Lecture 13a: Markov Decision Processes

CSCI 360

Introduction to Artificial Intelligence

USC

Here is where we are…

	3/1		Project 2 Out	
9	3/4	3/5	Quantifying Uncertainty	[Ch 13.1-13.6]
	3/6	3/7	Bayesian Networks	[Ch 14.1-14.2]
10	3/11	3/12	*(spring break, no class)*	
	3/13	3/14	*(spring break, no class)*	
11	3/18	3/19	Inference in Bayesian Networks	[Ch 14.3-14.4]
	3/20	3/21	Decision Theory	[Ch 16.1-16.3 and 16.5]
	3/23		Project 2 Due	
12	3/25	3/26	***Advanced topics*** *(Chao traveling to National Science Foundation)*	
	3/27	3/28	***Advanced topics*** *(Chao traveling to National Science Foundation)*	
	3/29		Homework 2 Out	
13	4/1	4/2	Markov Decision Processes	[Ch 17.1-17.2]
	4/3	4/4	Decision Tree Learning	[Ch 18.1-18.3]
	4/5		Homework 2 Due	
	4/5		Project 3 Out	
14	4/8	4/9	Perceptron Learning	[Ch 18.7.1-18.7.2]
	4/10	4/11	Neural Network Learning	[Ch 18.7.3-18.7.4]
15	4/15	4/16	Statistical Learning	[Ch 20.2.1-20.2.2]
	4/17	4/18	Reinforcement Learning	[Ch 21.1-21.2]
16	4/22	4/23	Artificial Intelligence Ethics	
	4/24	4/25	Wrap-Up and Final Review	
	4/26		Project 3 Due	
	5/3	5/2	**Final Exam** (2pm-4pm)	

Outline

- What is AI?
- Part I: Search
- Part II: Logical reasoning
- **Part III: Probabilistic reasoning**
  - Quantifying Uncertainty
  - Bayesian Networks
  - Inference in Bayesian Networks
  - Decision Theory
  - **Markov Decision Processes**
- Part IV: Machine learning

Recap: *Making a rational decision*

Rational decision depends on

(1) The relative importance of various goals and

(2) likelihood that (and degree to which) they will be reached

Decision theory = Utility theory + Probability theory

Choose an action that yields the ***maximum expected utility (MEU)***, averaged over all the possible outcomes of the action, weighted by the probability

Recap: *Maximum expected utility (MEU)*

- Choosing an action that maximizes the expected utility

$$\boxed{action = \operatorname*{argmax}_{a} EU(a|\mathbf{e})}$$

- This principle defines all of AI

Given the evidence (e) and a set of actions, pick the action (a) that has the MEU

Recap: *Utility function (U)*

- Utility function, denoted ***U(s')***, expresses the desirability of the state ***s'***

- Example:

$$s' = \{\ getting\ A,\ getting\ C\ \}$$

$U(\ getting\ A\ ) = 4.0$

$U(\ getting\ C\ ) = 2.0$

Recap: *Expected utility (EU)*

- Expected utility, ***EU(a|e)***, is the weighted average

$$EU(a|\mathbf{e}) = \sum_{s'} P(\textsc{Result}(a) = s' \mid a, \mathbf{e})\, U(s')$$

*a: with professor **Y***   *e: taking the course*   ***s'*** = { *getting A*, *getting C* }

Example:

*P(**RESULT**(a) = getting A | a, e) * U(getting A) = 75% * 4.0*

*P(**RESULT**(a) = getting C | a, e) * U(getting C) = 25% * 2.0*

*EU(a|e) = 75%*4.0 + 25%*2.0*

Recap: *Example*

- *Taking a course with* ***Professor X***

getting B(3)

- *Taking a course with* ***Professor Y***

getting A(4) getting C(2)

✓ *Choose Professor Y (seeking the best case – 4.0)*
✓ *Choose Professor X (avoiding the worst case – 2.0)*

Which one do **you** choose?

Recap: *Example*

- *Taking a course with* ***Professor X***

getting B(3)

(100%)

3*100% = 3

- *Taking a course with* ***Professor Y***

getting A(4) getting C(2)

(75%) (25%)

4*75% + 2*25% = 3.5

SHOULD

Which one ~~do~~ you choose?

Recap: *Example*

- *Taking a course with* ***Professor X***

getting B(3)

(100%)

3*100% = 3

- *Taking a course with* ***Professor Y***

getting A(4) getting C(2)

(25%) (75%)

4*25% + 2*75% = 2.5

SHOULD

Which one ~~do~~ you choose?

Recap: *Maximum expected utility (MEU)*

- Choosing an action that maximizes the expected utility

$$\boxed{action = \underset{a}{\operatorname{argmax}}\, EU(a|\mathbf{e})}$$

- This principle defines all of AI

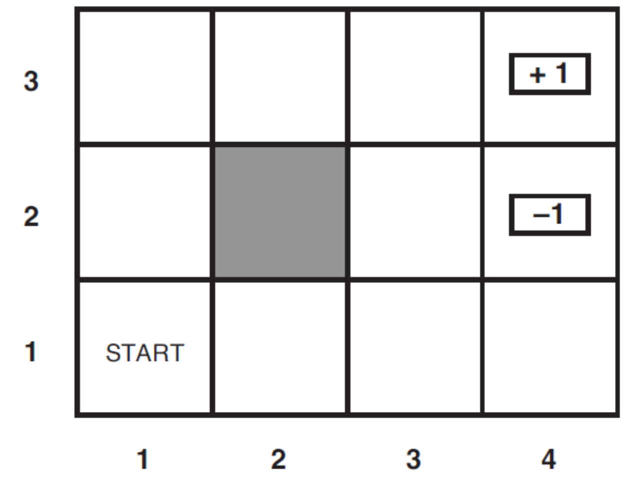
Given the evidence (e) and a set of actions, pick the action (a) that has the MEU

Outline of today's lecture

- Sequential Decision
  - A sequence of decisions (versus *one-shot* decision)
- Value Iteration
  - Algorithm for solving the sequential decision problem
- Policy Iteration
  - An alternative algorithm

Example problem

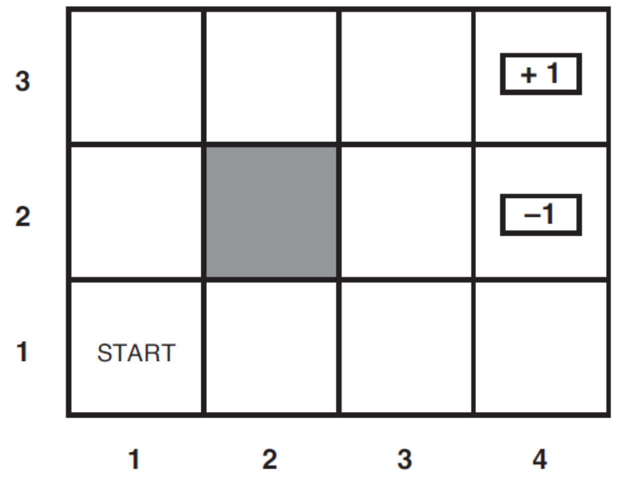
- Initial state: (1,1)
- Goal state: (4,1) utility +1
  (4,2) utility -1
- Actions:
  – Up, Down, Left, Right utility -0.04
  – *(won't move it running into the wall)*
- Transition model:
  – If RESULT(s, a) is **deterministic**, then it's a **search** problem

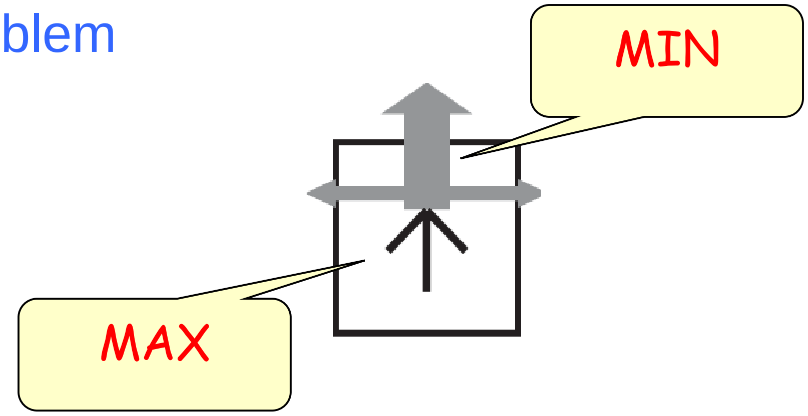


What's the **branching factor**?

Example problem (2)

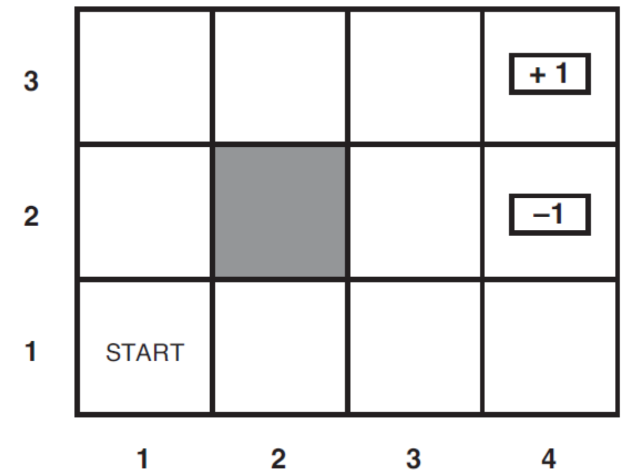
- Initial state: (1,1)
- Goal state: (4,1) utility +1
  (4,2) utility -1
- Actions:
  - Up, Down, Left, Right utility -0.04
  - *(won't move it running into the wall)*
- Transition model:
  - If RESULT(s, a) is **non-deterministic**, then it's an **adversarial search** problem

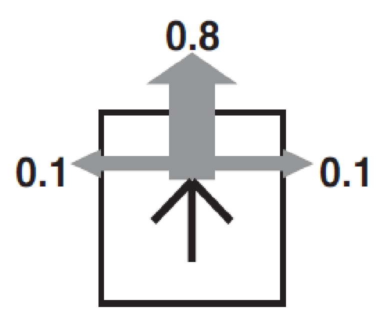




Example problem (3)

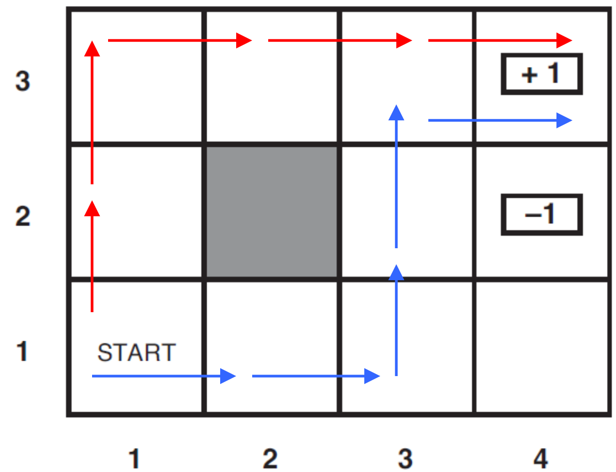
- Initial state: (1,1)
- Goal state: (4,1) utility +1
  (4,2) utility -1
- Actions:
  - Up, Down, Left, Right utility -0.04
  - *(won't move it running into the wall)*
- Transition model:
  - If RESULT(s, a) is **non-deterministic**, and **probabilistic**, then it's called MDP (Markov decision process)

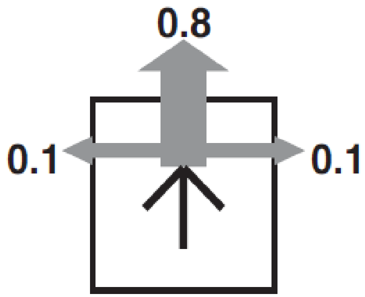




Example action sequence

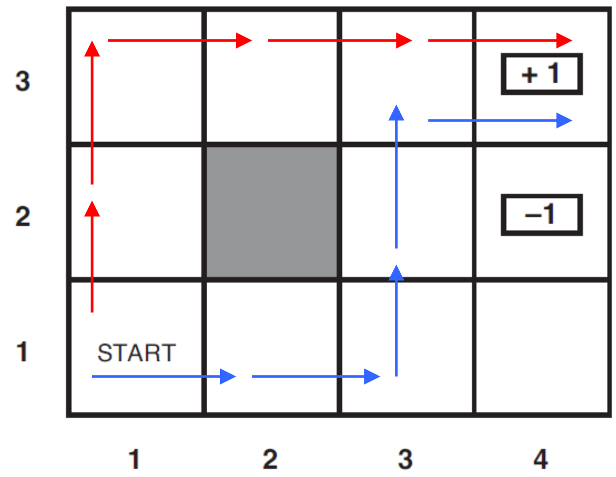
- Consider *[Up, Up, Right, Right, Right]*
- What’s the probability that it will reach the goal state (4,1)?
  - 0.8 * 0.8 * 0.8 * 0.8 * 0.8 = **0.32768**
- The other way for the given action sequence to reach (4,1)
  - 0.1 * 0.1 * 0.1 * 0.1 * 0.8 = **0.00008**

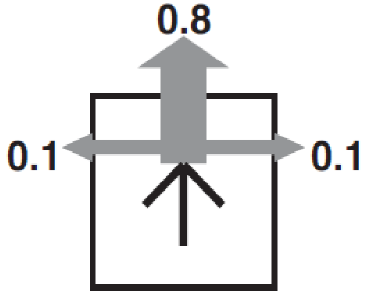




Example action sequence

- Consider *[Up, Up, Right, Right, Right]*
- What's the probability that it will reach the goal state (4,1)?
  - 0.8 * 0.8 * 0.8 * 0.8 * 0.8 = **0.32768**
  - *Reward: 4 * (-0.04) + 1 = 0.84*
- The other way for the given action sequence to reach (4,1)
  - 0.1 * 0.1 * 0.1 * 0.1 * 0.8 = **0.00008**
  - *Reward: 4 * (-0.04) + 1 = 0.84*

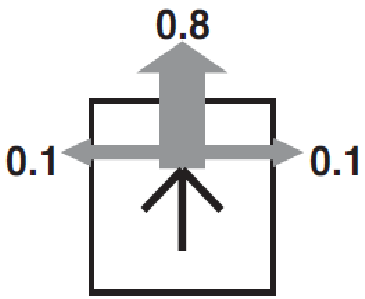




Markov decision process (MDP)

- MDP is a sequential decision problem that has
  - a **fully observable, stochastic** environment
  - a **Markovian** transition model, and
  - the **additive** rewards
- State space formalism
  - Initial state (s0)
  - Actions(s) = {a1, a2, … } for each state
  - Transition model P(s' | s, a) for each state and each action
  - Reward function R(s)

Solution to a MDP



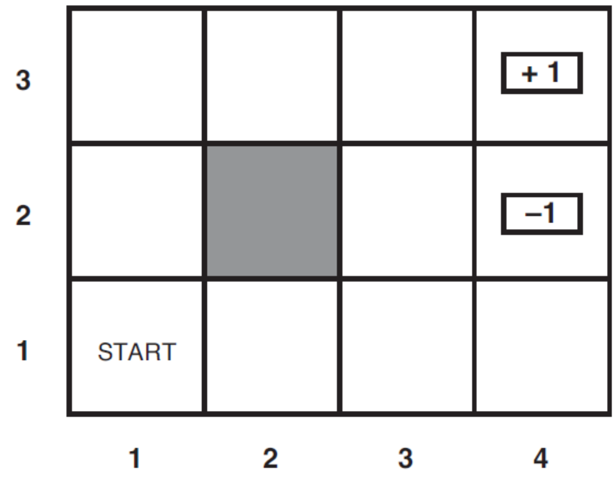
- It <u>should not</u> be any fixed action sequence
  - Example: *[Up, Up, Right, Right, Right]*
  - Because of non-deterministic outcomes of an action, there must be contingency plans

- It should be a **policy**, denoted *$\pi(s)$*, which specifies what the agent should do for any state *(s)* that it might reach

Question:

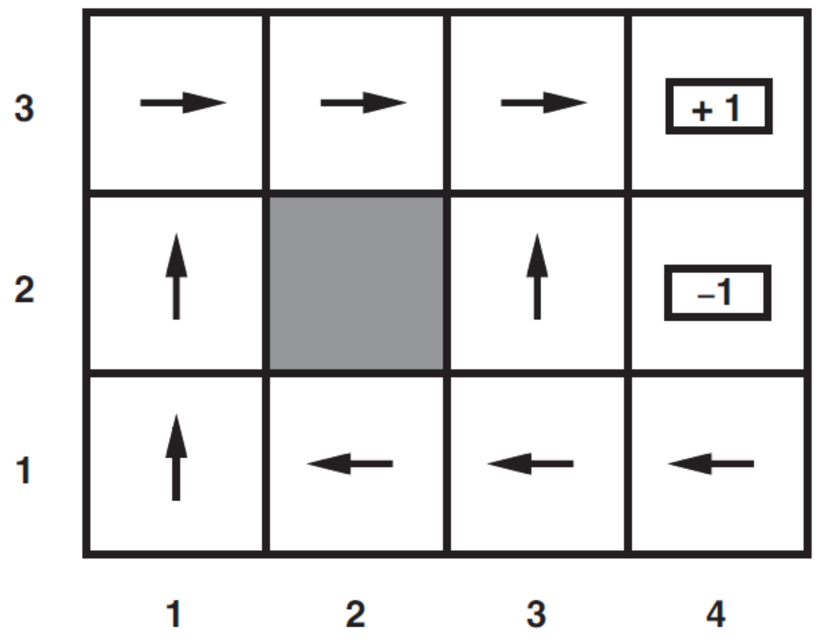
How many policies are there?

$4*4*...*4 = 4^9$



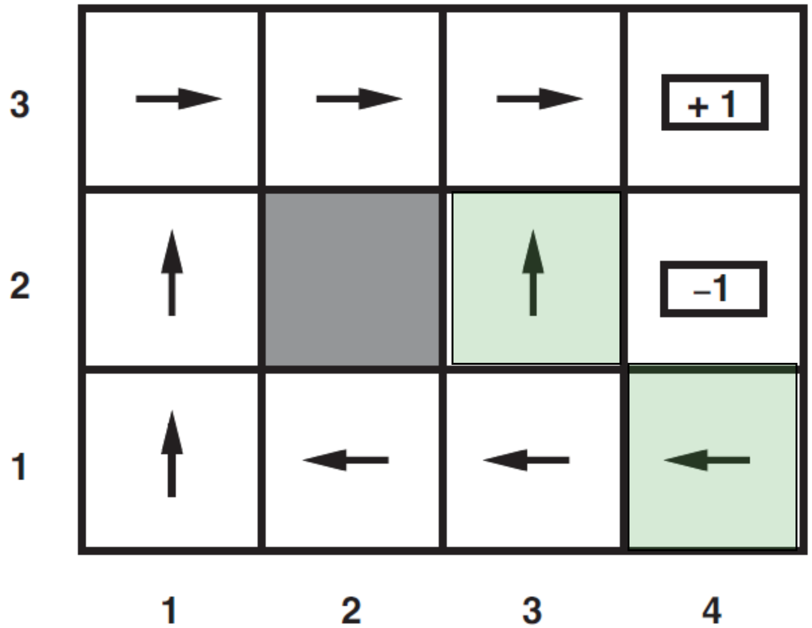
Optimal solution for *R(s) = -0.04*

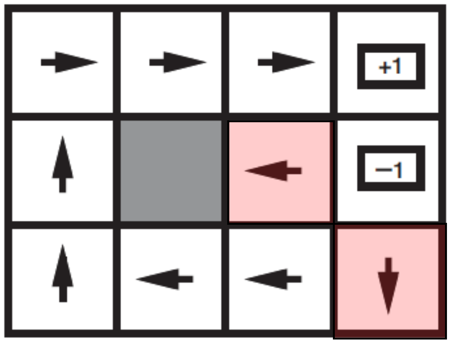
- When the cost of taking a step is fairly small compared to the penalty for ending up in (4,2) by accident
  - The policy recommends taking the long way round



Optimal solution for *-0.0221 ≤ R(s) ≤ 0*

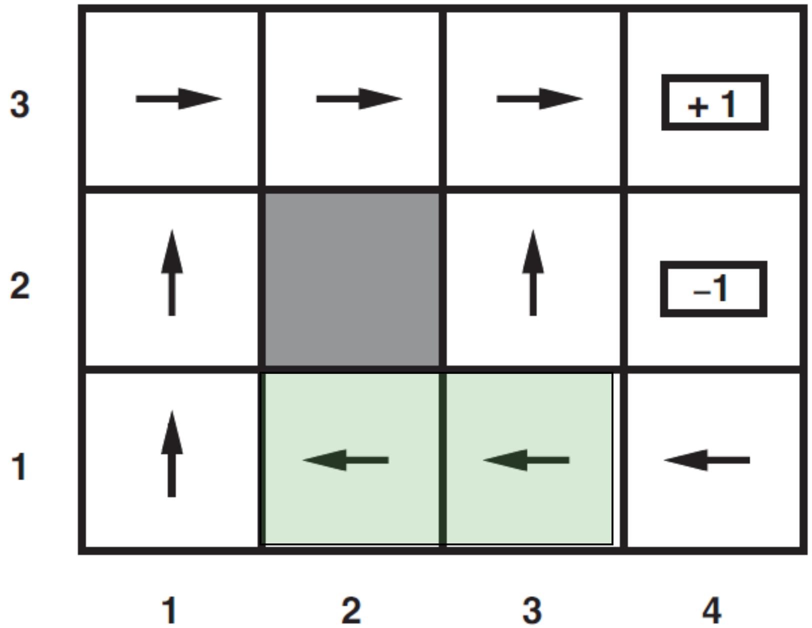
- When life is only slightly dreary…
  - Agent takes no risks at all

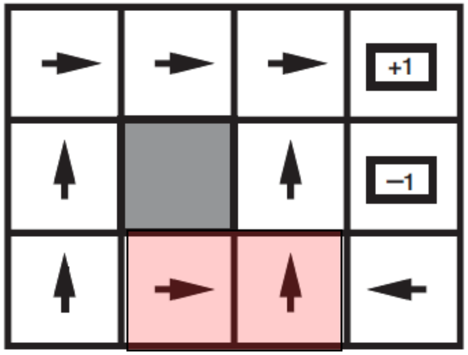




Optimal solution for *-0.4278 ≤ R(s) ≤ 0.0850*

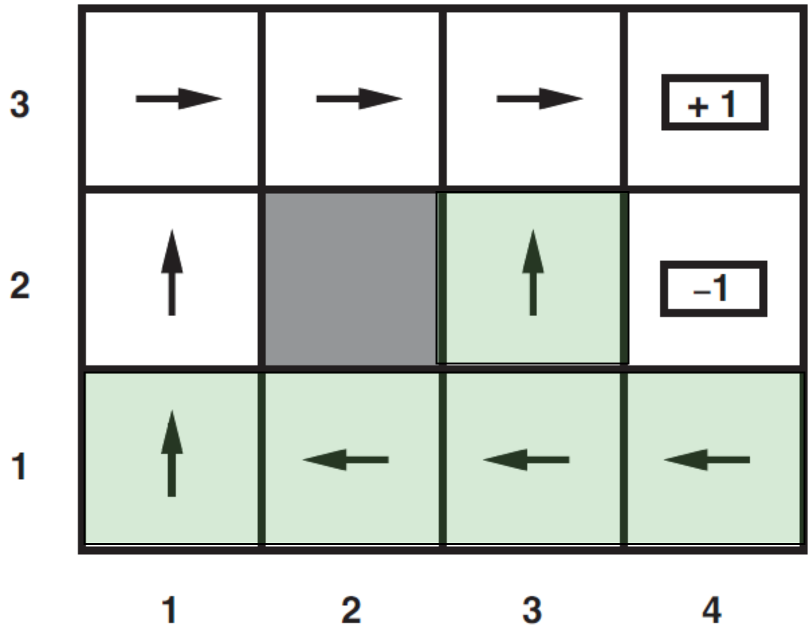
- When life is quite unpleasant..
  - Agent takes the shortest route to +1 state, and is willing to risk failing into the -1 state by accident

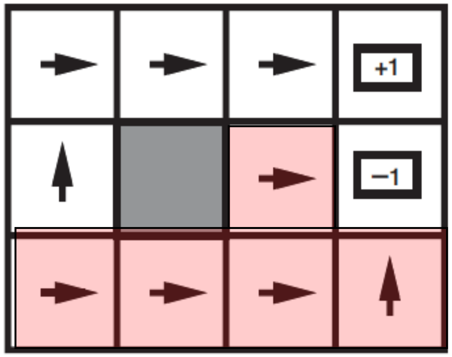




Optimal solution for *R(s) ≤ -1.6284*

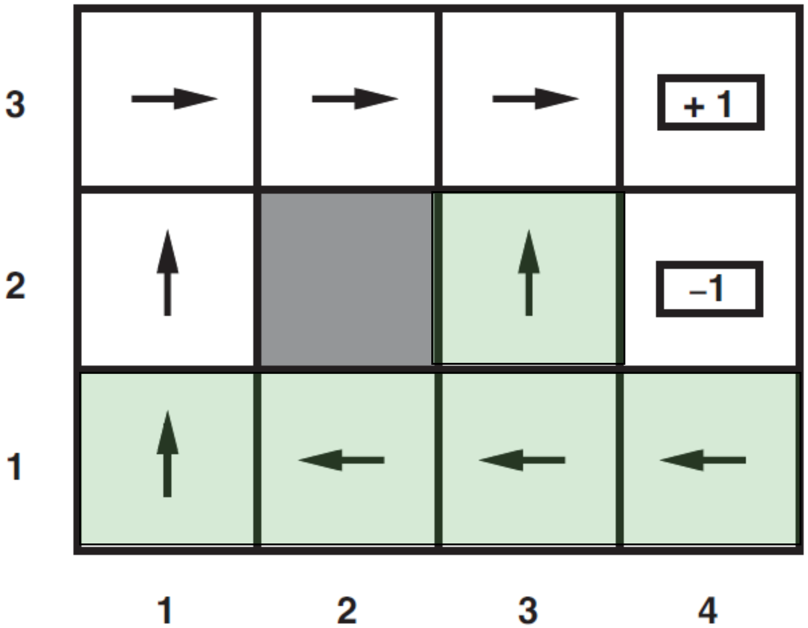
- Life is so painful…
  - Agent heads to the nearest exit, even if the exit is the -1 state

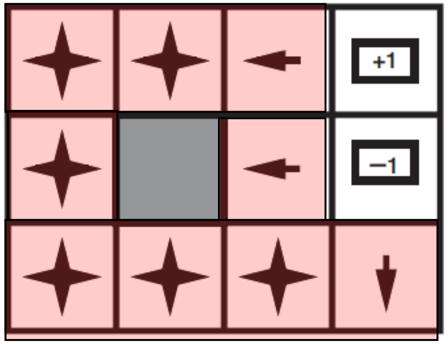




Optimal solution for *R(s) > 0*

- Life is so enjoyable…
  - Agent avoids both exit





Characteristic of MDPs

- Careful balancing of **risk** and **reward**, which is required by many real-world decision problems
  - AI
  - Operations research
  - Economics
  - Control theory

Utilities over time: *two options*

- Performance of agent is measured by a **sum** of rewards for the states visited
  - **Finite horizon**: a fixed time (N) after which nothing matters
    - This is actually complicated, since the optimal action in a given state could change over time
  - **Infinite horizon**: without a fixed time limit
    - This is simpler, since the optimal action depends only on the current state

We focus only on the "**infinite horizon**"

Sum of rewards: *two options*

- Additive rewards

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots$$

  - **Problem**: the sum is always infinite – hard to compare

- Discount rewards

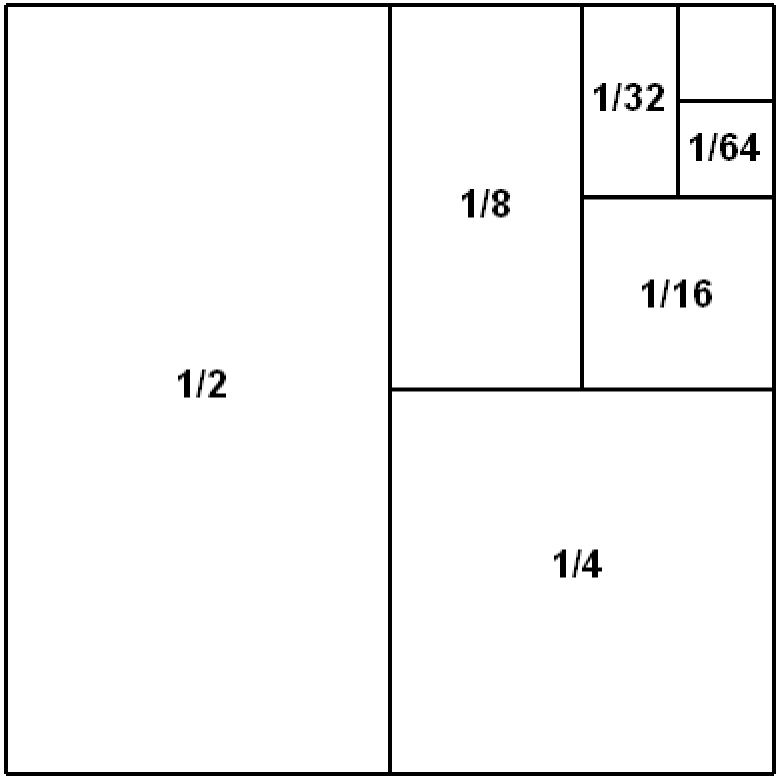
$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

  - Discount factor (γ) is a number between 0 and 1 *(e.g., γ=0.95)*
  - Equivalent to an interest rate of *(1/γ) – 1 = 0.0526*

  - We focus only on "**discount rewards**"

Recap: *Geometric series*

- Question: $1/2 + 1/4 + 1/8 + 1/16 + \cdots \;=\; ?$



Discount factor: example

America's Got Talent Winner is Not an Instant Millionaire

Last night, NBC's *America's Got Talent* announced the winner of its sixth season. Landau Eugene Murphy, Jr., a 36 year old car wash detailer from West Virginia, was overcome with emotion as he was told of the $1 million prize and the opportunity to headline a show at Caesar's Palace in Las Vegas.

. . .

But if you read the fine print on the screen at the end of the finale last night, the million dollar prize is actually a 40-year long annuity. In reality, Murphy, whose impressive singing voice resembles that of Frank Sinatra, can expect an annual payout of only $25,000—before taxes, that is.

$25,000 * 40 = $1,000,000

Source: Forbes, September 15, 2011

Discount factor: example

America's Got Talent Winner is Not an Instant Millionaire

Last night, NBC's *America's Got Talent* announced the winner of its sixth season. Landau Eugene Murphy, Jr., a 36 year old car wash detailer from West Virginia, was overcome with emotion as he was told of the $1 million prize and the opportunity to headline a show at Caesar's Palace in Las Vegas.

...

But if you read the fine print on the screen at the end of the finale last night, the million dollar prize is actually a 40-year long annuity. In reality, Murphy, whose impressive singing voice resembles that of Frank Sinatra, can expect an annual payout of only $25,000—before taxes, that is. Murphy will be offered a lump cash payment in lieu of the annuity, but this will likely be in the $300,000 range (again, before taxes).

$25,000 * 40 = $1,000,000

Source: Forbes, September 15, 2011

Which one is better?

Discount factor: simplified example

- A similar example with fewer payouts (to fit on the slide):

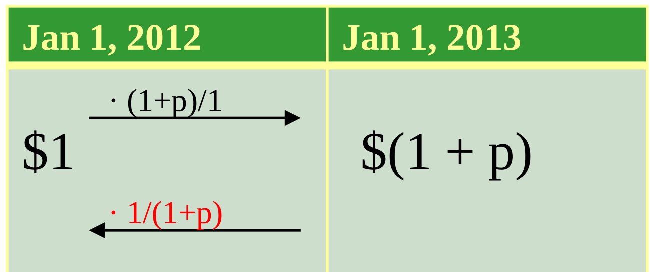
Jan 1, 2012	Jan 1, 2013	Jan 1, 2014	Jan 1, 2015
$25,000	$25,000	$25,000	$25,000

the interest rate is 5% per year

- Equivalent to a lump sum of $93,081
  - **How do we know this?**

Discount factor: simplified example

- If we put $1 into a savings account with **interest rate** p, after one year, we will have $(1 + p) in the account



- The $(1 + p) after one year is equivalent to $1 now

- We call 0 < 1/(1+p) ≤ 1 the **discount factor** γ (gamma).

Discount factor: simplified example

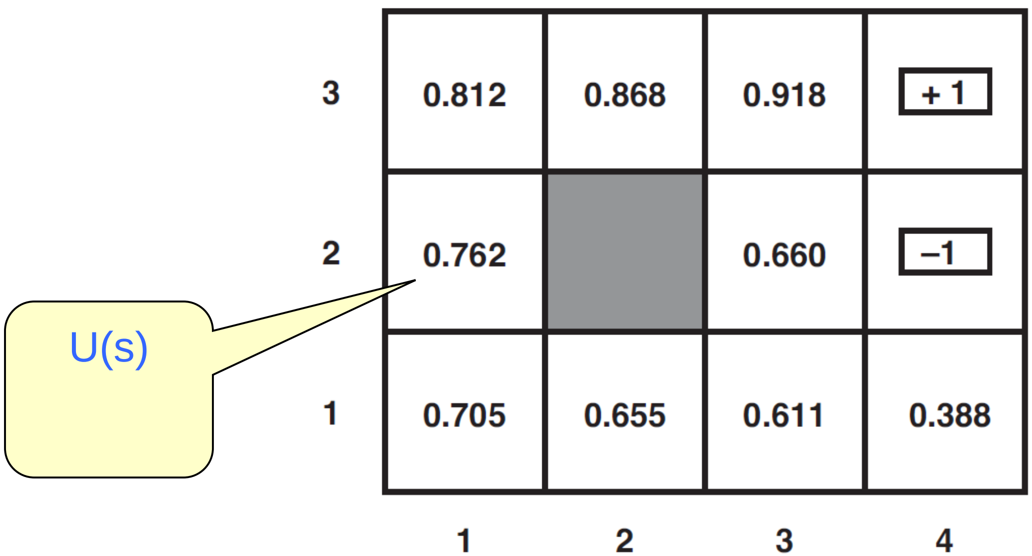
- For an interest rate of 5% (i.e. discount factor of $\gamma \approx 0.952$), what is the lump-sum payoff?

Jan 1, 2012	Jan 1, 2013	Jan 1, 2014	Jan 1, 2015
$25,000	$25,000	$25,000	$25,000

- $(1+ \gamma+ \gamma^2+\gamma^3) * \$25,000 \approx \$$**93,081**.20
  - The amount in 2015
    - $\$93,081.20 * (1.05)^3 = \$107,753$
    - $(1.05^3 + 1.05^2 + 1.05 + 1) * \$25,000 = \$107,753$

Difference between R(s) and U(s)

- ***R(s)*** – the "**short-term**" reward for being in state (s)
- ***U(s)*** – the "**long-term**" **total** reward from (s) onward



$$R(s) = \begin{cases} +1 \\ -0.04 \\ -1 \end{cases}$$

Problem – *need to compute both of them*

- Expected utility of executing policy (*π*) starting in state (*s*)

$$\boxed{U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R(S_t)\right]}$$

- Out of all policies (*π*) that the agent could choose, the one with the highest utilities is

$$\boxed{\pi_s^* = \underset{\pi}{\operatorname{argmax}}\, U^{\pi}(s)}$$

Do we have to enumerate all policies?

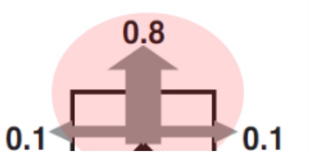
Outline of today's lecture

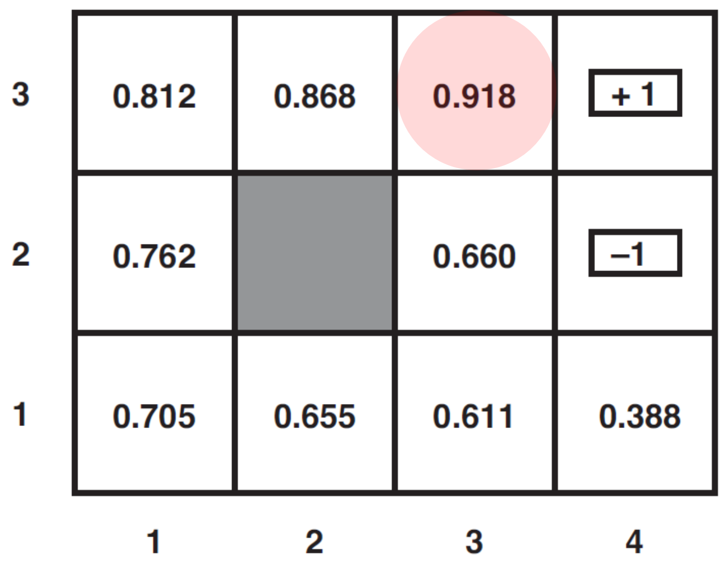
- Sequential Decision (MDP)
  - A sequence of decisions (versus *one-shot* decision)
- **Value Iteration**
  - Algorithm for computing optimal policy for MDP
- Policy Iteration
  - An alternative algorithm

Given U(s), compute optimal policy π*(s)

- Pick the action with the maximal expected utility (MEU)

$$\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$





Up:

Down:

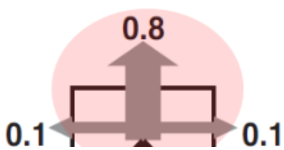
Left:

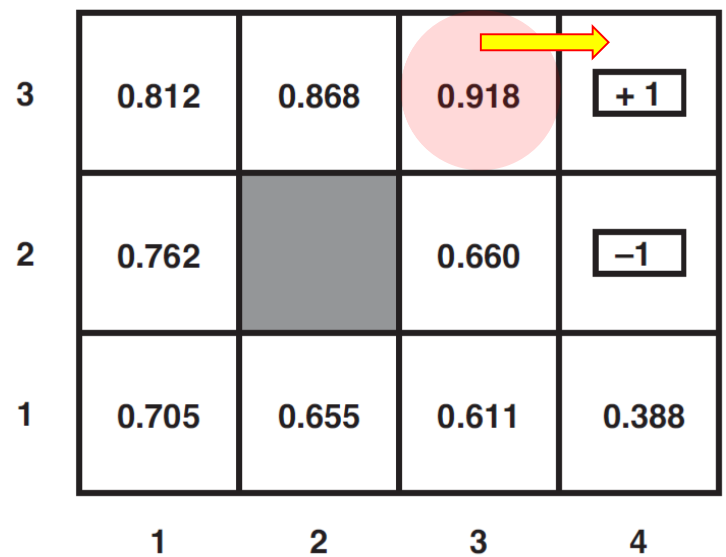
Right:

Given U(s), compute optimal policy π*(s)

- Pick the action with the maximal expected utility (MEU)

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) U(s')$$





Up:

$0.8*0.918 + 0.1*0.868 + 0.1*1 = 0.9212$

Down:

$0.8*0.660 + 0.1*0.868 + 0.1*1 = 0.7148$

Left:

$0.8*0.868 + 0.1*0.660 + 0.1*0.918 = 0.8522$

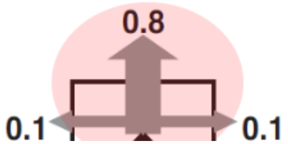
Right:

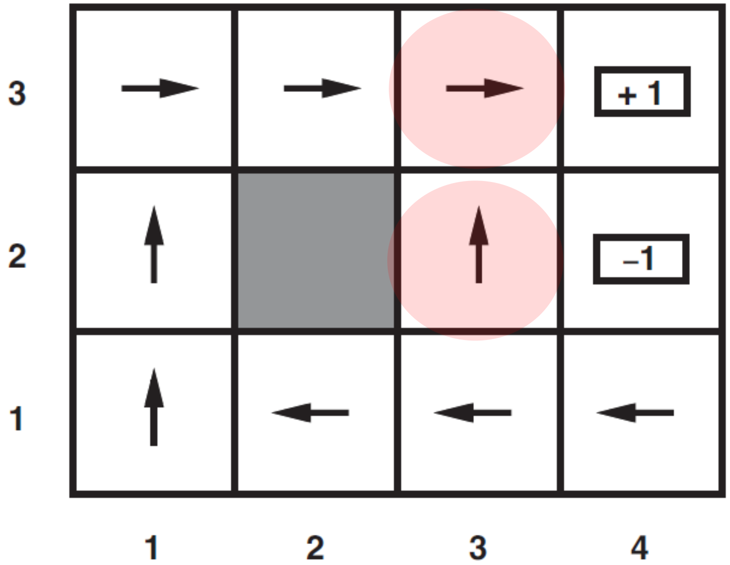
$0.8*1 + 0.1*0.918 + 0.1*0.660 = \boxed{0.9578}$

Given optimal policy π*(s), compute U(s)

- State utility is maximized when following the optimal policy

$$\pi^*_s = \operatorname*{argmax}_{\pi} U^{\pi}(s)$$





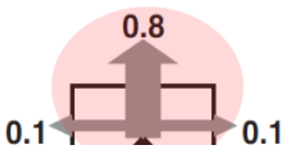
(3,3) : Right

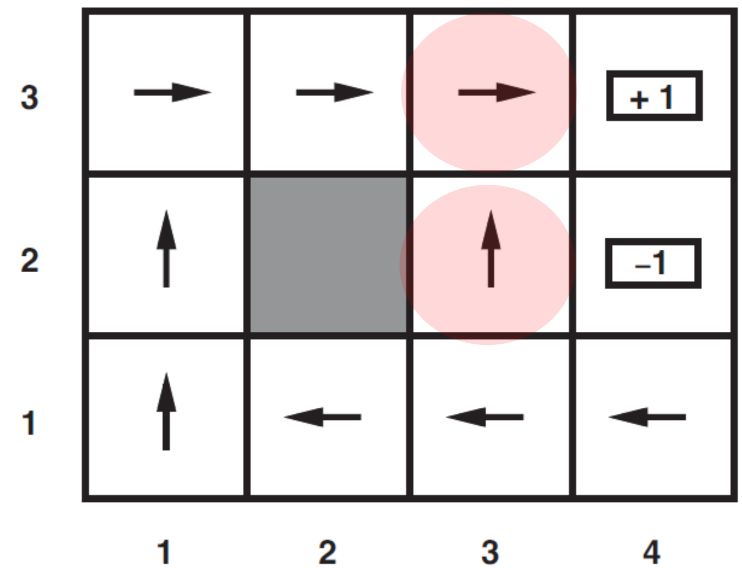
(3,2) : Up

Given optimal policy π*(s), compute U(s)

- State utility is maximized when following the optimal policy

$$\pi_s^* = \operatorname*{argmax}_{\pi} U^{\pi}(s)$$





(3,3) : Right

$0.8*1 + 0.1*U(S_{3,3}) + 0.1*U(S_{3,2}) - 0.04 = U(S_{3,3})$

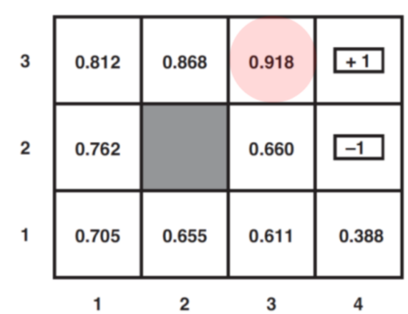
(3,2) : Up

$0.8*U(S_{3,3}) + 0.1*U(S_{3,2}) + 0.1*(-1) - 0.04 = U(S_{3,2})$

$0.76 + 0.1*U(S_{3,2}) = 0.9*U(S_{3,3})$

$0.8*U(S_{3,3}) - 0.14 = 0.9*U(S_{3,2})$

$U(S_{3,3}) = 0.9179$



A “chicken and egg” situation

- Which comes first?
  - Given U(s), we can easily compute the optimal policy π*(s)
  - Given the optimal policy π*(s), we can easily compute U(s)

3	→	→	→	+1
2	↑		↑	–1
1	↑	←	←	←
	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
	1	2	3	4

It’s the maximum number among all policies - enumerate all policies to get it?

Basic idea for solving MDP

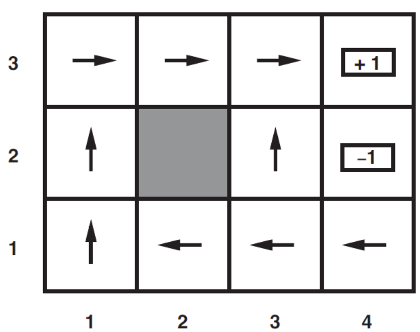
- (1) Calculate the utility of each state
- (2) Use state utilities to select optimal action in each state

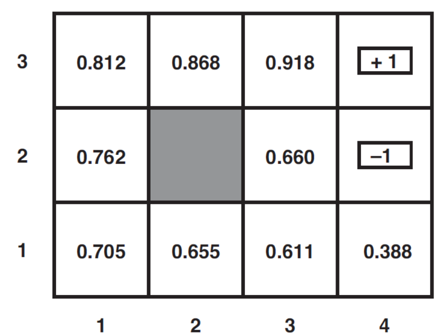
Richard Bellman
USC professor
(from 1965 to 1984)

Basic idea for solving MDP

- (1) Calculate the utility of each state
- (2) Use state utilities to select optimal action in each state

Richard Bellman
USC professor
(from 1965 to 1984)



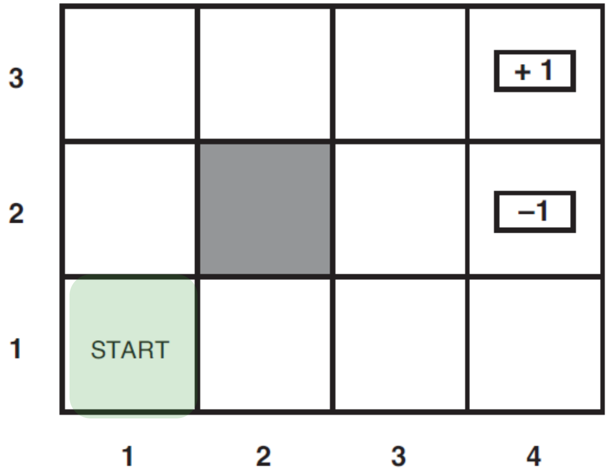


Bellman equation (1957)

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

Example:

$$\begin{aligned} U(1,1) = -0.04 + \gamma \max[\ & 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), && (Up) \\ & 0.9U(1,1) + 0.1U(1,2), && (Left) \\ & 0.9U(1,1) + 0.1U(2,1), && (Down) \\ & 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\ ]. && (Right) \end{aligned}$$



Similarly, write down U(1,2), U(1,3), U(1,4),
U(2,1), U(2,2), U(2,3), U(2,4),
U(3,1), U(3,2), U(3,3), U(3,4)

Solutions to these equations are unique!

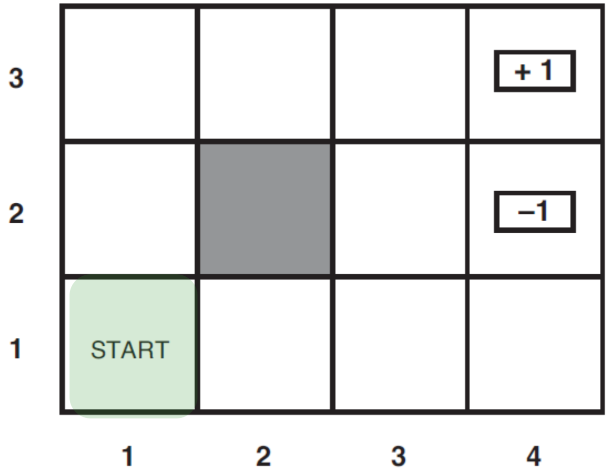
Bellman equation (1957)

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

You can't directly solve these equations due to the non-linear (**max**) operation

Example:

$$\begin{aligned} U(1,1) = -0.04 + \gamma \max[\ & 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), && (Up) \\ & 0.9U(1,1) + 0.1U(1,2), && (Left) \\ & 0.9U(1,1) + 0.1U(2,1), && (Down) \\ & 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\ ]. && (Right) \end{aligned}$$



Similarly, write down U(1,2), U(1,3), U(1,4),
U(2,1), U(2,2), U(2,3), U(2,4),
U(3,1), U(3,2), U(3,3), U(3,4)

Solutions to these equations are unique!

Iterative approach

- Start with **arbitrary initial values** for U(s), compute right-hand side of the equation, and plug it into left-hand side
  - Updating U(s) based on U(s'), the utilities of neighbor states (s')

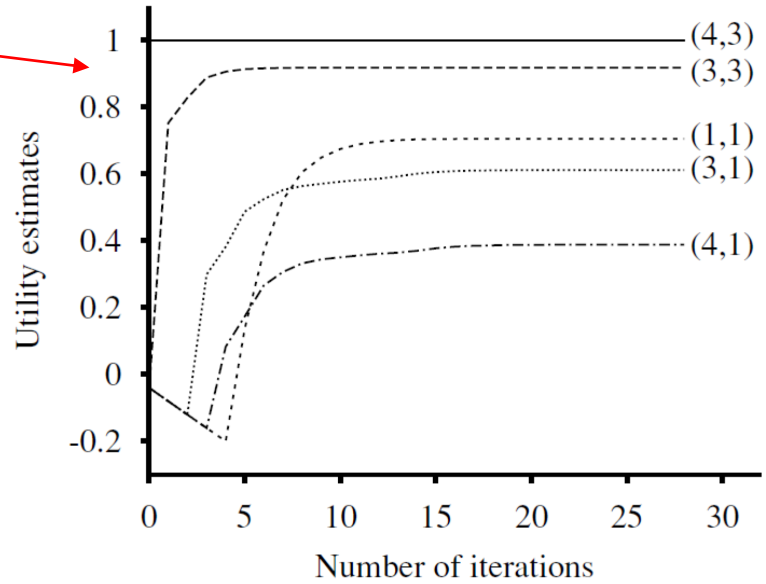
$$\boxed{U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s')}$$

Iterative approach: *how does it work in practice?*

- Start with **arbitrary initial values** for U(s), compute right-hand side of the equation, and plug it into left-hand side
  - Updating U(s) based on U(s'), the utilities of neighbor states (s')

$$\boxed{U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s')}$$

U($S_{3,3}$) = 0.9179



Value iteration algorithm

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Value iteration algorithm

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    $\boxed{\epsilon\text{, the maximum error allowed in the utility of any state}}$
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Wikipedia: geometric progression

$$\sum_{k=m}^{\infty} ar^k = \frac{ar^m}{1-r}$$

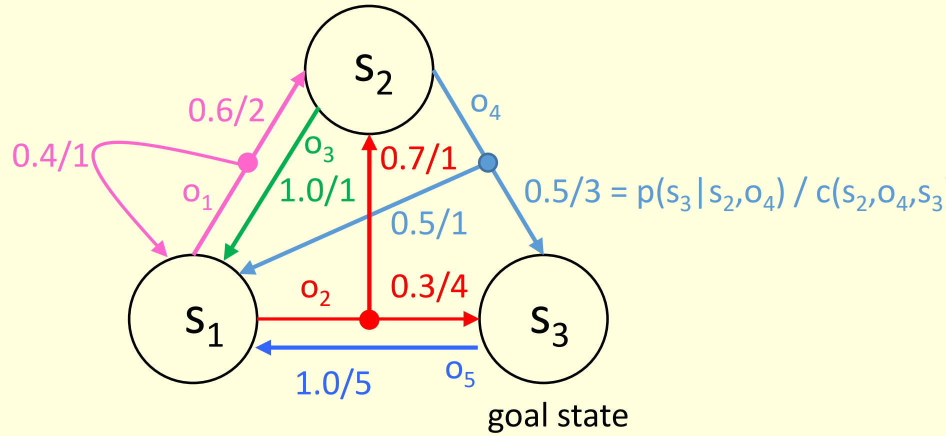
MDP solving example

```
#include <stdio.h>
#define min(x,y) ((x)<(y))?(x):(y);

main()
{
  float gamma=0.95;
  float s1=0.0, s2=0.0, s3=0.0;
  float o1,o2,o3,o4,o5;
  char *ostar1, *ostar2, *ostar3;
  int i;

  for (i=1; i<10000; ++i)
    {
      o1 = 0.4*(1.0+gamma*s1)+0.6*(2.0+gamma*s2);
      o2 = 0.7*(1.0+gamma*s2)+0.3*(4.0+gamma*s3);
      o3 = 1.0*(1.0+gamma*s1);
      o4 = 0.5*(1.0+gamma*s1)+0.5*(3.0+gamma*s3);
      o5 = 1.0*(5.0+gamma*s1);
      if (o1<o2)
        {
          s1 = o1;
          ostar1 = "o1";
        }
      else
        {
          s1 = o2;
          ostar1 = "o2";
        }
      if (o3<o4)
        {
          s2 = o3;
          ostar2 = "o3";
        }
      else
        {
          s2 = o4;
          ostar2 = "o4";
        }
      s3 = o5;
      ostar3 = "o5";
      printf("iteration %5d: s1 (execute %s) %5.2f (o1 %5.2f, o2 %5.2f), s2 (execute %s) %5.2f (o3 %5.2f, o4 %5.2f), s3 (execute %s) %5.2f\n",
             i, ostar1, s1, o1, o2, ostar2, s2, o3, o4, ostar3, s3);
    }
}
```

γ=0.95



MDP solving result

γ=0.95

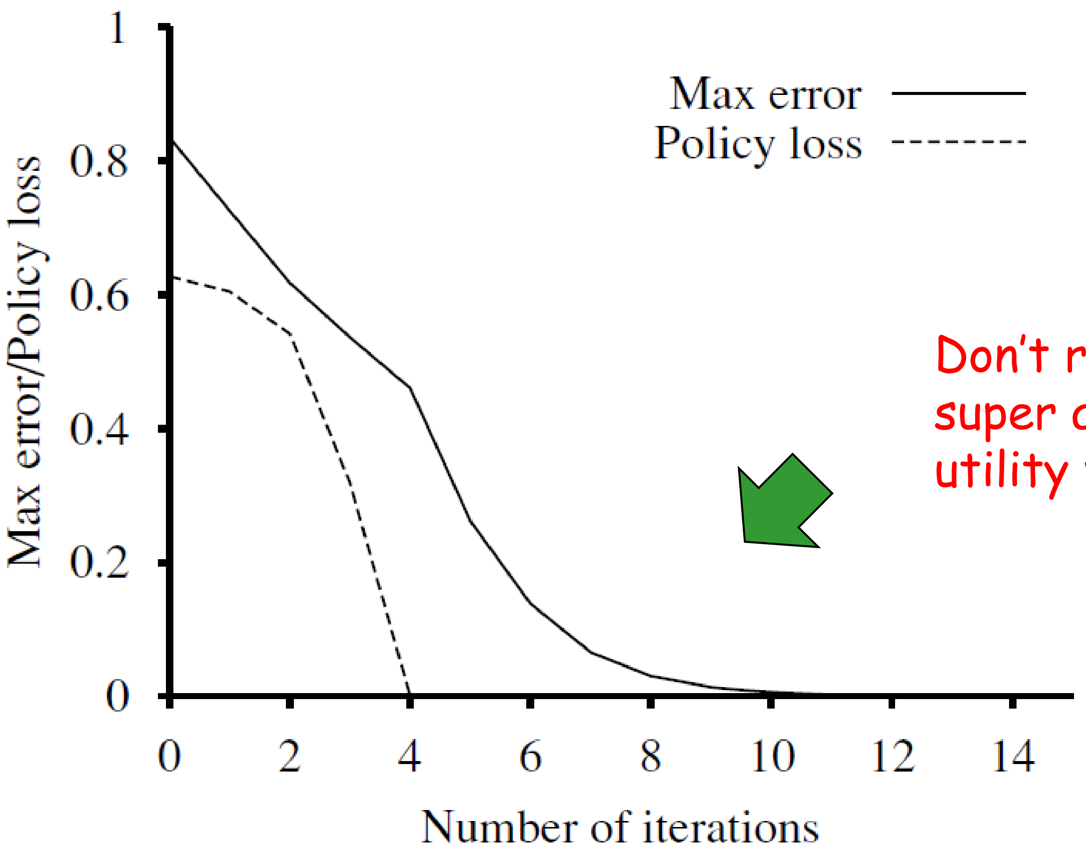
$c_{s_1,iteration}$ $c_{s_2,iteration}$ $c_{s_3,iteration}$

```
iteration    1: s1 (execute o1)  1.60 (o1  1.60, o2  1.90), s2 (execute o3)  1.00 (o3  1.00, o4  2.00), s3 (execute o5)  5.00
iteration    2: s1 (execute o1)  2.78 (o1  2.78, o2  3.99), s2 (execute o3)  2.52 (o3  2.52, o4  5.14), s3 (execute o5)  6.52
iteration    3: s1 (execute o1)  4.09 (o1  4.09, o2  5.43), s2 (execute o3)  3.64 (o3  3.64, o4  6.42), s3 (execute o5)  7.64
iteration    4: s1 (execute o1)  5.23 (o1  5.23, o2  6.50), s2 (execute o3)  4.89 (o3  4.89, o4  7.57), s3 (execute o5)  8.89
iteration    5: s1 (execute o1)  6.37 (o1  6.37, o2  7.68), s2 (execute o3)  5.97 (o3  5.97, o4  8.71), s3 (execute o5)  9.97
iteration    6: s1 (execute o1)  7.42 (o1  7.42, o2  8.71), s2 (execute o3)  7.05 (o3  7.05, o4  9.76), s3 (execute o5) 11.05
iteration    7: s1 (execute o1)  8.44 (o1  8.44, o2  9.74), s2 (execute o3)  8.05 (o3  8.05, o4 10.78), s3 (execute o5) 12.05
iteration    8: s1 (execute o1)  9.40 (o1  9.40, o2 10.69), s2 (execute o3)  9.02 (o3  9.02, o4 11.73), s3 (execute o5) 13.02
iteration    9: s1 (execute o1) 10.31 (o1 10.31, o2 11.61), s2 (execute o3)  9.93 (o3  9.93, o4 12.65), s3 (execute o5) 13.93
iteration   10: s1 (execute o1) 11.18 (o1 11.18, o2 12.47), s2 (execute o3) 10.80 (o3 10.80, o4 13.51), s3 (execute o5) 14.80
...
iteration 9995: s1 (execute o1) 27.64 (o1 27.64, o2 28.94), s2 (execute o3) 27.26 (o3 27.26, o4 29.98), s3 (execute o5) 31.26
iteration 9996: s1 (execute o1) 27.64 (o1 27.64, o2 28.94), s2 (execute o3) 27.26 (o3 27.26, o4 29.98), s3 (execute o5) 31.26
iteration 9997: s1 (execute o1) 27.64 (o1 27.64, o2 28.94), s2 (execute o3) 27.26 (o3 27.26, o4 29.98), s3 (execute o5) 31.26
iteration 9998: s1 (execute o1) 27.64 (o1 27.64, o2 28.94), s2 (execute o3) 27.26 (o3 27.26, o4 29.98), s3 (execute o5) 31.26
iteration 9999: s1 (execute o1) 27.64 (o1 27.64, o2 28.94), s2 (execute o3) 27.26 (o3 27.26, o4 29.98), s3 (execute o5) 31.26
```

- If one can stop after executing a handful of iterations, then
  - execute o_1 in s_1, o_3 in s_2 and o_5 in s_3 (see iteration 9999)
  - …
  - execute o_1 in s_1, o_3 in s_2 and o_5 in s_3 (see iteration 2)
  - execute o_1 in s_1, o_3 in s_2 and o_5 in s_3 (see iteration 1)

Policy loss: *due to $U_i(s) - U(s)$*

- Difference between
  - The MEU policy based on $U_i(s)$ of the *i-th* value iteration
  - The MEU policy based on the actual (perfect) utility $U(s)$



Don't really need super accurate utility values…

Outline of today's lecture

- Sequential Decision (MDP)
  - A sequence of decisions (versus *one-shot* decision)
- Value Iteration
  - Algorithm for computing optimal policy for MDP
- **Policy Iteration**
  - An alternative algorithm

Alternative approach to solving MDP

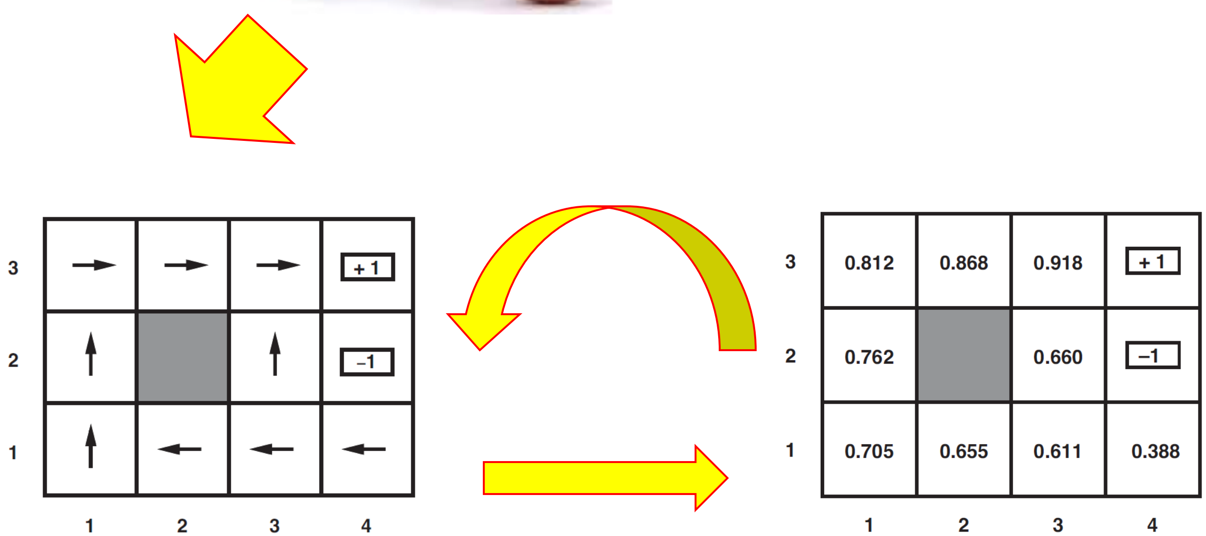
- (1) **Policy evaluation**: given a policy ***π(s)***, compute ***U(s)***
- (2) **Policy improvement**: compute a new policy

$$\boxed{\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')}$$

Alternative approach to solving MDP

- (1) **Policy evaluation**: given a policy ***π(s)***, compute ***U(s)***
- (2) **Policy improvement**: compute a new policy

$$\pi^*(s) = \operatorname*{argmax}_{a\in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$



Policy evaluation

- Given a policy *π(s)*, compute *U(s)*

$$\boxed{U_i(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s)) U_i(s')}$$

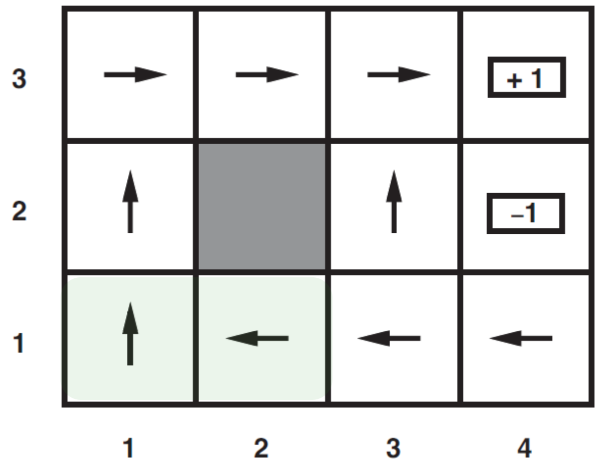
This is a set of *linear* equations, which is polynomial time solvable (by LP solver); in contrast, the Bellman equation $U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$ is nonlinear.

Policy evaluation: *example*

- Given a policy ***π(s)***, compute ***U(s)***

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s)) U_i(s')$$

This is a set of ***linear*** equations, which is polynomial time solvable (by LP solver); in contrast, the Bellman equation $U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$ is nonlinear.



$$\begin{aligned} U_i(1,1) &= -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1)\ , \\ U_i(1,2) &= -0.04 + 0.8U_i(1,3) + 0.2U_i(1,2)\ , \\ &\vdots \end{aligned}$$

Policy improvement: *Given U(s), compute policy π*(s)*

- Pick the action with the maximal expected utility (MEU)

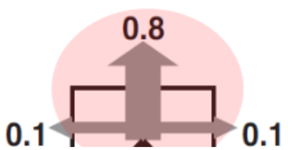
$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) U(s')$$

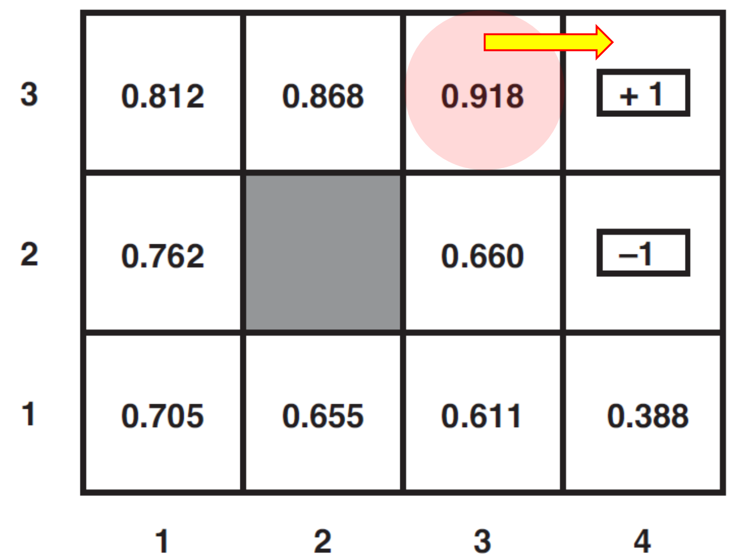
- We already talked about this!

Policy improvement: *Given U(s), compute policy π*(s)*

- Pick the action with the maximal expected utility (MEU)

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) U(s')$$





Up:

$0.8*0.918 + 0.1*0.868 + 0.1*1 = 0.9212$

Down:

$0.8*0.660 + 0.1*0.868 + 0.1*1 = 0.7148$

Left:

$0.8*0.868 + 0.1*0.660 + 0.1*0.918 = 0.8522$

Right:

$0.8*1 + 0.1*0.918 + 0.1*0.660 = \boxed{0.9578}$

Overall algorithm

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

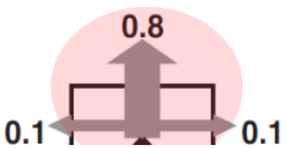
Summary of today's lecture

- Sequential Decision (MDP)
  - A sequence of decisions (versus *one-shot* decision)
- Value Iteration
  - Algorithm for computing optimal policy for MDP
- Policy Iteration
  - An alternative algorithm

Quiz 9

- Pick the action with the maximal expected utility (MEU)

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) U(s')$$



Up:

0.8*(???)+ 0.1*{???) + 0.1*(???) = ???

Down:

0.8*(???)+ 0.1*{???) + 0.1*(???) = ???

Left:

0.8*(???)+ 0.1*{???) + 0.1*(???) = ???

Right:

0.8*(???)+ 0.1*{???) + 0.1*(???) = ???

	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