Making Complex Decisions

Sequential Decision Problems

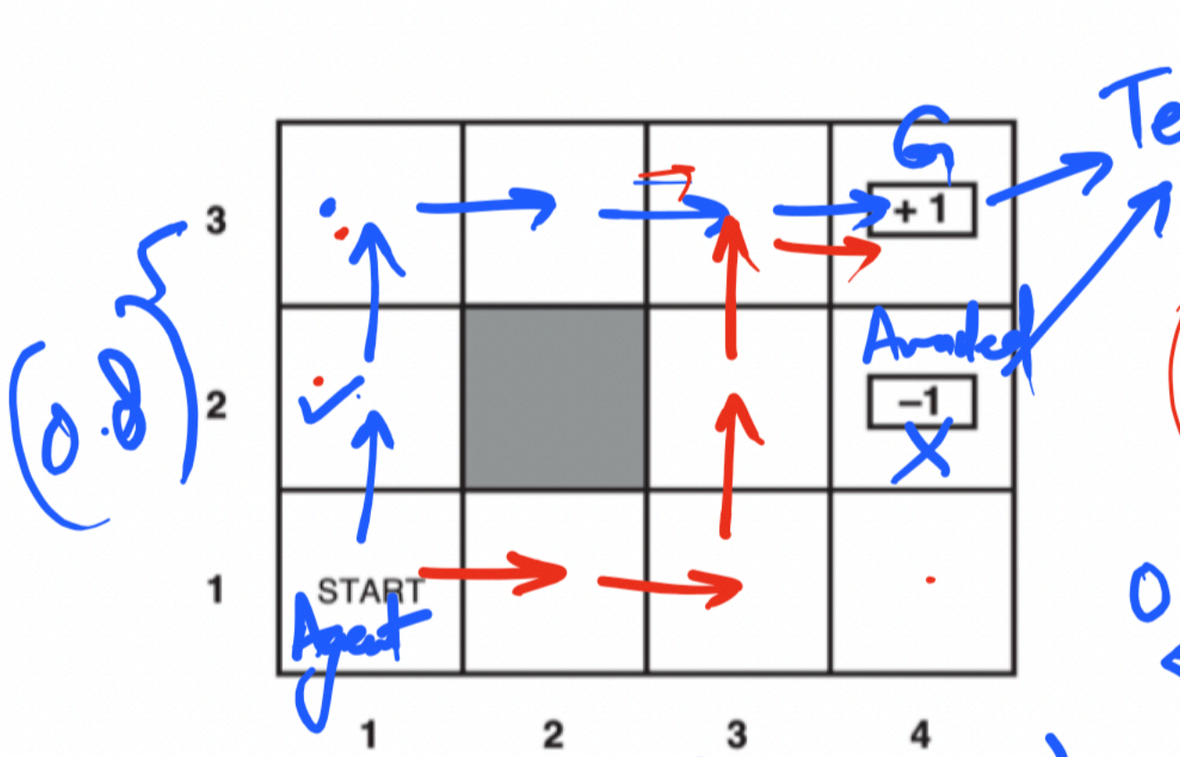
- In sequential decision problems, the agent's utility depends on a sequence of decisions.
  - A sequence of states is called as environment history

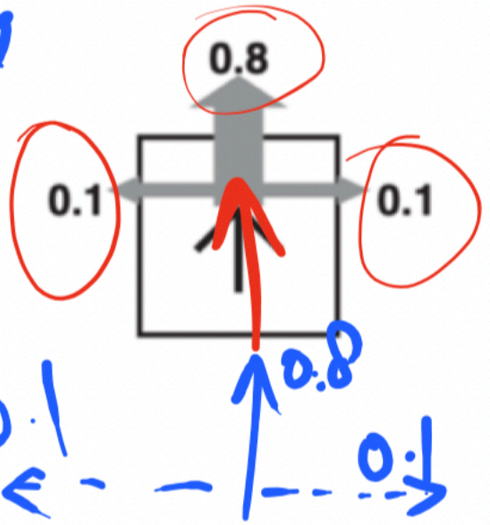
$U(h)$ $h: s^0 s^1 s^3 \ldots s^k$

- The environment is stochastic – the outcome of actions is uncertain (described using distribution over possible states).

$P(s' \mid s, a)$ $s \xrightarrow{a} s'$

- A simple 4x3 environment (fully observable)





- The state transitions are stochastic and Markovian, governed by distribution

- The state transitions are stochastic and Markovian, governed by distribution $P(s' \mid s, a)$
- The utility function is a function of a sequence of states (an environment history)
- The agent receives a reward $R(s)$ in a state s
- The utility of an environment can be formulated as the additive rewards (i.e sum of the rewards) received.

Markov Decision Process (MDP)

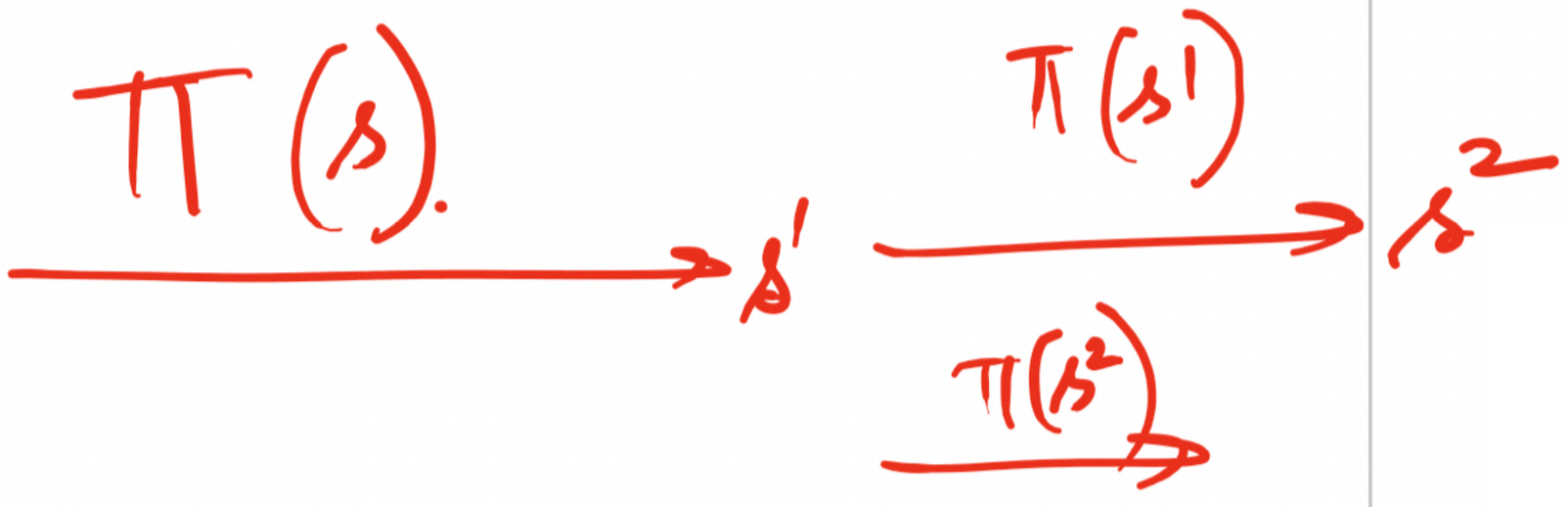
- An MDP is a sequential decision problem for a fully observable, stochastic environment, with a Markov transition model and additive rewards.

- An MDP consists of
  - a set of states with initial state s_0
  - ACTIONS (s) applicable in a state s
  - Transition model $P(s' \mid s, a)$
  - Reward function $R(s)$

Stationarity Assumption behind this discounted additive rewards

Solving an MDP

- The solution to a sequential decision problem cannot be a fixed sequence of actions.
  - After all, the outcome state of an action is not fixed
- Instead, the solution should be a policy that specifies the action to be taken for any state.



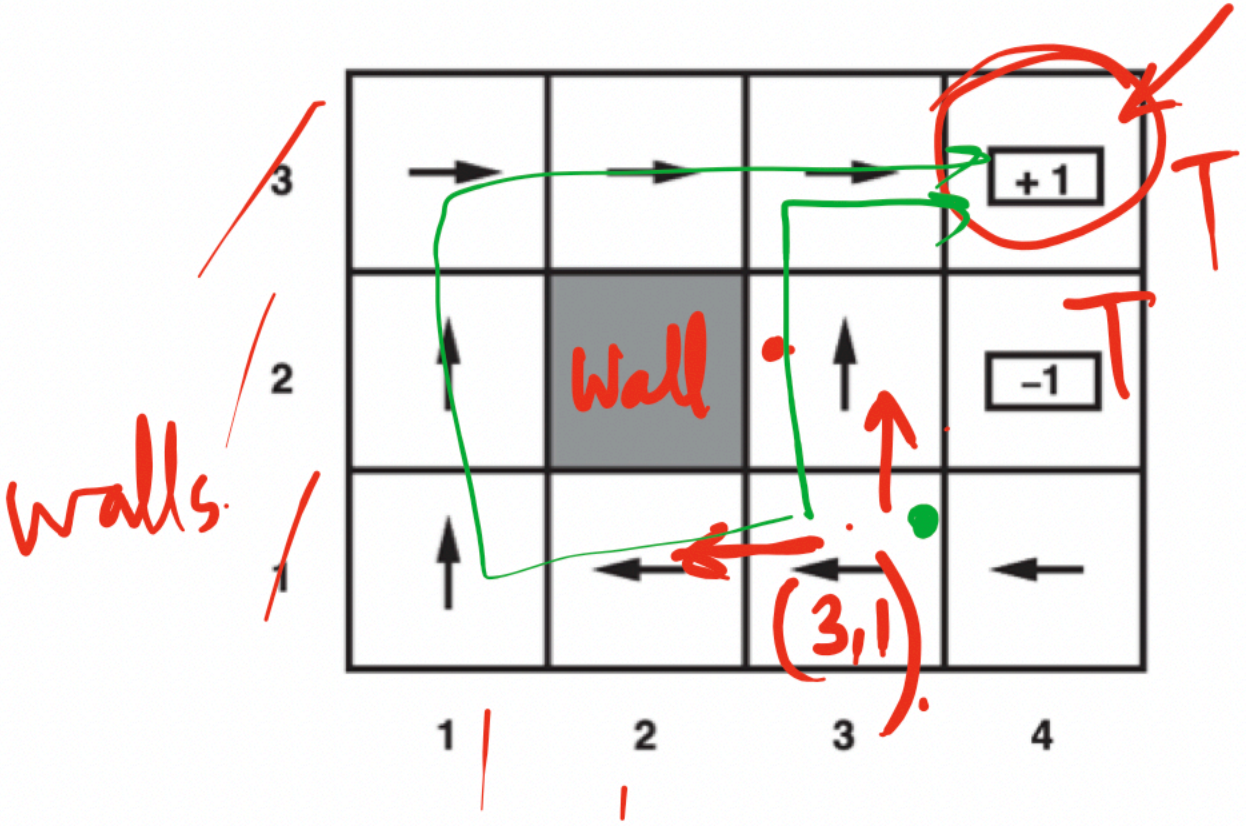
Stochastic nature of the environment – use of expected utility

- When an agent sets out to execute a fixed policy in a stochastic environment the outcome can be different every time
  - That means, different environment histories can be created

- Therefore, to assess a policy, we measure its expected utility over the possible environment histories that can be generated by the policy.

An optimal policy π^*

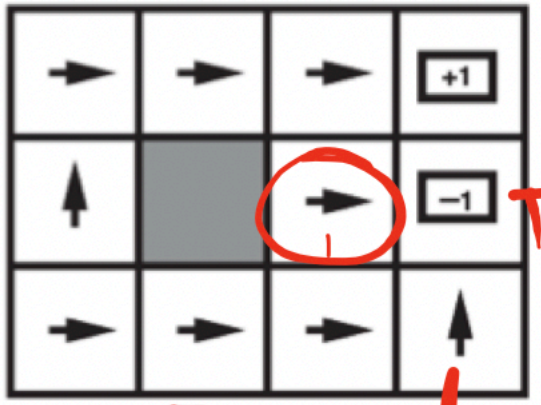
- An optimal policy is a policy that yields the highest expected utility.
- The action to be executed in a given state s is given as $\pi^*(s)$



The figure shows the optimal policy when $R(s) = -0.04$ for the non-terminal states

Very −

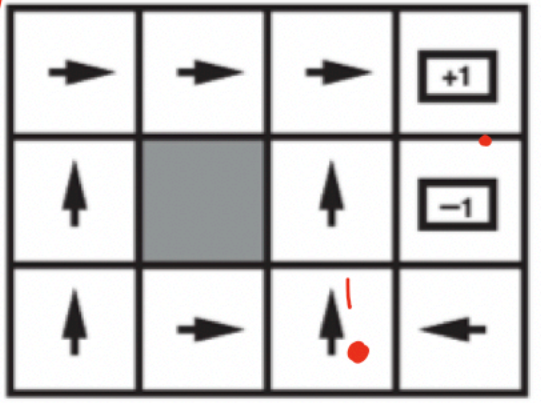
$R(s) < -1.6284$



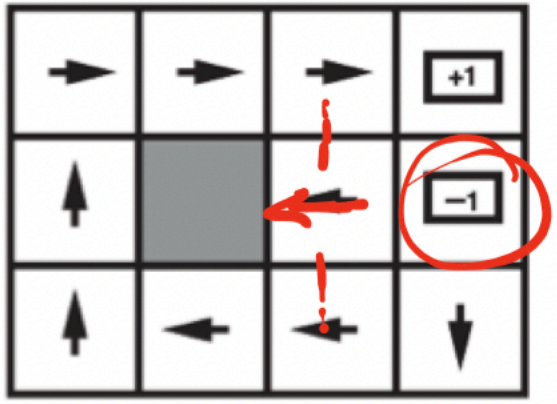
desperate to get to any Terminal state.

Moderately −

$-0.4278 < R(s) < -0.085$

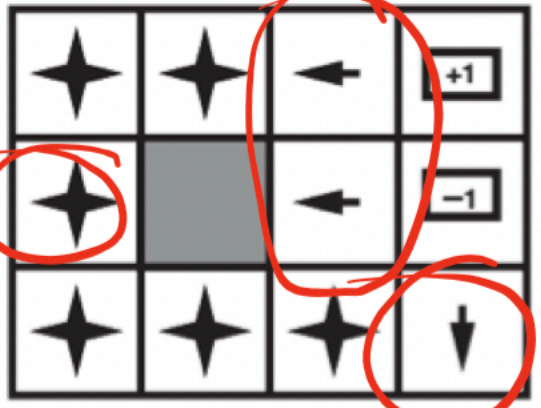


$-0.0221 < R(s) < 0$



Mildly −ve.

$R(s) > 0$



Decision making over a finite horizon

- A finite horizon means that there is a fixed time N after which the game is over
- That means the utility is not affected by what states are visited after the finite horizon

$$U_h([s_0, s_1, \ldots, s_{N+k}]) = U_h([s_0, s_1, \ldots, s_N]) \text{ for all } k > 0$$

- With finite horizon, the optimal action in a given state could change over time
  - That is, the optimal policy for a finite horizon is non stationary.

Decision making over a infinite horizon

- For an infinite horizon (no fixed deadline), the optimal action for a given state remains fixed every time the agent visits the state.
- The optimal policy is stationary

→ helps to maximize the Expected Utility

Stationarity of Agent's Preferences

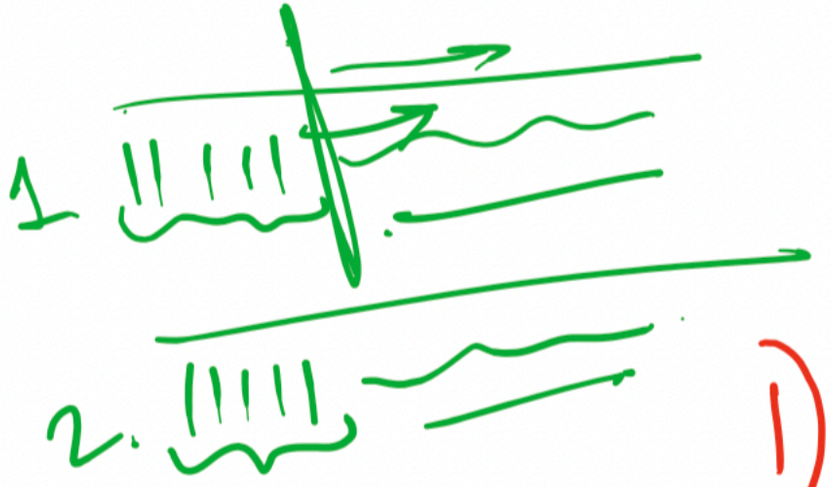
- Stationarity for preferences means that if two sequences

1. $[s_0, s_1, s_2, \ldots]$ and
2. $[s'_0, s'_1, s'_2, \ldots]$ begin with the same state $[s'_0 = s_0]$ then the two sequences

should be preference-ordered in the same way as the sequences

$[s_1, s_2, \ldots]$ and

$[s'_1, s'_2, \ldots]$



1) MPI
2) MUI

Consequence of stationarity of Agent's Preferences

- Under stationarity there are just two coherent ways to assign utilities to sequences:

- Additive Rewards:

The utility of a state sequence is

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

- Discounted Rewards

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

γ = discount factor, $\gamma \leq 1$

Infinitely long environment histories

- If the sequence of actions continues indefinitely,

  i.e the policy is such that the agent does not reach the terminal state,

  then the additive rewards lead to utility values tending to ∞

- However, for discounted rewards, the utility of an infinite sequence is finite.

$$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = \frac{R_{\max}}{(1-\gamma)}$$

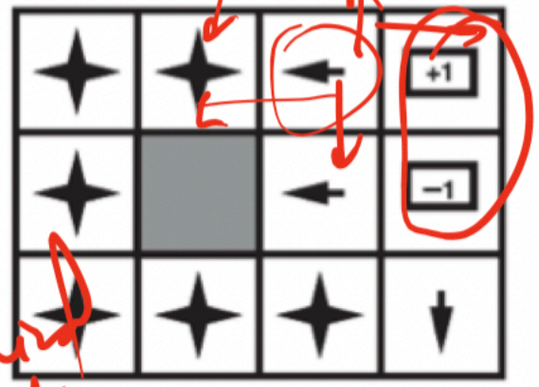
$R_{max} = \max\left(R(s_0), R(s_1) \ldots\right)$

$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \gamma^3 R(s_3) + \ldots \gamma^t R(s_t) + \ldots$

$\leq R_{max}\left(1 + \gamma + \gamma^2 + \ldots\right)$

- Thus, additive rewards can be used only with a proper policy, i.e. a policy which is guaranteed to reach the terminal states.

- An example of improper policy



$R(s) > 0$

Agent will never reach Terminal state.

agent wants to stay or has no particular choice of actions

How to compare policies?

- We can compare policies by comparing their expected utilities
- The expected utility $U^\pi(s)$ obtained by executing a policy π starting in state s is given by

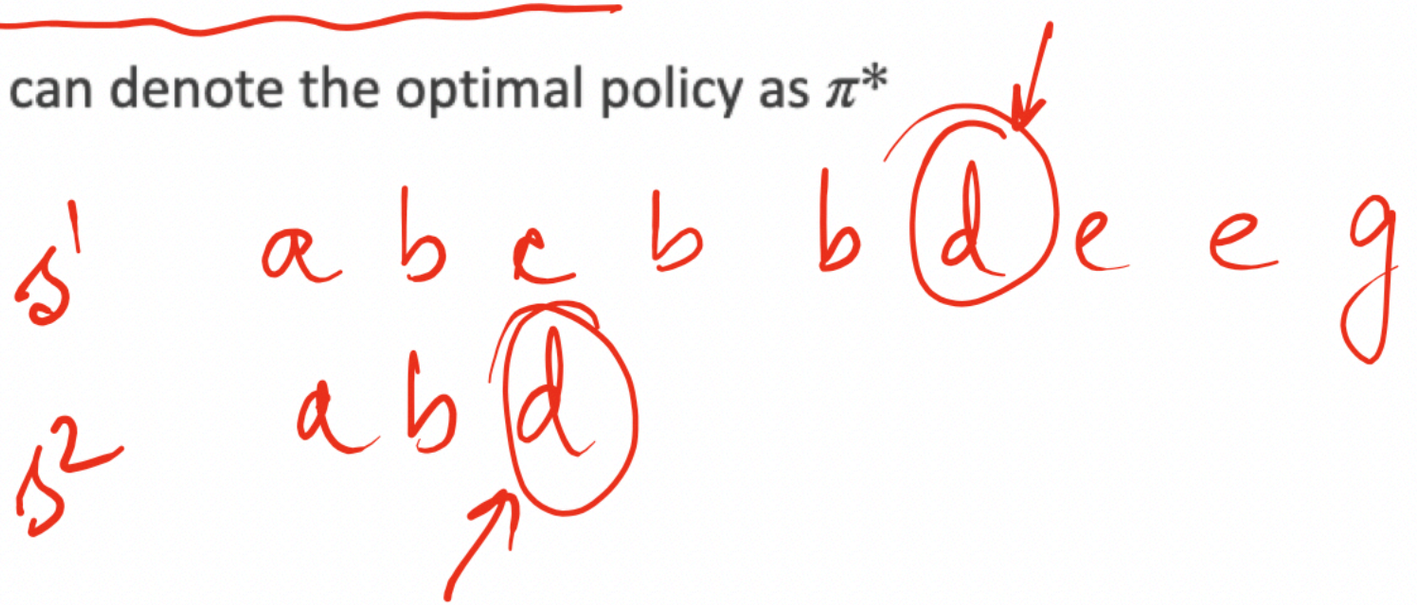
$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

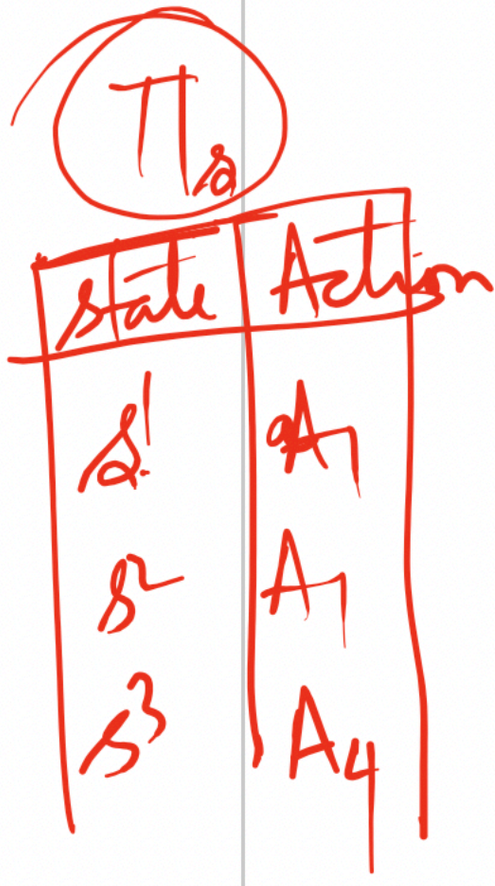
Here, the expectation is with respect to the probability distribution over state sequences determined by s and π

- We can select the optimal policy when s is the starting state

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

- When using discounted utilities with infinite horizons, the optimal policy is independent of the starting state.
  - So we can denote the optimal policy as π^*

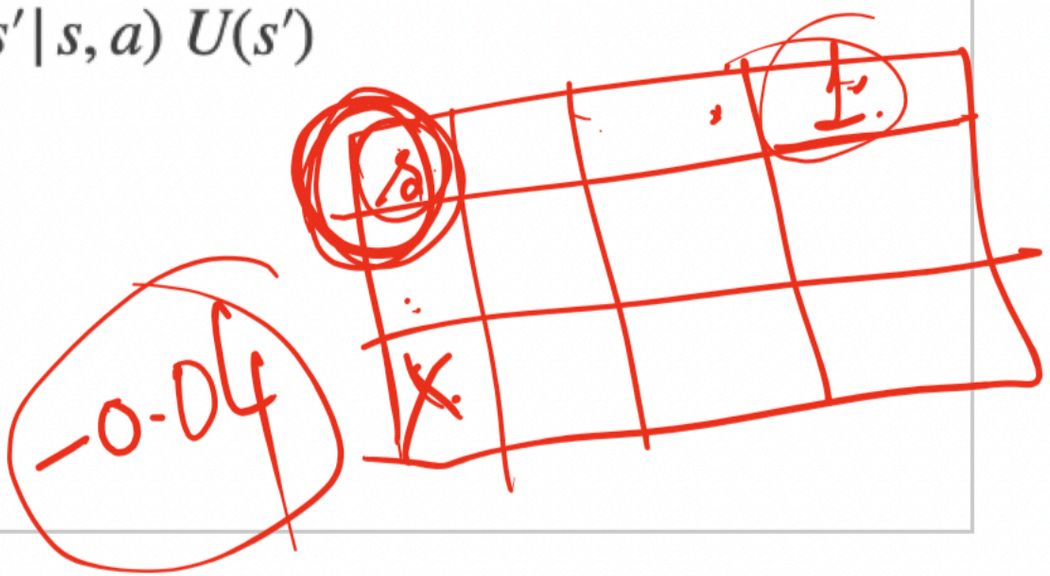




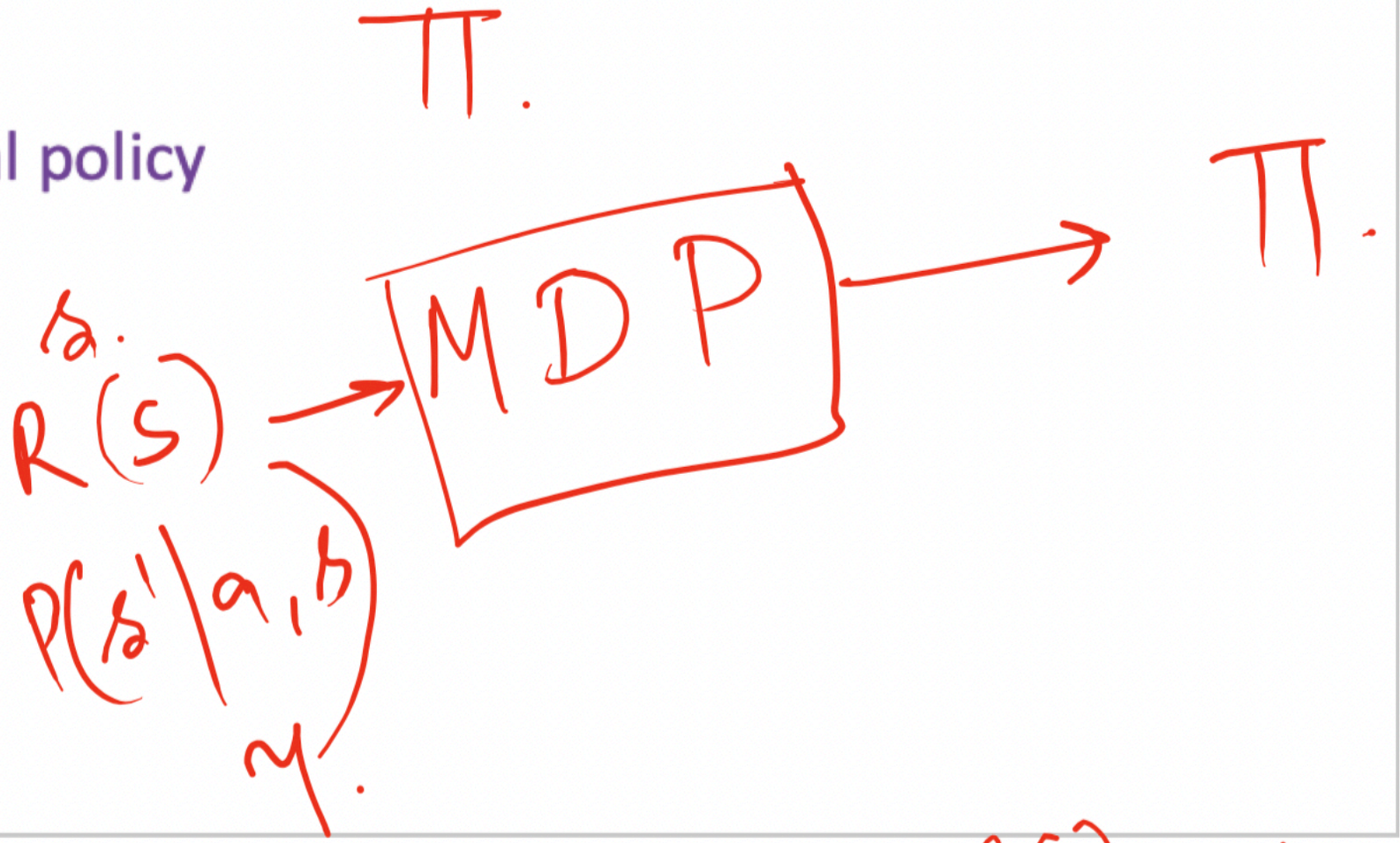
- The expected utility of a state obtained by executing the optimal policy π^* is the true utility $U^{\pi^*}(s)$ for the state s.
- We can denote the true utility function as $U(s)$
- The agent can choose the action that maximises the expected utility of the subsequent state

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

(an average) what best benefit we can derive by proceeding further from that state



Calculating the optimal policy



Value Iteration Method

- Bellman equation

  The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action

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

$$U(1,1) = -0.04 + \gamma \max[\, 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), \quad (Up)$$
$$0.9U(1,1) + 0.1U(1,2), \quad (Left)$$
$$0.9U(1,1) + 0.1U(2,1), \quad (Down)$$
$$0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\,]. \quad (Right)$$

Value Iteration Method

- For n possible states, there are n Bellman equations containing a total of n unknowns.
- However, these simultaneous equations are nonlinear.
- So we use an iterative scheme which makes use of Bellman update in each step

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

- If we apply the update infinite often, we are guaranteed to reach an equilibrium (a unique solution for final utility values).

Value Iteration Method

- Let U_i and U_i' be any two utility vectors.

  Then we have the contraction property:

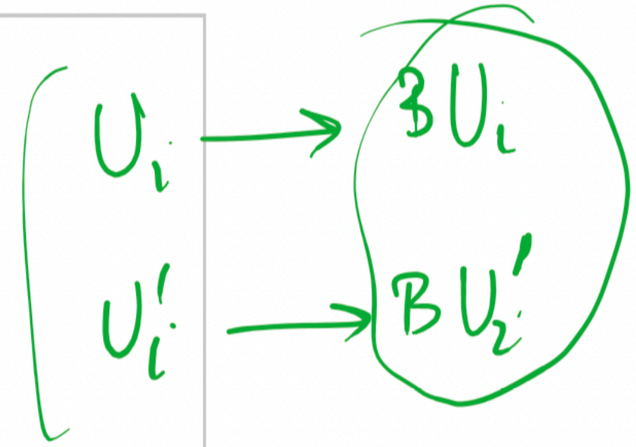
$$\|BU_i - BU_i'\| \leq \gamma \|U_i - U_i'\|$$

- The Bellman Update is a contraction by a factor of γ on the space of utility vectors.

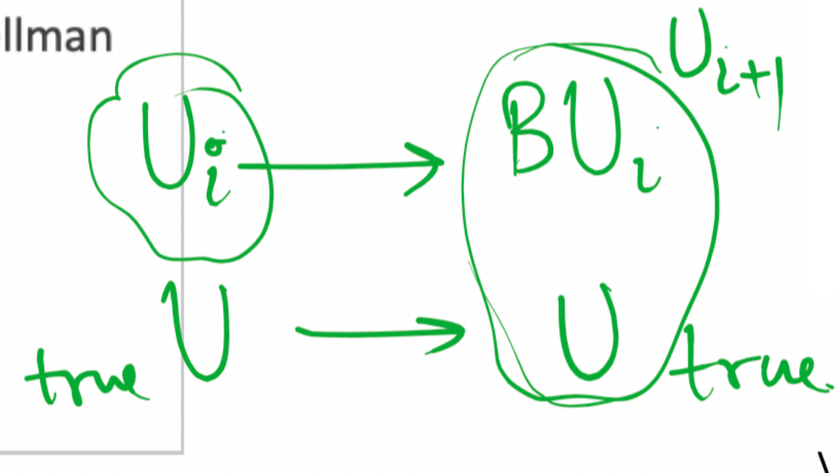
- It follows that value iteration always converges to a unique solution of the Bellman equations when $\gamma < 1$

- If we replace U_i' with the true utilities U, then

$$\|BU_i - U\| \leq \gamma \|U_i - U\|$$



$U \leftarrow BU$



- The number of iterations N required for the error $\|BU_i - U\|$ to fall below ϵ can be shown as

$$N = \left\lceil \frac{\log(2R_{max}/(\epsilon(1-\gamma)))}{log(1/\gamma)} \right\rceil$$ where $R_{\max}$ is the maximum reward of any state

- N grows rapidly as γ becomes close to 1

γ: discount rewards

additive rewards

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

RHS

$U \xrightarrow{B} U'$

$|U' - U|$

- If $\|U_{i+1} - U_i\| < \epsilon(1-\gamma)/\gamma$ then $\|U_{i+1} - U\| < \epsilon$

↓ RHS.

↓ true utility

Convergence of Value Iteration Method.

- It is possible to get an optimal policy even when the utility function estimate is inaccurate
- If the utility estimate yields a MEU policy π_i based on one-step look ahead using U_i and

$$\pi_i(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U_i(s')$$

if $U^{\pi_i}(s)$ is the utility obtained if π_i is executed starting in s

then the policy loss is $\|U^{\pi_i} - U\|$

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

- In practice, π_i becomes optimal long before U_i has converged

Expected utility of the next state

$s \xrightarrow{a} s'$

$P(s' \mid s, a)$

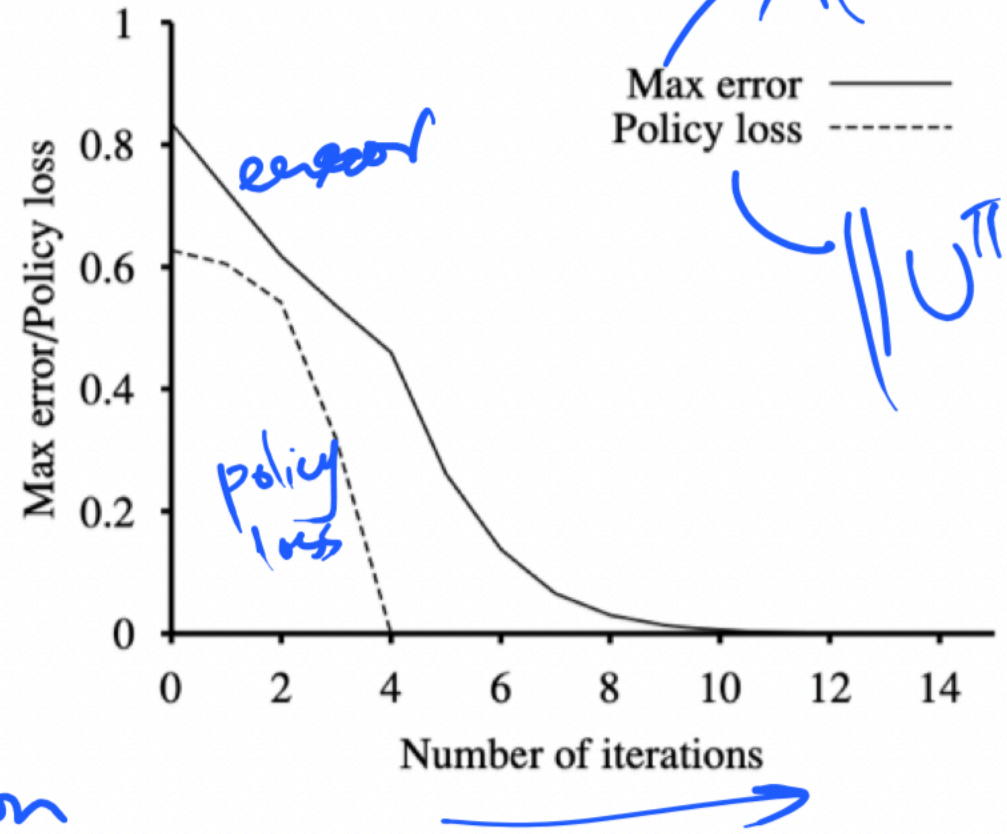
U : true utility vector

U^{π_i} : utility computed by following the policy π

true $U \rightarrow \pi^*$

$U_i \rightarrow \pi_i \rightarrow \pi_i \rightarrow U^{\pi_i}$

policy iteration


Max error
Policy loss
Max error/Policy loss
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10
12
14
Number of iterations

2 Policy Iteration to estimate the Utility vector

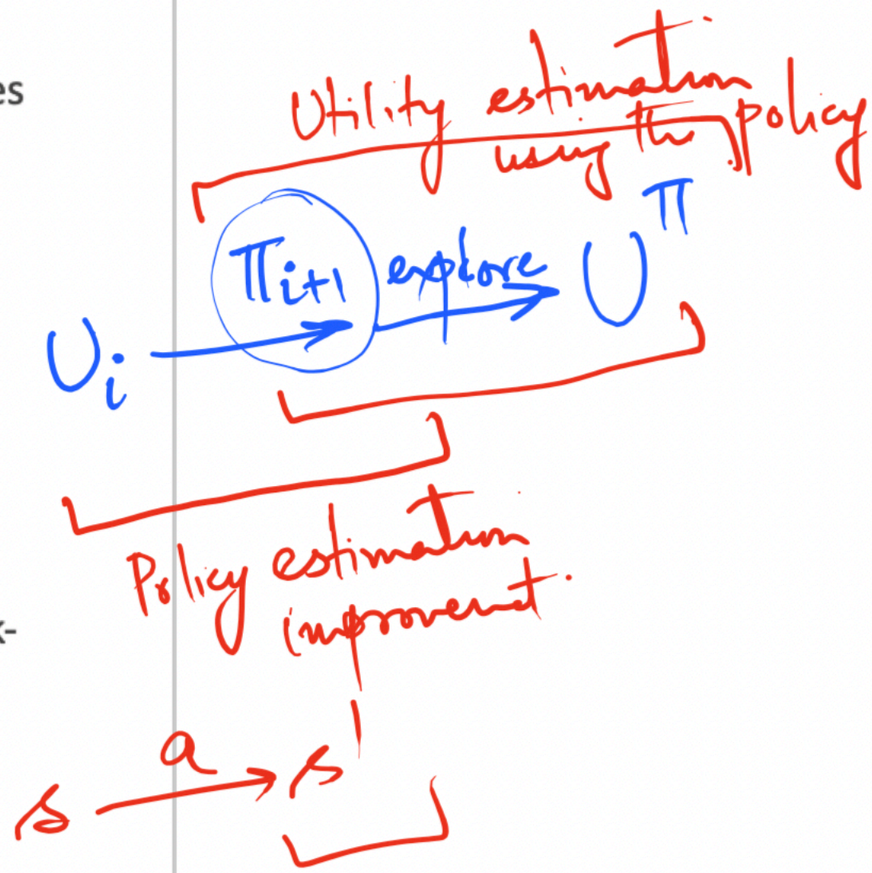
- The policy iteration algorithm begins with some initial policy π_0 and alternates the following two steps:
  - Policy evaluation: Given a policy π_i, calculate $U_i = U^{\pi_i}$, the utility of each state if π_i were to be executed.

Using π to explore the environment

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

  - Policy improvement: Calculate a new MEU policy π_{i+1} using one-step look-ahead based on U_i

$$\pi_{i+1}(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U_i(s')$$



- The algorithm terminates when the policy improvement step yields no change in the utilities. The final utility function U_i is a fixed point in the Bellman update.

- In the policy evaluation step we solve for the updated utility vector

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

$a \equiv \pi_i(s)$

Bellman's Update
No max operator

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

- The equations are linear and can be solved in $O(n^3)$ time.
- If n is large, we can get a good approximation of utilities by performing some number of simplified value iterations steps

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s))\, U_i(s')$$

action is recommended by the policy.

2 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)
    $unchanged? \leftarrow$ 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**
        $\pi[s] \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

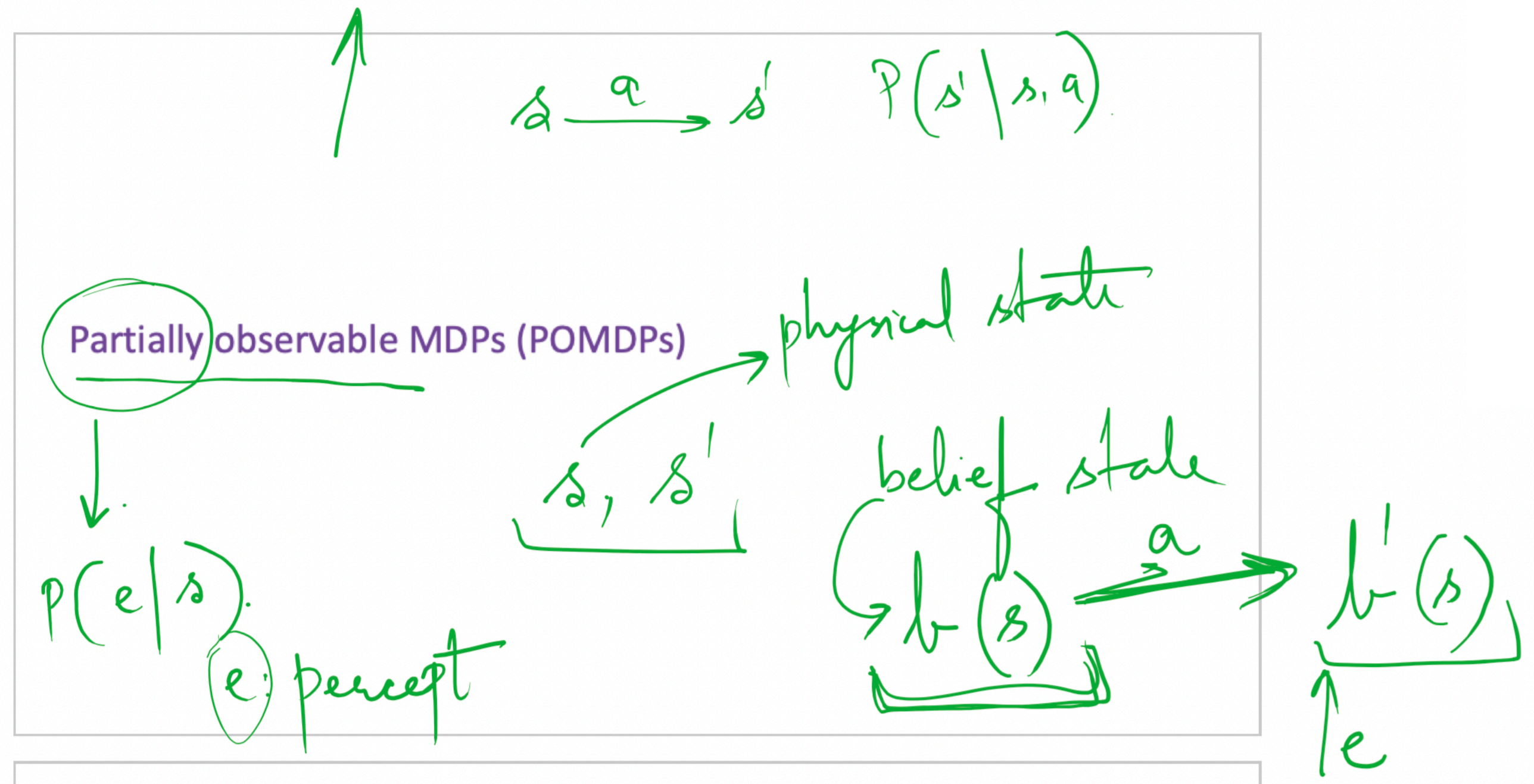
estimate the U_i given π_i

Policy update

Further implications

- Instead of updating the utility or policy for all states at once one can pick up any subset of states and apply either of the two update steps to that subset.
- This very general algorithm is called asynchronous policy iteration.

- Use of heuristics can lead to efficiency – e.g., the algorithm can concentrate on updating the values of states that are likely to be reached by a good policy.


Partially observable MDPs (POMDPs)
physical state
belief state
percept

Partially Observable Environment

- An agent does not really know in which state it is in.
- So it cannot take an action $\pi(s)$ recommended for that state.

Partially Observable MDP

- A POMDP has
  - the transition model $P(s' \mid s, a)$ for physical states
  - actions $A(s)$
  - Reward $R(s)$
  - Sensor Model $P(e \mid s)$ to capture the likely noise in the measurements
  - Belief state – a probability distribution over the actual states

actions depend on the percepts that are received

→ Contingency plans.

Actions → plans.

$\Pi(b)$ → plans.

POMDP → Convert → MDP.

transition $P(b' \mid b, a, e)$

- The 4×3 world can make use of a noisy sensor that can measure the number of adjacent walls, but may give a wrong value with probability 0.1