Markov Decision Process (MDP)

KUO-SHIH TSENG (曾國師)
DEPARTMENT OF MATHEMATICS
NATIONAL CENTRAL UNIVERSITY, TAIWAN

2021/04/07

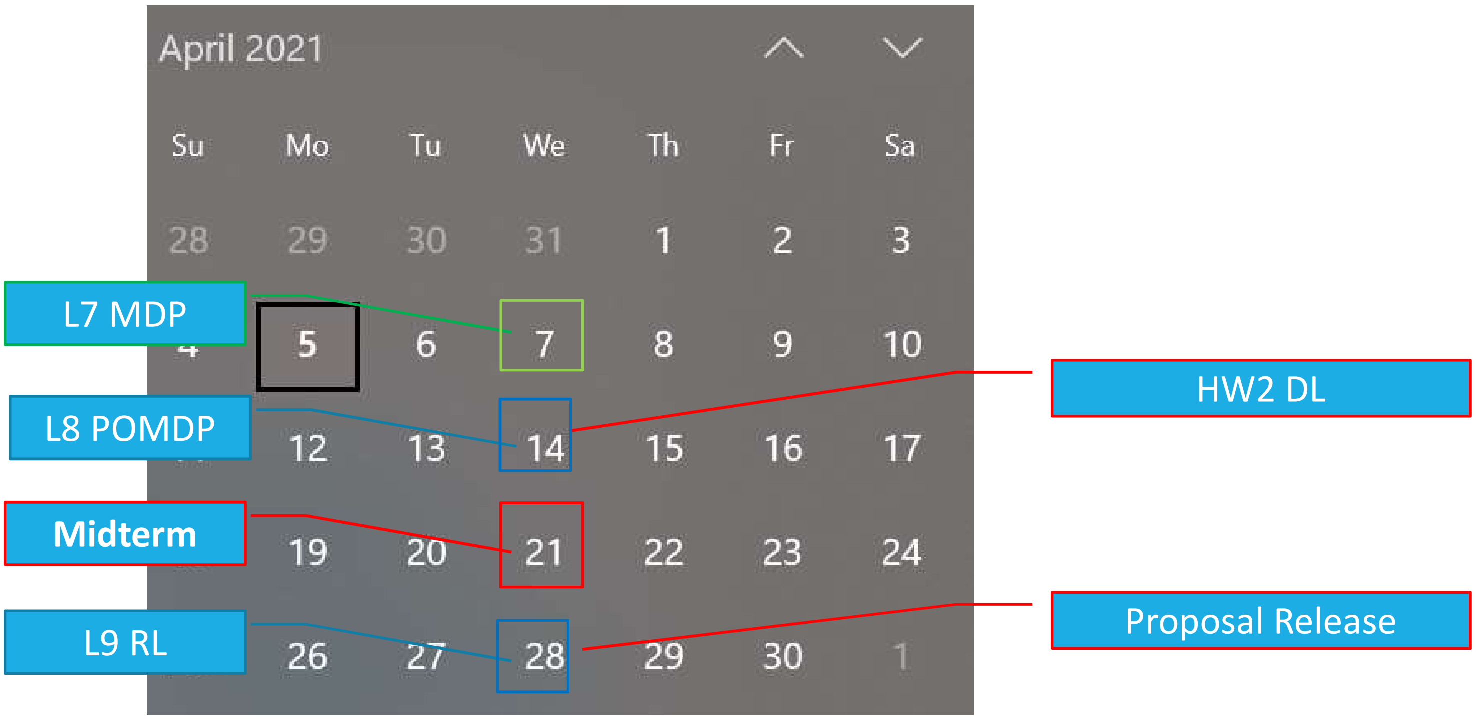
Course Announcement

- Work on your HW2 ASAP. The deadline is 4/14(Wed).
  - Bayesian inference (40%)
  - **MDP solver (40%)**
  - **A MDP problem (20%)**
- You can discuss course materials with me and classmates. You **should** work on your HW *independently*.
- NOTICE: I will NOT do curve fitting (e.g., "sqrt(X)*6" for your score) for your scores.
- **Late policy: If your HW is late for 1 day, the discount rate is 0.8. For 2 days, the discount rate is 0.8^2. and so on.**
- Remember: life is a real-time model! Not a offline search!

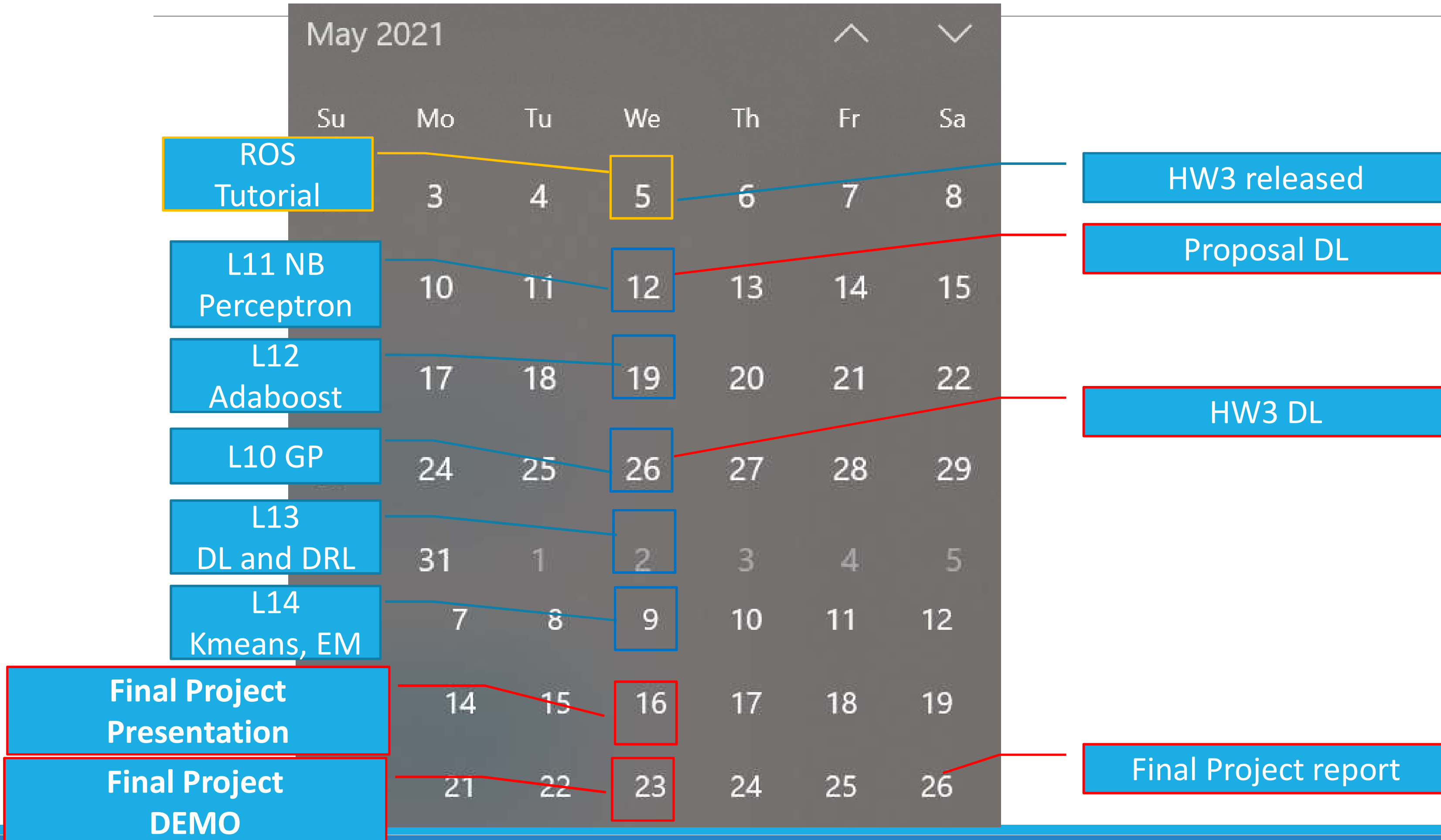
Course Announcement

- ***Midterm (04/21/2020), 3-5pm, in M430***
  - Given a real world problem.
  - Design a perception and decision-making system for this problem using MDP , MCTS and Bayesian approaches.
- You can take one A4-size cheating sheet.
- You cannot use any electrical devices (e.g., Notebook or mobilephone), which can access to internet.
- You don’t need calculators.
- You can find the **midterm_sample.pdf** on the eeclass.

Course Announcement



Course Announcement



Outline

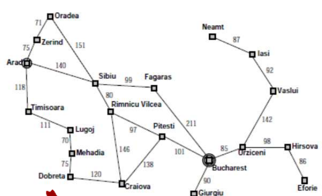
- MDP
- Recitation: LRTA*
- Bellman equation
- Value iteration
- Policy iteration

Outline

[Problem solving]

Search problems

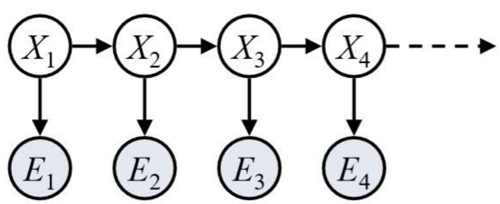
Adversarial Search



[Perception and Uncertainty]

Bayes Theorem

Bayes Filter and Smoothing

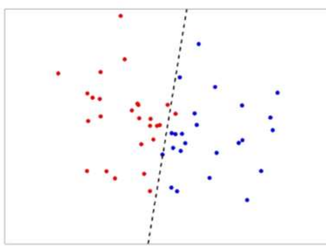


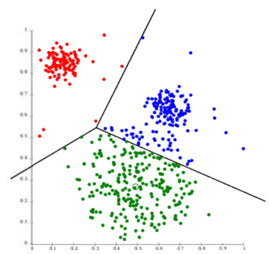
[Learning and Decision-making]

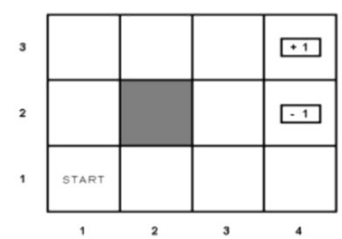
Supervised learning

Unsupervised learning

Reinforcement learning







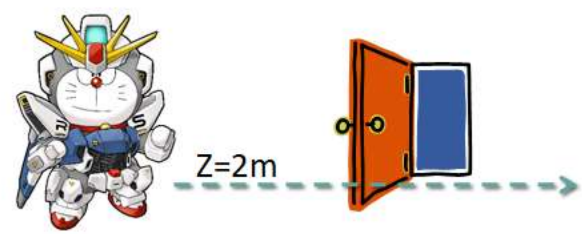
MDP

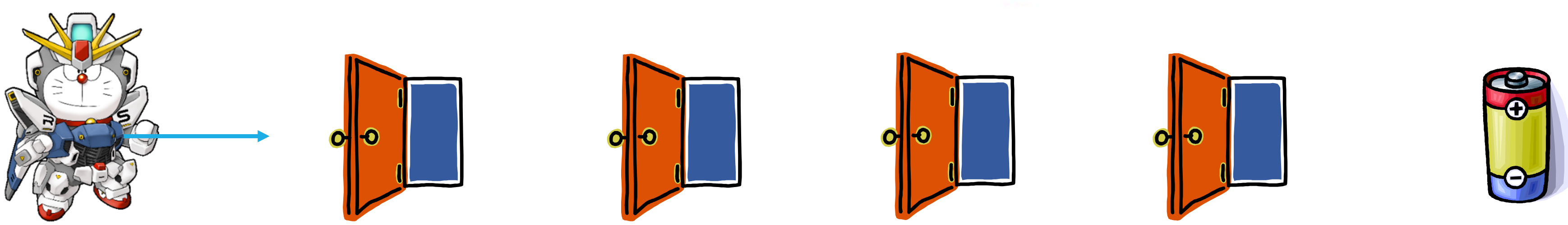
- Why do humans buy lotteries?
- Why do humans keep doing something with low probability?
- Humans made decisions based on
  - Probability
  - Utility
  - **Future**

MDP

- Bayes decision only considers one-step expectation. For most of applications, the robot needs to do ***sequential*** optimal decisions for achieving the goal.
- In LRTA*, the transition/motion model is deterministic. Let's further consider a probabilistic transition/motion model for finding ***sequential*** optimal decisions.

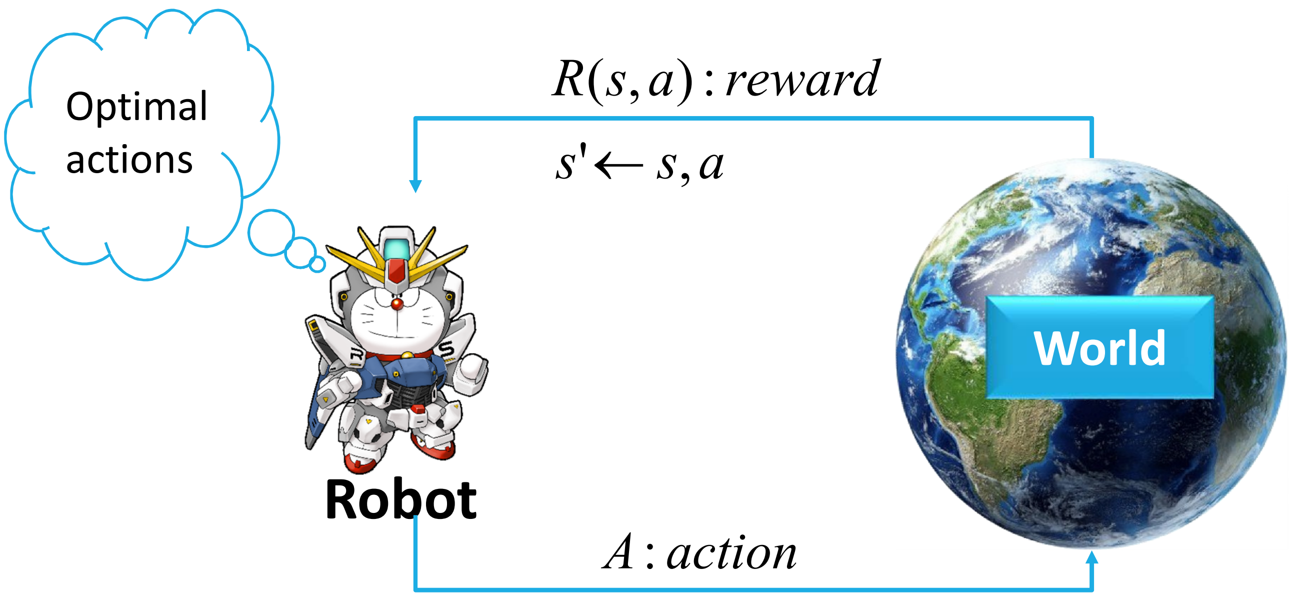
$$\frac{p(open \mid z_2)}{p(close \mid z_2)} > \frac{(R_{CC} - R_{OC})}{(R_{OO} - R_{CO})} \Rightarrow Decision : Move!$$





MDP

- MDP is a model for finding sequential optimal decisions.
  - State: fully observable
  - State transition: stochastic **(Motion model)**



MDP

- MDP is a model for finding sequential optimal decisions.
  - State: fully observable
  - State transition: stochastic **(Motion model)**

$[GIVEN]$

$S : state$

$A : action$

$P(s'|\, s, a) : Transition\ \ probability$

$R(s, a) : reward$

$\gamma : discount$

$s :$ state in t

$s' :$ state in t + 1

$[Find]$

$\pi^* = \arg\max U^{\pi}(s)$

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

$\pi : policy$

$\pi^* : optimal\ \ policy$

$U : utility$

MDP

- The M.S. life

$[GIVEN]$

$S: state \Rightarrow 4 \text{ states}$

$A: action \Rightarrow a \in \{\text{work hard, lazy}\}$

$P(s'|s,a): Transition\ probability$

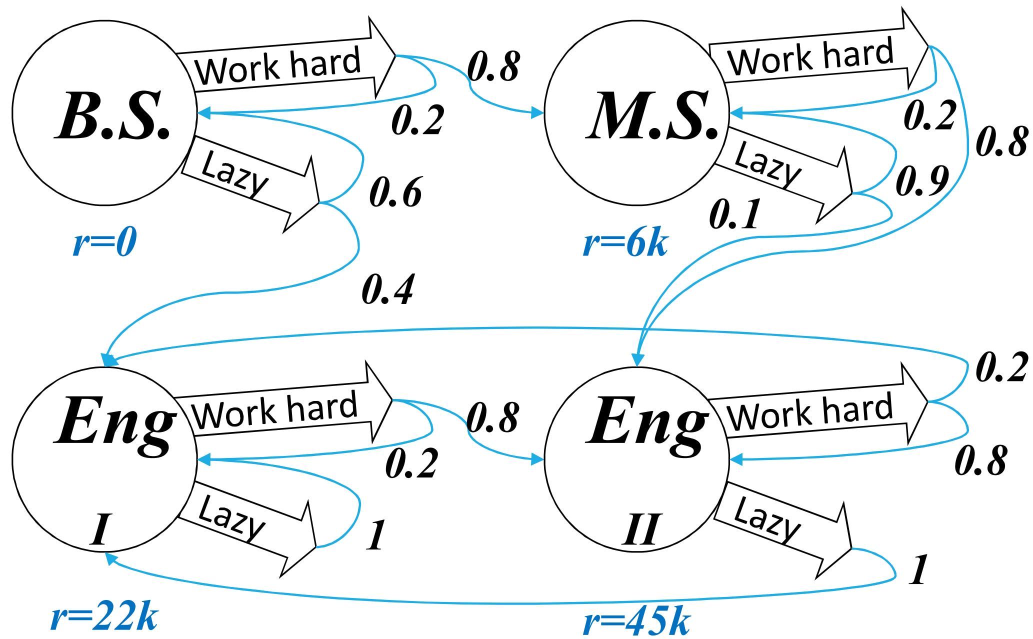
$R(s,a): reward$

$\gamma: discount = 0.9$

$[Find]$

$\pi^* = \arg\max U^\pi(s)$

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



Stochastic automata (state machine) diagram

MDP

- 4X3 world

$[GIVEN]$

$S : state \Rightarrow (x, y), x \in \{1,..,4\}, y \in \{1,..,3\}$

$A : action \Rightarrow a \in \{\uparrow, \leftarrow, \rightarrow, \downarrow\}$

$P(s'|s,a) : Transition\ probability$

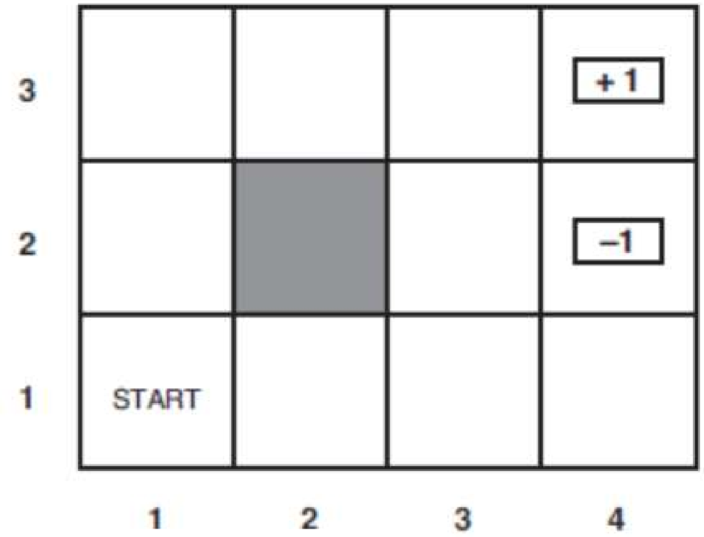
$R(s,a) : reward$

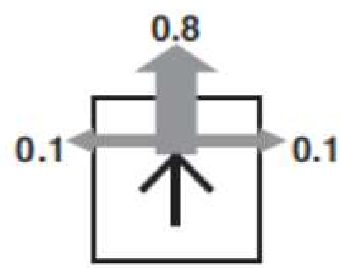
$\gamma : discount = 0.9$

$[Find]$

$\pi^* = \arg\max U^\pi(s)$

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



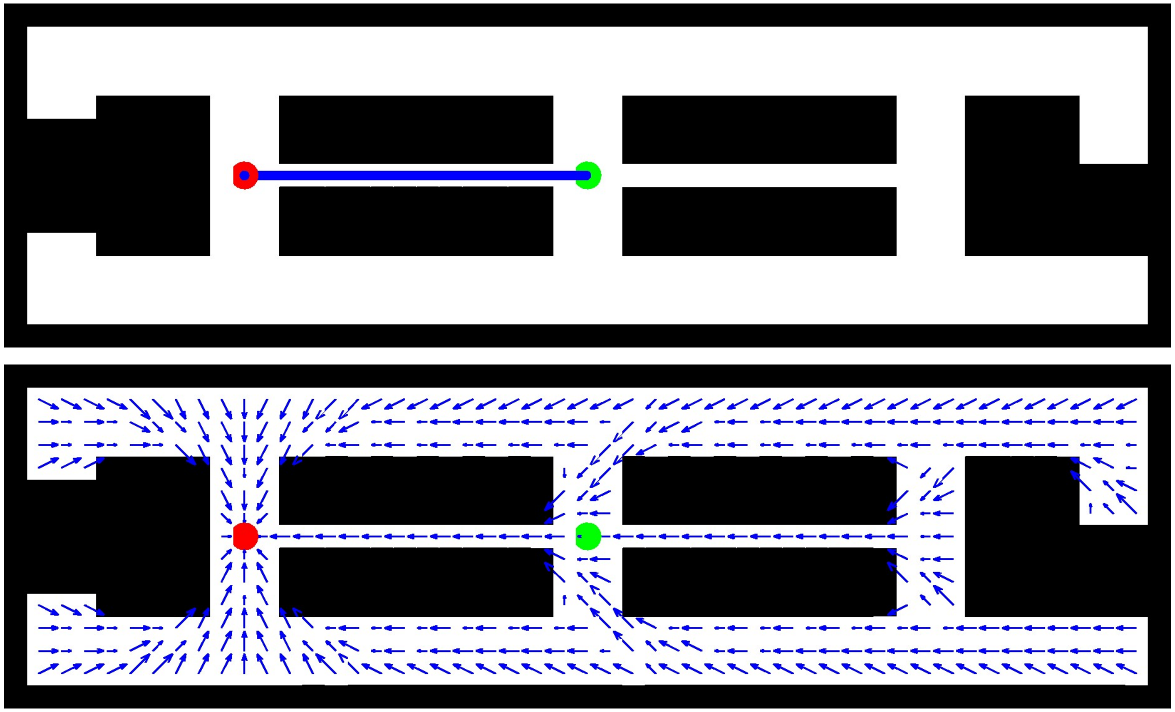


If the robot hits the wall,
It will bounce back.

s=(4,3), R=+1
s=(4,2), R=-1
Others, R=-0.04

MDP

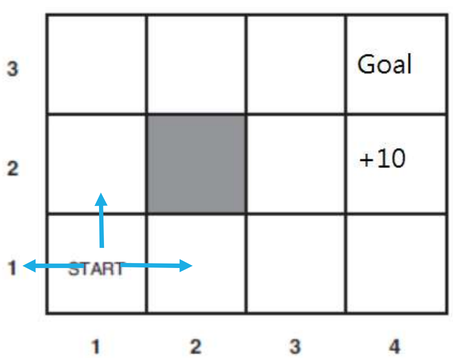
- Planning

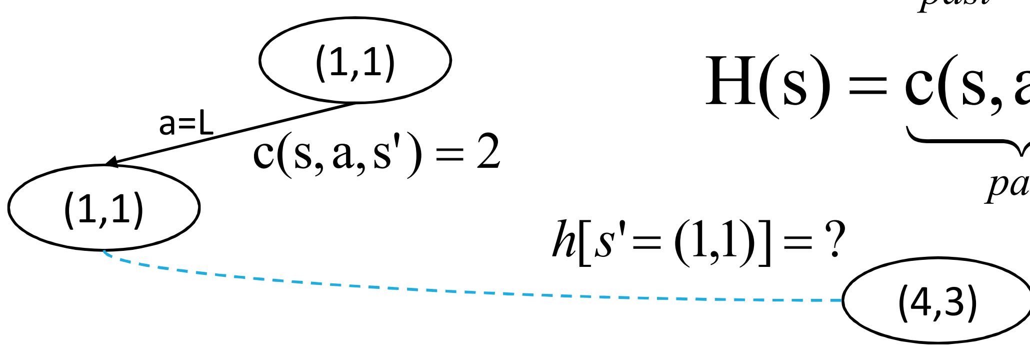


How to solve these problems?

Recitation: LRTA*

- For offline search, the robot has the environment information. However, for online search, the robot doesn't know how large the environment is. Hence, the robot needs to adopt a memory efficiency way – Markov chain.
- The state (s') at time t+1 only depends on the state (s) at time t.





$$f(i) = \underbrace{g(i)}_{past} + \underbrace{h(i)}_{future}$$

$$\mathrm{H(s)} = \underbrace{\mathrm{c(s, a, s')}}_{past} + \underbrace{h(s')}_{future}$$

Bellman Equation

- Assuming the agent chooses the optimal action,
  The utility of a state is the immediate reward for that state + the expected discounted utility of the next state.
- Bellman equation is dynamic programming, which solving subproblem to find the optimal solution of the problem.

$$U(s) = \gamma \max_a \left[R(s,a) + \sum_{s'} U(s') P(s' | s,a) \right]$$

$$or$$

$$U(s) = \underbrace{R(s)}_{\text{immediate reward}} + \underbrace{\gamma \max_a \left[\sum_{s'} U(s') P(s' | s,a) \right]}_{\text{expected discounted utility}}$$

Bellman Equation

- To illustrate Bellman equation, let's look at an example in deterministic and probabilistic cases.
- A cleaner robot in a 3X3 world.
- State: (1~3,1~3)
- Action: (left, up, right)
- Reward: +5 at goal (charging), -1 at other cells.
- Discount factor =1

	Goal	

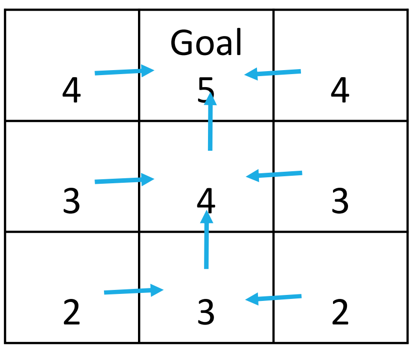
Bellman Equation

- Utility in deterministic cases

$$U(s) = R(s_0) + \gamma^1 R(s_1) + \gamma^2 R(s_2) + \ldots + \gamma^n R(s_n)$$
$$= \sum_t \gamma^t R(s_t)$$

4	Goal 5	4
3	4	3
2	3	2

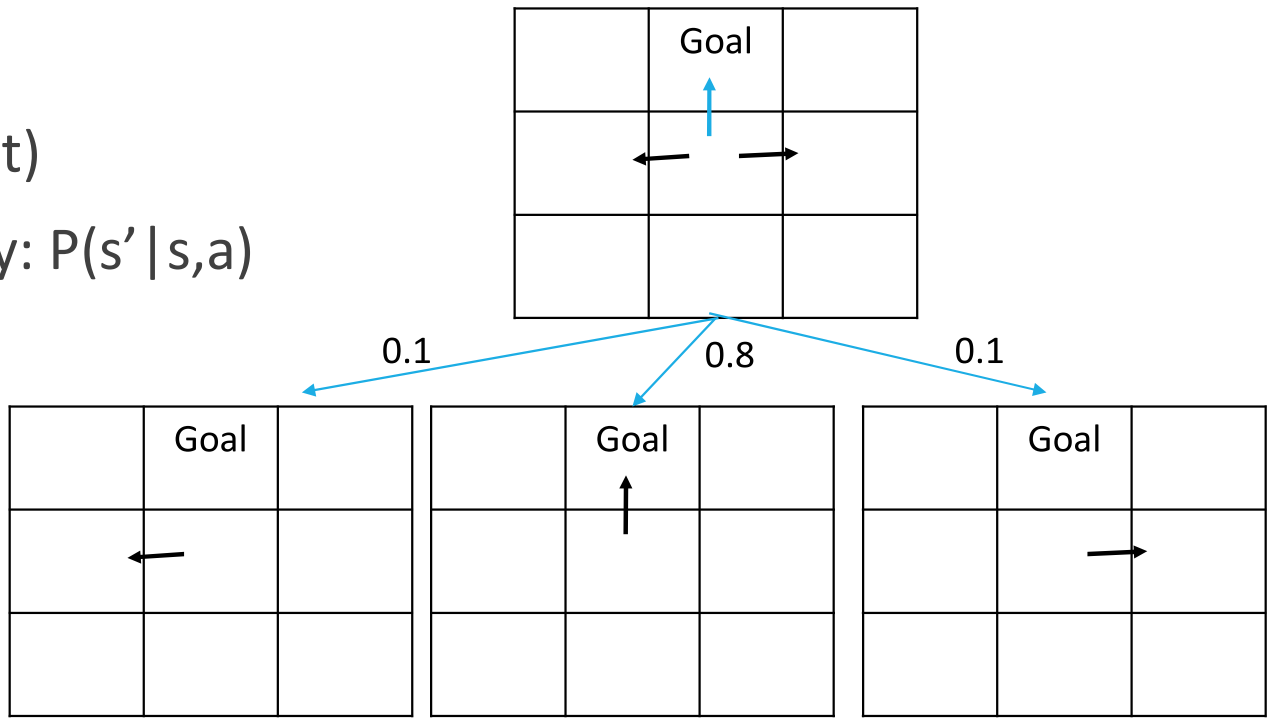
We can compute the utility function from the goal to each state



We can make optimal decisions with the utility function

Bellman Equation

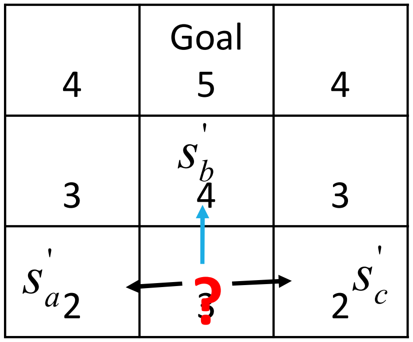
- Utility in probabilistic cases
  - When the robot made an action, the next state s' is with uncertainty
- State: (1~3,1~3)
- Action: (left, up, right)
- Transition probability: P(s'|s,a)



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

Bellman Equation

- Utility and action in deterministic cases
- After an action (a), s → s’



$$U(s) = R(s) + \gamma \max U(s')$$

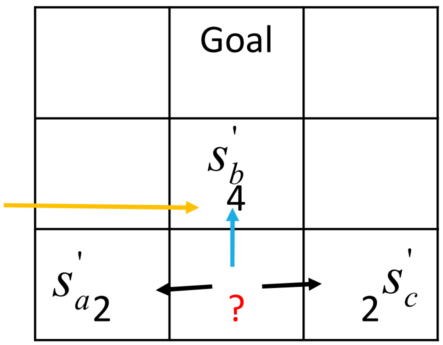
$$? = -1 + \max \begin{cases} U(s'_a) = 2 \\ U(s'_b) = 4 \\ U(s'_c) = 2 \end{cases}$$

The robot chooses the next state with the maximal utility (optimal decision)

Bellman Equation

- Utility and action in probabilistic cases
- After an action (a), s → s’ with P(s’|s,a)

Illustrate this case



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

$$U(s) = -1 + \max_a \begin{cases} 2*0.8 + 4*0.1 + 2*0.1 = 2.2 \\ 2*0.1 + 4*0.8 + 2*0.1 = 3.6 = 2.6 \\ 2*0.1 + 4*0.1 + 2*0.8 = 2.2 \end{cases}$$

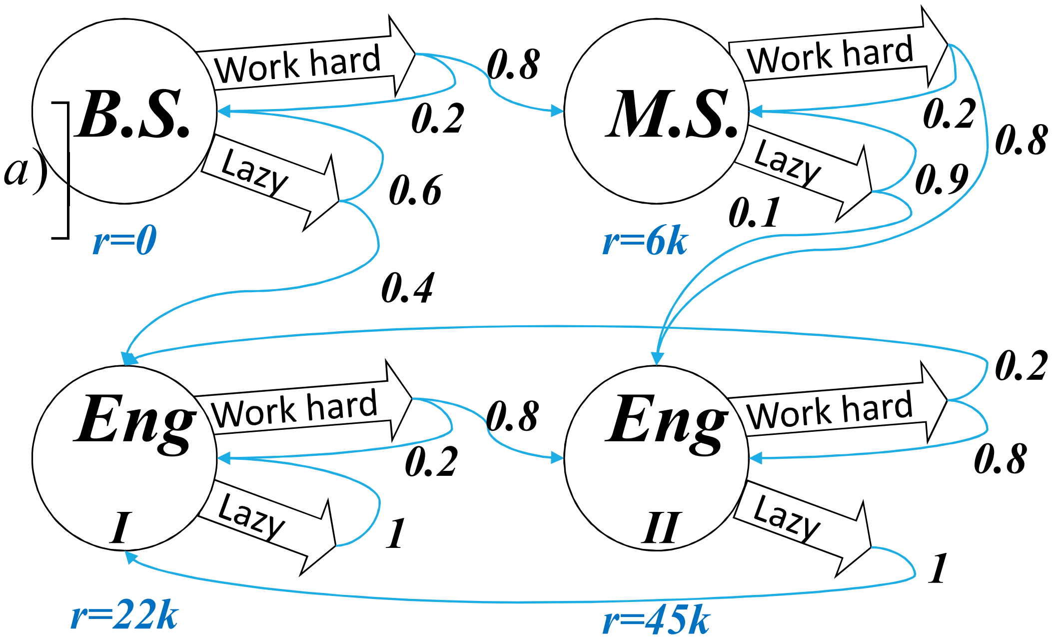
Bellman equation shows the relationship between U(s), U(s‘) and actions (a).
The assumption is “Markov chain.” The s’ is only depending on s and a.

- Assuming the agent chooses the optimal action,
  The utility of a state is <u>the immediate reward for that state</u> + <u>the expected discounted utility of the next state</u>.

Bellman Equation

- The M.S. life

$$U(s) = R(s) + \gamma \max_a \left[\sum_{s'} U(s') P(s' | s, a) \right]$$

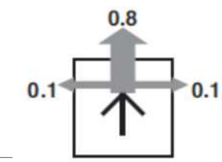


$$U(s = M.S.) = 6000 + \gamma \max_a \begin{bmatrix} 0.8U(s = E.II.) + 0.2U(s = M.S.) \\ 0.9U(s = M.S.) + 0.1U(s = E.II.) \end{bmatrix} \begin{matrix} \text{(work hard)} \\ \text{(lazy)} \end{matrix}$$

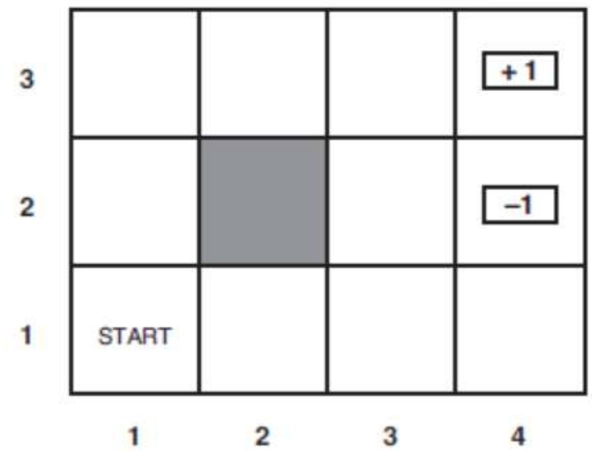
If we know U function, we can make optimal decisions! But, we don’t know it.

Bellman Equation

- 4x3 world



$$U(s) = R(s) + \gamma \max_a \left[\sum_{s'} U(s')P(s'|s,a) \right]$$



$$U(1,1) = -0.04 + \gamma \max_a \begin{array}{l} \left[\begin{array}{l} 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1) \\ 0.9U(1,1) + 0.1U(1,2) \\ 0.9U(1,1) + 0.1U(2,1) \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) \end{array}\right] \end{array} \begin{array}{c} (up) \\ (left) \\ (down) \\ (right) \end{array}$$

If we know U function, we can make optimal decisions! But, we don’t know it.

Bellman Equation

- We try to find optimal actions for 1-step, 2-step to infinite-step.
- 1-step

$$\pi_1 = \arg\max U(s)$$
$$U_1(s) = \gamma R(s,a)$$

- 2-step

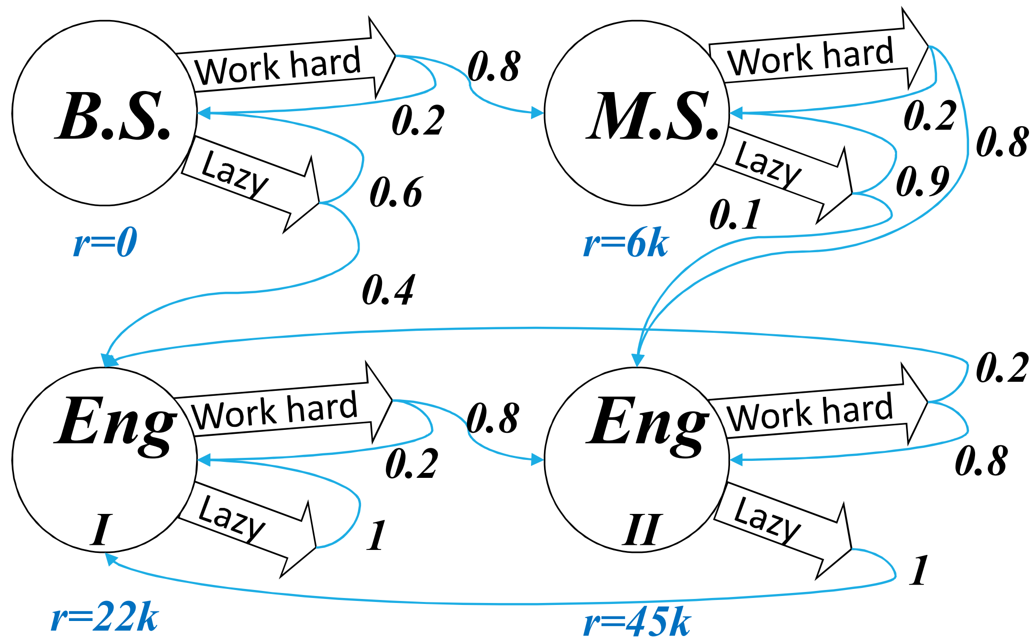
$$\pi_2 = \arg\max\left[R(s,a) + \sum_{s'} U_1(s')P(s'|s,a)\right]$$
$$U_2(s) = \gamma \max\left[R(s,a) + \sum_{s'} U_1(s')P(s'|s,a)\right]$$

4	Goal 5	4
3	4	3
2	3	2

- T=1: greedy policy
- T>1: finite horizon case, typically no discount
- T=infty: infinite-horizon case, finite reward if discount < 1

Bellman Equation

- How about the utility in infinity steps?



Bellman Equation

- We try to find optimal actions for 1-step, 2-step to infinite-step.
- Nth-step

$$\pi_T = \arg\max \left[R(s,a) + \sum_{s'} U_{T-1}(s') P(s' \mid s,a) \right]$$

$$U_T(s) = \gamma \max \left[R(s,a) + \sum_{s'} U_{T-1}(s') P(s' \mid s,a) \right]$$

4	Goal 5	4
3	4	3
2	3	2

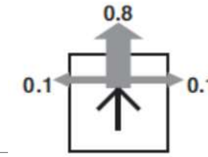
- Infinite-step

$$\pi_\infty = \arg\max \left[R(s,a) + \sum_{s'} U_\infty(s') P(s' \mid s,a) \right]$$

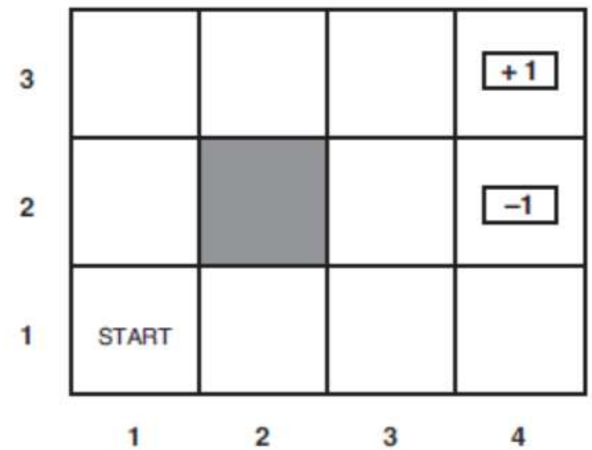
$$U_\infty(s) = \gamma \max \left[R(s,a) + \sum_{s'} U_\infty(s') P(s' \mid s,a) \right]$$

- T=1: greedy policy
- T>1: finite horizon case, typically no discount
- T=infty: infinite-horizon case, finite reward if discount < 1

Value iteration



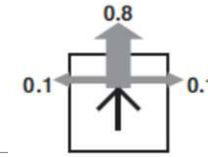
- If there are N states, there are N Bellman equations. There are N unknown utilities of states.
- However, we cannot use *A=BX* to solve this problem since it's not a linear algebra formulation. (max is a nonlinear operator)



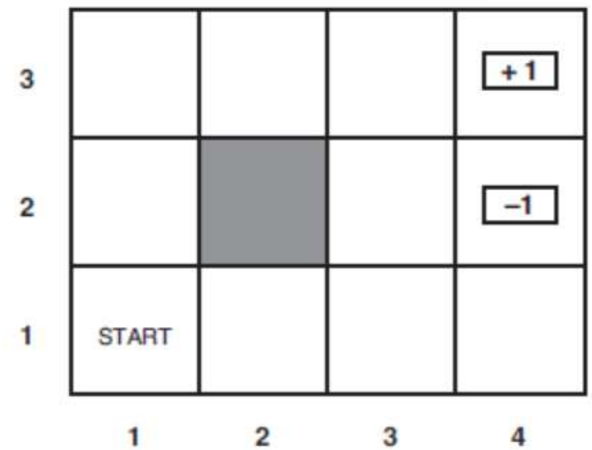
$$U(1,1) = -0.04 + \gamma \max_a \begin{bmatrix} 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1) \\ 0.9U(1,1) + 0.1U(1,2) \\ 0.9U(1,1) + 0.1U(2,1) \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) \end{bmatrix} \begin{matrix} (up) \\ (left) \\ (down) \\ (right) \end{matrix}$$

[1] R. Bellman, "A markovian decision process," Technical report, DTIC Document, 1957.

Value iteration



- We can solve this problem using iterative algorithms.
- Just guess some values and keep updating U functions until it’s converged.
- How to solve it iteratively?
- Is it converged?



$$U(1,1) = -0.04 + \gamma \max_a \begin{bmatrix} 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1) \\ 0.9U(1,1) + 0.1U(1,2) \\ 0.9U(1,1) + 0.1U(2,1) \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) \end{bmatrix} \begin{matrix} (up) \\ (left) \\ (down) \\ (right) \end{matrix}$$

Value iteration

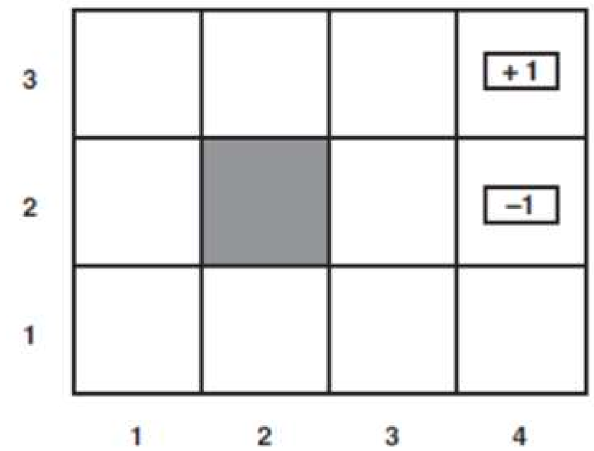
function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
   ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states 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**
    $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$ (Update by Bellman EQ)
    if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Value iteration

- 1^{st} iteration

$$U(1,1) = U(1,2) = U(1,3) = U(2,1) =$$
$$U(2,3) = U(3,1) = U(3,2) = U(3,3) = U(4,1) = 0$$
$$U(4,2) = -1$$
$$U(4,3) = +1$$



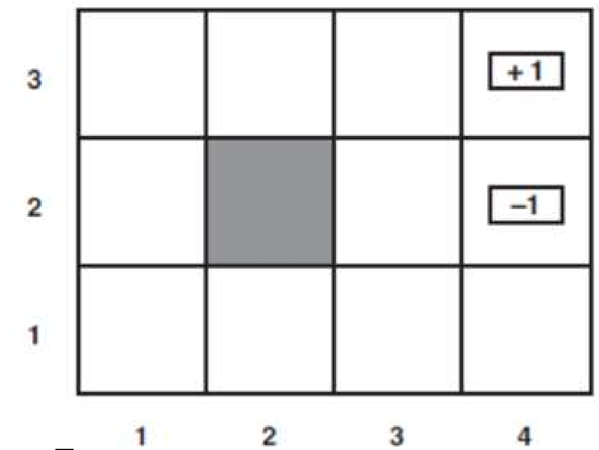
$$U(1,1) = -0.04 + \gamma \max_a \begin{bmatrix} 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1) \\ 0.9U(1,1) + 0.1U(1,2) \\ 0.9U(1,1) + 0.1U(2,1) \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) \end{bmatrix} \begin{matrix} (up) \\ (left) \\ (down) \\ (right) \end{matrix}$$

$$= -0.04$$

Value iteration

- 1st iteration

$$U(1,1)=U(1,2)=U(1,3)=U(2,1)=$$
$$U(2,3)=U(3,1)=U(3,2)=U(3,3)=U(4,1)=0$$
$$U(4,2)=-1$$
$$U(4,3)=+1$$



$$U(4,3)=R(4,3)+\gamma\cdot 0=+1$$

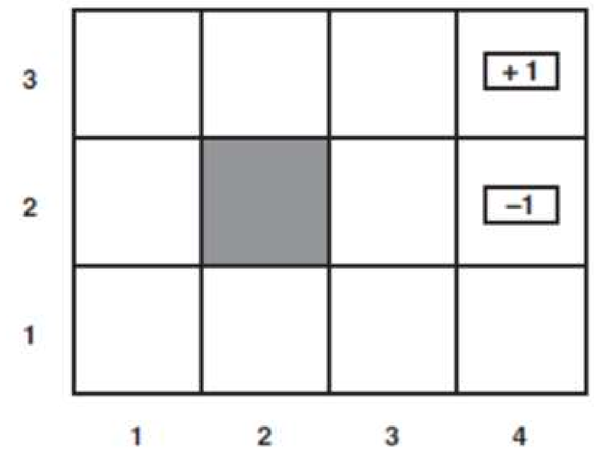
$$U(3,3)=-0.04+\gamma\max_a\begin{bmatrix}0.8U(3,3)+0.1\underline{U(4,3)}+0.1U(2,3)\\0.8U(2,3)+0.1U(3,2)+0.1U(3,3)\\0.8U(3,2)+0.1U(2,3)+0.1\underline{U(4,3)}\\0.8\underline{U(4,3)}+0.1U(3,3)+0.1U(3,2)\end{bmatrix}\begin{matrix}(up)\\(left)\\(down)\\(right)\end{matrix}$$

$$=-0.04+0.9(0.8*1+0.1*0+0.1*0)=0.68$$

Value iteration

- 2^{nd} iteration – update U function

$$U(1,1) = U(1,2) = U(1,3) = U(2,1) =$$
$$U(2,3) = U(3,1) = U(3,2) = U(4,1) = -0.04$$
$$U(3,3) = 0.68$$
$$U(4,2) = -1$$
$$U(4,3) = +1$$



Compute U(S) again until it's converged!

Value iteration

- After convergence, the robot can find optimal decision at each state

$$\pi = \arg\max_a P(s'|s,a)U(s')$$

$$= \arg\max_a \begin{bmatrix} 0.8U(1,2)+0.1U(2,1)+0.1U(1,1) \\ 0.9U(1,1)+0.1U(1,2) \\ 0.9U(1,1)+0.1U(2,1) \\ 0.8U(2,1)+0.1U(1,2)+0.1U(1,1) \end{bmatrix} \begin{matrix} (up) \\ (left) \\ (down) \\ (right) \end{matrix}$$

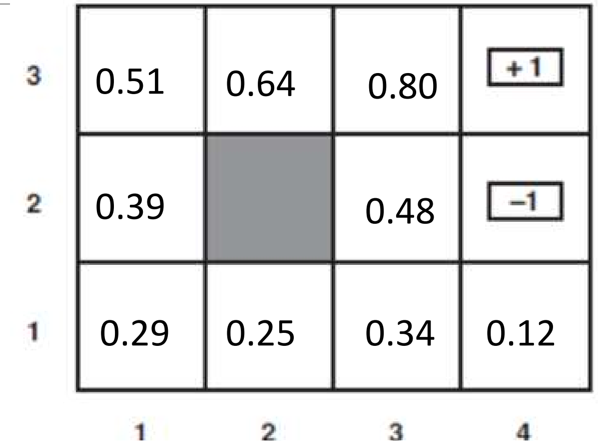
$$= \arg\max_a \begin{bmatrix} 0.8*0.39+0.1*0.25+0.1*0.29 \\ 0.9*0.29+0.1*0.39 \\ 0.9*0.29+0.1*0.25 \\ 0.8*0.25+0.1*0.39+0.1*0.29 \end{bmatrix} \begin{matrix} (up) \\ (left) \\ (down) \\ (right) \end{matrix}$$

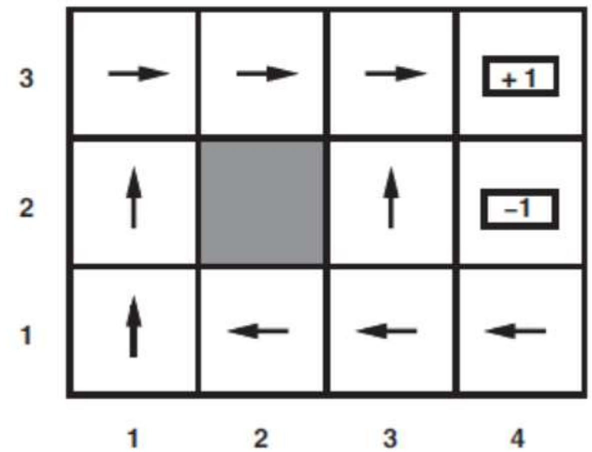
3	0.51	0.64	0.80	+1
2	0.39		0.48	–1
1	0.29	0.25	0.34	0.12
	1	2	3	4

Repeat this step for each state.

Value iteration

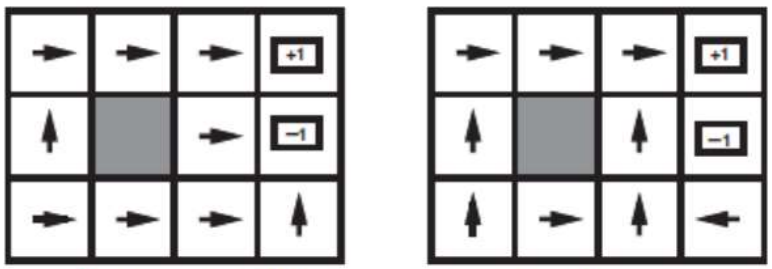
- After convergence, the robot can find optimal decision at each state



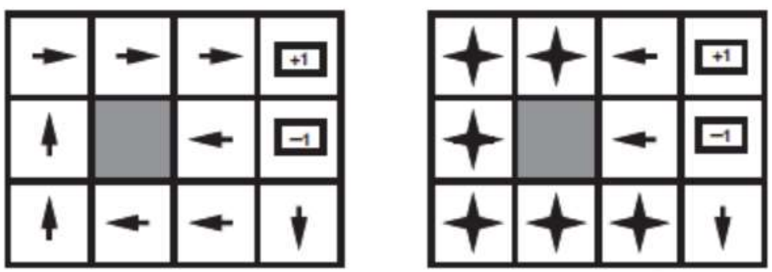


Value iteration

- Punishment for not arriving at goals will change the optimal policy.



$R(s) < -1.6284$ $\quad$ $-0.4278 < R(s) < -0.0850$



$-0.0221 < R(s) < 0$ $\quad$ $R(s) > 0$

3	0.51	0.64	0.80	+1
2	0.39		0.48	−1
1	0.29	0.25	0.34	0.12
	1	2	3	4

The rewards of S=(4,2) and S=(4,3) will affect the optimal policy also.

Value iteration

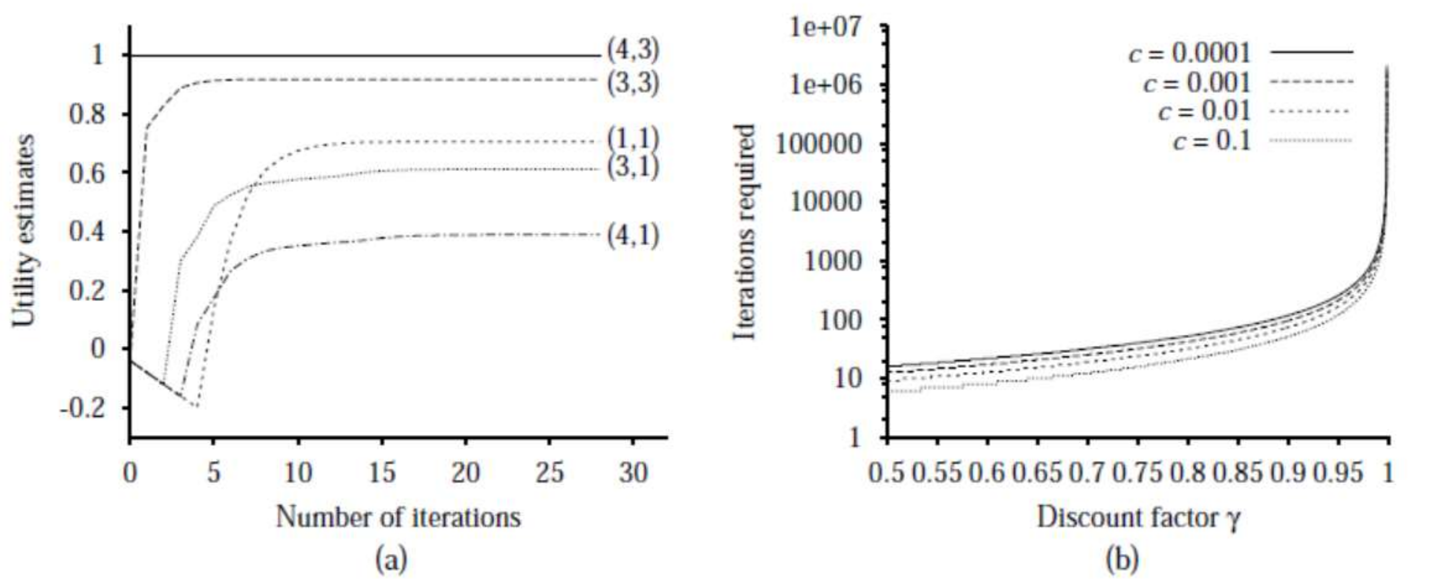


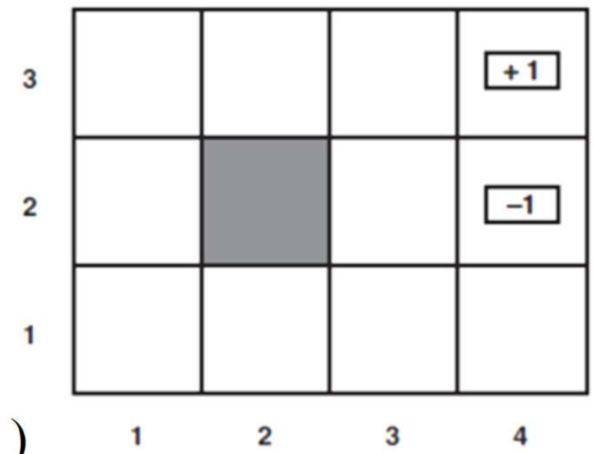
Figure 17.5 **FILES:** . (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations k required to guarantee an error of at most $\epsilon = c \cdot R_{\max}$, for different values of c, as a function of the discount factor γ.

Policy iteration

- If you check the policy of each iteration of VI algorithm, you will find the optimal policy is converged before the U function is converged.
- Hence, policy iteration could find the solution faster than value iteration.

Policy iteration

- The major concept of PI is as follows:
  - Assign actions
  - Compute U based on current actions
  - Repeat until convergence



$$\text{Assign a} = \{\text{up}\} \text{ in each state}$$

$$U_i(1,1) = -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1)$$

$$U_i(1,2) = -0.04 + 0.8U_i(1,3) + 0.2U_i(1,2)$$

$$\vdots$$

$$U_i(4,1) = -0.04 + 0.8U_i(4,2) + 0.1U_i(3,1) + 0.1U_i(4,1)$$

Since there is no “max” operator in PI, there are N unknown variables and N equations.
Solve AX=B problem. However, it’s O(N^3).

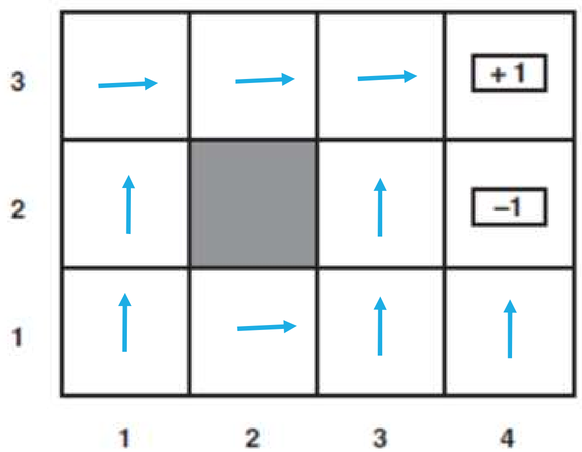
Policy iteration

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

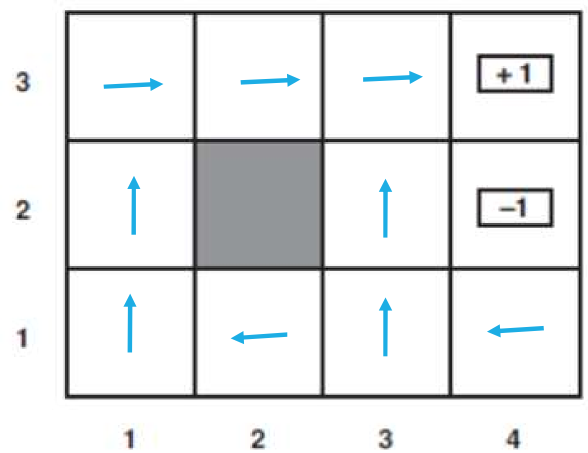
  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)   (Compute by Pseudo inverse)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\, U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a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

Policy iteration

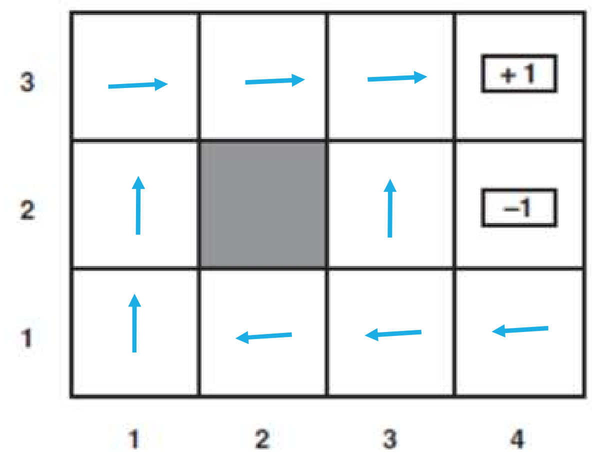
- Results



1st iteration



2nd iteration



3rd iteration

In this initial setting, the robot got the optimal solution after 4 iterations.

Policy iteration

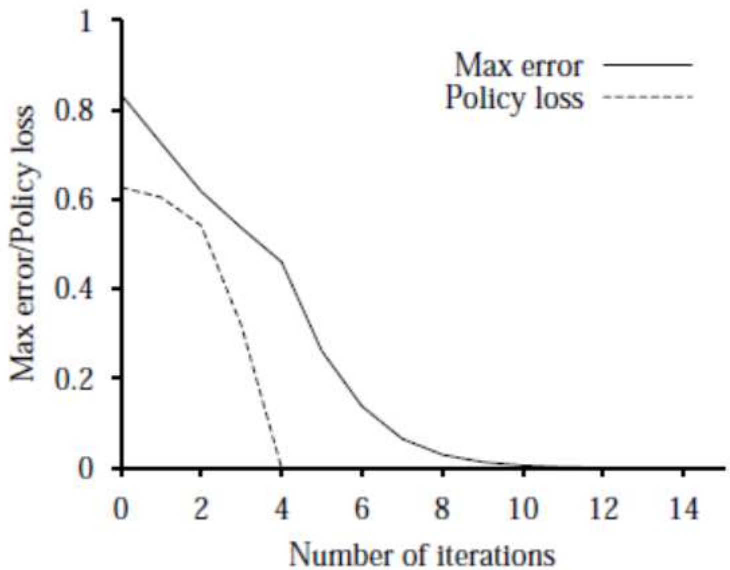


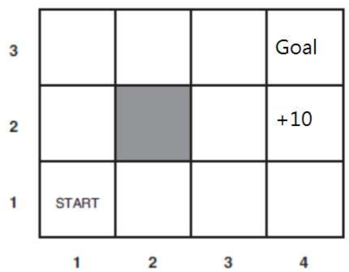
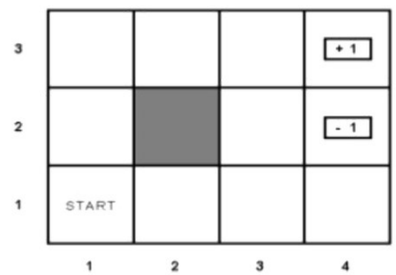
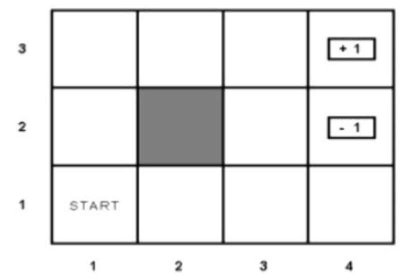
Figure 17.6 **FILES:** . The maximum error $||U_i - U||$ of the utility estimates and the policy loss $||U^{\pi_i} - U||$, as a function of the number of iterations of value iteration.

Conclusions

- MDP is a model for finding sequential optimal decisions.
  - State: fully observable
  - State transition: stochastic (motion model)
- Bellman equation is the key to solve MDP problems.
- MDP is widely applied to decision problems. However, the basic assumption is that the state is deterministic!
- POMDP is a model for finding sequential optimal decisions.
  - State: stochastic (sensor model → Bayes theorem)
  - State transition: stochastic (motion model)

Conclusions

- LRTA* | MDP (RL) | POMDP

LRTA*	MDP (RL)	POMDP
Deterministic action	Probabilistic actions	Probabilistic actions and states
		
$s, a \rightarrow s'$	$P(s' \mid s, a)$	$P(s' \mid s, a), P(s)$

L2: Uninformed search
L3: Heuristic search (LRTA*)
L4: Adversarial search
L5: Bayes theorem
L6: Bayes theorem over time

L7: MDP
L9: Reinforcement learning
L10: GP and LWPR
L11: Naïve Bayes and Perceptron
L12: Adaboost
L13: Deep learning and DRL

L8: POMDP

Q&A