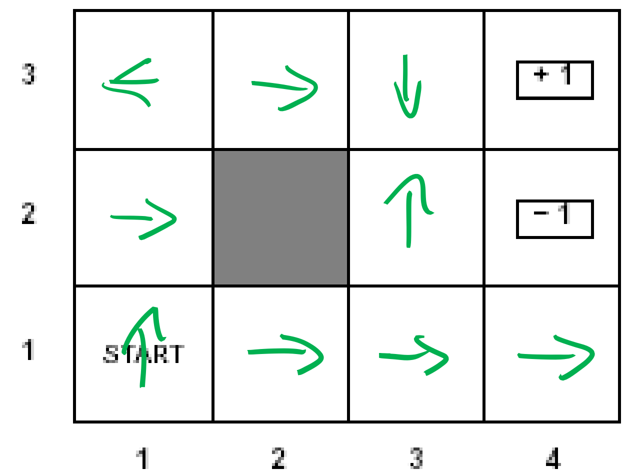
MDPs: Policy

- 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?

policy

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



Sketch of ideas to find the optimal policy for a MDP (Value Iteration)

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

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

reward obtained in *s*

Discount factor

states reachable from *s* by doing *a*

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

expected value of following policy π in *s'*

Value of a policy and Optimal policy

We can also 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 following π after that

action indicated by π in s

For the optimal policy π^* we also have

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

Value of Optimal policy

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

Remember for any policy π

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

But the Optimal policy π^* is the one that gives the action that maximizes *the future reward* for each state

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

So…

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

Value Iteration Rationale

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

$$V(s_1) = R(s_1) + \gamma \max_a \sum_{s'} P(s' \mid s_1, a) V(s')$$

$$V(s_2) = R(s_2) + \gamma \max_a \sum_{s'} P(s' \mid s_2, a) V(s')$$

- 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 V values and the corresponding
- optimal policy

Value Iteration in Practice

- Let $V^{(i)}(s)$ be the utility of state *s* at the i^{th} iteration of the algorithm
- Start with arbitrary utilities on each state *s*: $V^{(0)}(s)$
- Repeat simultaneously for every *s* until there is "no change"

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

- 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

(sorry (column, row) to indicate state)

- Suppose, for instance, that we start with values $V^{(0)}(s)$ that are all 0

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	$\boxed{-0.04}$	0	0	0
	1	**2**	**3**	**4**

$$V^{(1)}(1,1) = -0.04 + \overset{\gamma}{1} * \max \begin{bmatrix} 0.8V^{(0)}(1,2) + 0.1V^{(0)}(2,1) + 0.1V^{(0)}(1,1) & UP \\ 0.9V^{(0)}(1,1) + 0.1V^{(0)}(1,2) & LEFT \\ 0.9V^{(0)}(1,1) + 0.1V^{(0)}(2,1) & DOWN \\ 0.8V^{(0)}(2,1) + 0.1V^{(0)}(1,2) + 0.1V^{(0)}(1,1) & RIGHT \end{bmatrix}$$

$$V^{(1)}(1,1) = -0.04 + \max \begin{bmatrix} 0 & UP \\ 0 & LEFT \\ 0 & DOWN \\ 0 & RIGHT \end{bmatrix}$$

Example (cont'd)

(sorry (column, row) to indicate state)

- Let's compute $V^{(1)}(3,3)$

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	0.76	+1
2	0		0	-1
1	-0.04	0	0	0
	1	2	3	4

$$V^{(1)}(3,3) = -0.04 + 1 * \max \begin{bmatrix} 0.8V^{(0)}(3,3) + 0.1V^{(0)}(2,3) + 0.1V^{(0)}(4,3) & UP \\ 0.8V^{(0)}(2,3) + 0.1V^{(0)}(3,3) + 0.1V^{(0)}(3,2) & LEFT \\ 0.8V^{(0)}(3,2) + 0.1V^{(0)}(2,3) + 0.1V^{(0)}(4,3) & DOWN \\ 0.8V^{(0)}(4,3) + 0.1V^{(0)}(3,3) + 0.1V^{(0)}(3,2) & RIGHT \end{bmatrix}$$

$$V^{(1)}(3,3) = -0.04 + \max \begin{bmatrix} 0.1 & UP \\ 0 & LEFT \\ 0.1 & DOWN \\ 0.8 & RIGHT \end{bmatrix}$$

Example (cont'd)

(sorry (column, row) to indicate state)

- Let's compute $V^{(1)}(4,1)$

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	.76	+1
2	0		0	-1
1	-0.04	0	0	-0.04
	1	2	3	4

$$V^{(1)}(4,1) = -0.04 + \max \begin{bmatrix} 0.8V^{(0)}(4,2) + 0.1V^{(0)}(3,1) + 0.1V^{(0)}(4,1) & UP \\ 0.8V^{(0)}(3,1) + 0.1V^{(0)}(4,2) + 0.1V^{(0)}(4,1) & LEFT \\ 0.9V^{(0)}(4,1) + 0.1V^{(0)}(3,1) & DOWN \\ 0.9V^{(0)}(4,1) + 0.1V^{(0)}(4,2) & RIGHT \end{bmatrix}$$

$$V^{(1)}(4,1) = -0.04 + \max \begin{bmatrix} -0.8 & UP \\ -0.1 & LEFT \\ 0 & DOWN \\ -0.1 & RIGHT \end{bmatrix}$$

After a Full Iteration

Iteration 1

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

- Only the state one step away from a positive reward (3,3) has gained value, all the others are losing value

Some steps in the second iteration

Iteration 1

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

Iteration 2

	1	2	3	4
3	-.04	-.04	0.76	+1
2	-.04		-.04	-1
1	$\boxed{-0.08}$	-.04	-.04	-.04

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

$$V^{(2)}(1,1) = -0.04 + \max \begin{bmatrix} -.04 & UP \\ -.04 & LEFT \\ -.04 & DOWN \\ -.04 & RIGHT \end{bmatrix} = -0.08$$

Example (cont'd)

- Let's compute $V^{(2)}(2,3)$

Iteration 1

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

Iteration 2

	1	2	3	4
3	-.04	0.56	0.76	+1
2	-.04		-.04	-1
1	-0.08	-.04	-.04	-.04

$$V^{(2)}(2,3) = -0.04 + 1 * \max \begin{bmatrix} 0.8V^{(1)}(2,3) + 0.1V^{(1)}(1,3) + 0.1V^{(1)}(3,3) & UP \\ 0.8V^{(1)}(1,3) + 0.1V^{(0)}(2,3) + 0.1V^{(1)}(2,3) & LEFT \\ 0.8V^{(1)}(2,3) + 0.1V^{(1)}(1,3) + 0.1V^{(1)}(3,3) & DOWN \\ 0.8V^{(1)}(3,3) + 0.1V^{(1)}(2,3) + 0.1V^{(1)}(2,3) & RIGHT \end{bmatrix}$$

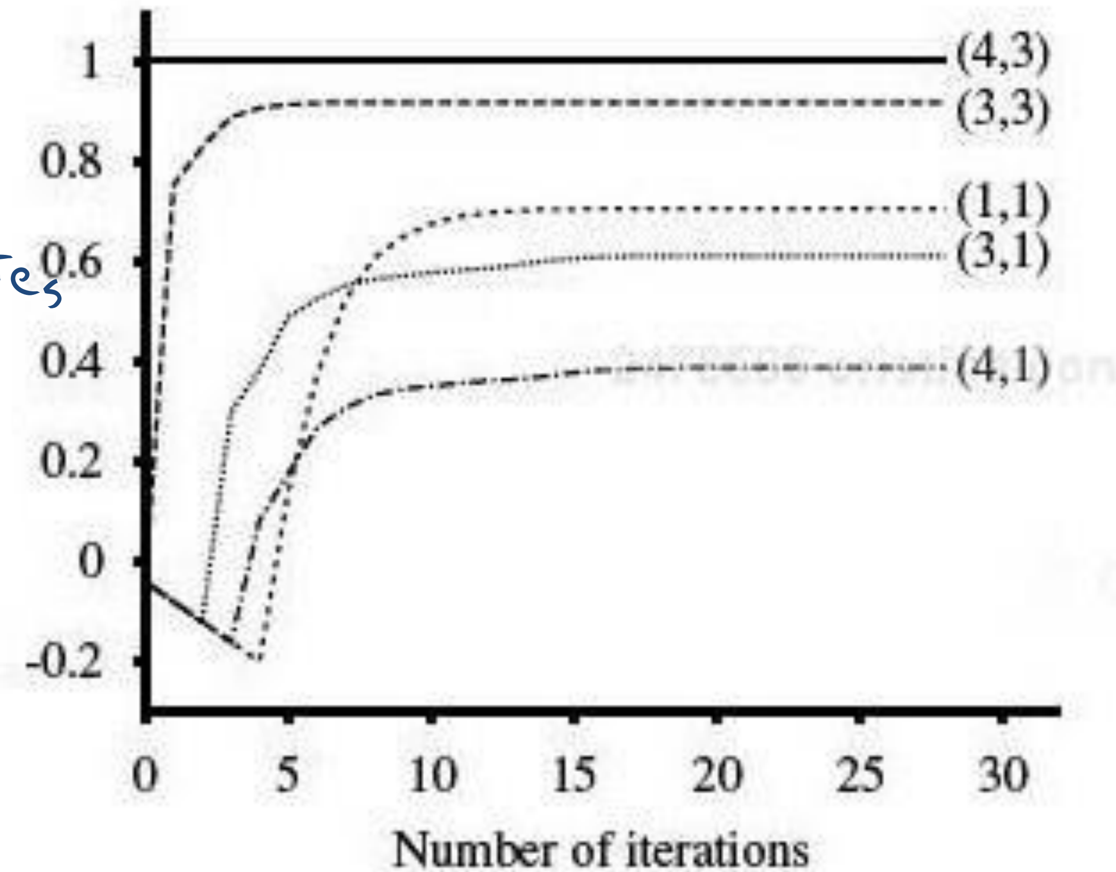
$$V^{(1)}(2,3) = -0.04 + (0.8 * 0.76 + 0.2 * -0.04) = 0.56$$

- Steps two moves away from positive rewards start increasing their value

State Utilities as Function of Iteration

(only for 5 states)

Estimates for V



		(3,3)	(4,3)
			(4,2)
(1,1)		(3,1)	(4,1)

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

Value Iteration: Computational Complexity

Value iteration works by producing successive approximations of the optimal value function.

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

What is the complexity of each iteration?

A. O(|A|²|S|) **B.** O(|A||S|²) **C.** O(|A|²|S|²)

…or faster if there is sparsity in the transition function.

small sets

Relevance to state of the art MDPs

FROM : Planning with Markov Decision Processes: AnAI Perspective Mausam (UW),Andrey Kolobov (MSResearch) Synthesis Lectures onArtificial Intelligence and Machine Learning Jun 2012

Free online through UBC

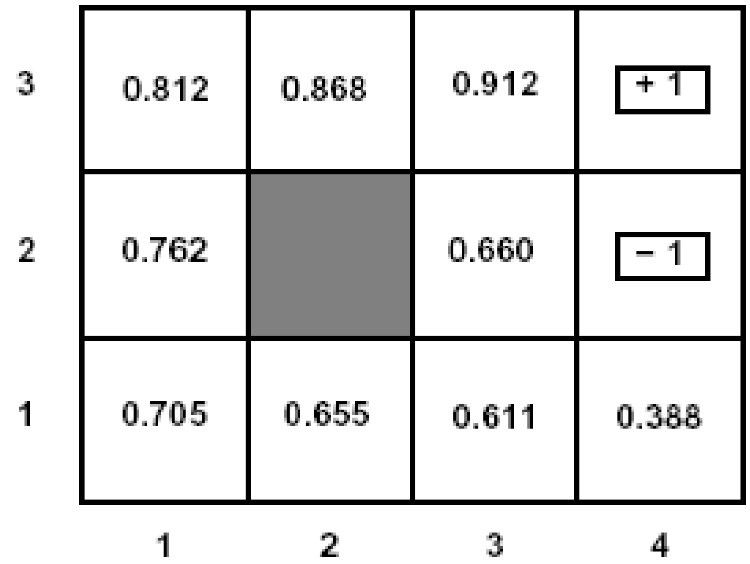
" Value Iteration (VI) forms the basis of most of the advanced MDP algorithms that we discuss in the rest of the book. ……..”

Lecture Overview

Markov Decision Processes

- ……
- Finding the Optimal Policy
  - Value Iteration
- **From Values to the Policy**
- Rewards and Optimal Policy

Value Iteration: from state values V to π^*



- Now the agent can chose the action that implements the **MEU principle**: maximize the expected utility of the subsequent state

Value Iteration: from state values V to π*

- Now the agent can chose the action that implements the MEU principle: maximize the expected utility of the subsequent state

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s' \mid s, a) V^{\pi^*}(s')$$

expected value of following policy π* in s'

states reachable from s by doing a

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

Example: from state values V to π*

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

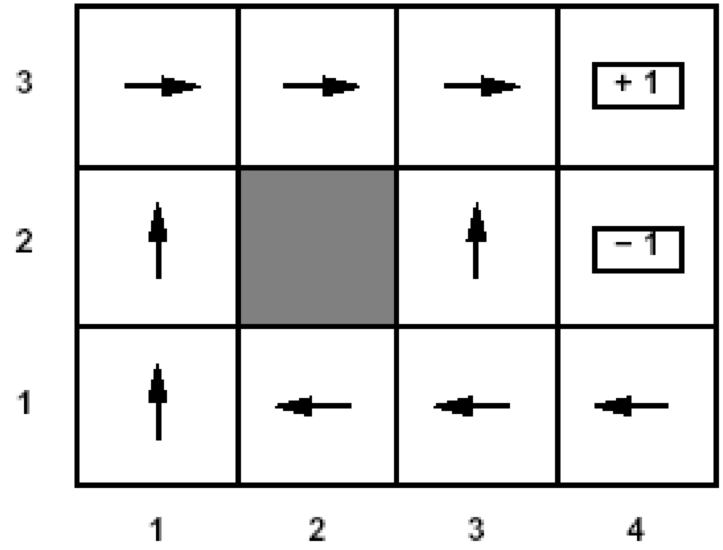
	1	2	3	4
3	0.812	0.868	0.912	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388

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

$$\pi^*(1,1) = \arg\max \left[\begin{array}{ll} 0.8\overset{.762}{\boxed{V(1,2)}} + 0.1\overset{.655}{\boxed{V(2,1)}} + 0.1\overset{.705}{\boxed{V(1,1)}} & UP \;\; * \\ 0.9\overset{.705}{\boxed{V(1,1)}} + 0.1\overset{.762}{\boxed{V(1,2)}} & LEFT \\ 0.9\overset{.705}{\boxed{V(1,1)}} + 0.1\overset{.655}{\boxed{V(2,1)}} & DOWN \\ 0.8V(2,1) + 0.1V(1,2) + 0.1V(1,1) & RIGHT \end{array} \right]$$

Optimal policy

- This is the policy that we obtain….



Learning Goals for today's class

You can:

Define/read/write/trace/debug the Value Iteration (VI) algorithm. Compute its complexity.

- Compute the Optimal Policy given the output of VI
- Explain influence of rewards on optimal policy

TODO for Mon

- **Read Textbook** 9.5.6 Partially Observable MDPs

- **Also Do Practice Ex. 9.C**

http://www.aispace.org/exercises.shtml