Reinforcement Learning and Dynamic Programming

Talk 5

by Daniela and Christoph

Content

Reinforcement Learning Problem

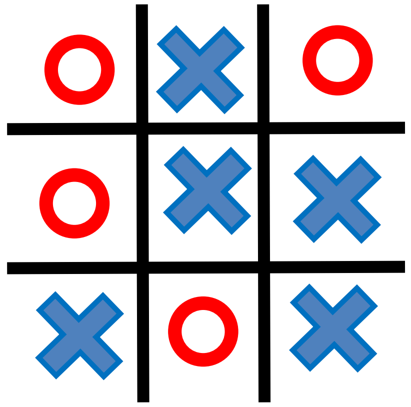
- Agent-Environment Interface
- Markov Decision Processes
- Value Functions
- Bellman equations

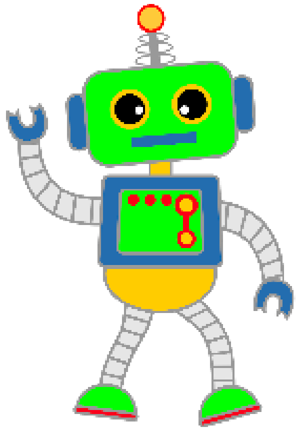
Dynamic Programming

- Policy Evaluation, Improvement and Iteration
- Asynchronous DP
- Generalized Policy Iteration

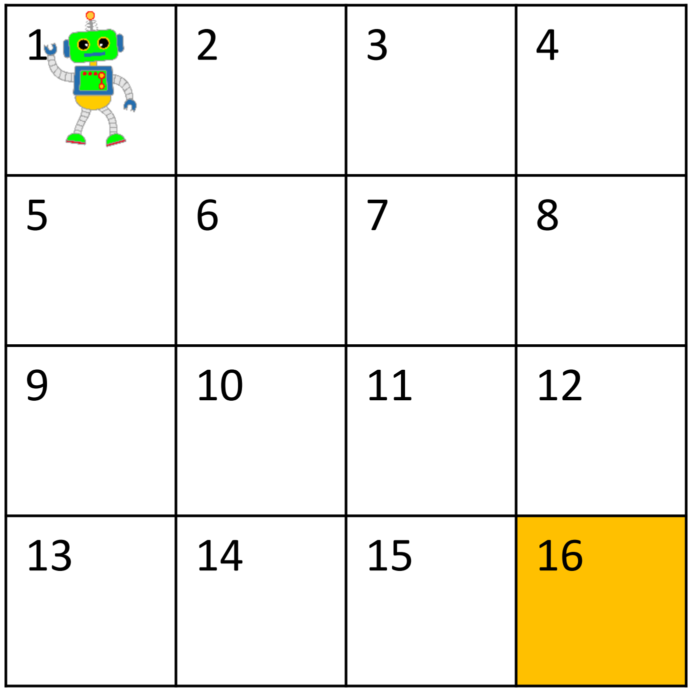
Reinforcement Learning Problem

- Learning from interactions
- Achieving a goal





Example robot



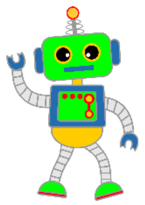
actions

Reward is -1 for all transition, except for the last transition.
Reward for the last transition is 2.

Agent-Environment Interface

Agent

- Learner
- Decision maker



Agent

Environment

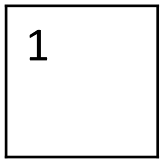
- Everything outside of the agent

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Environment

Interaction

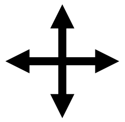
- State: $S_t \in S$

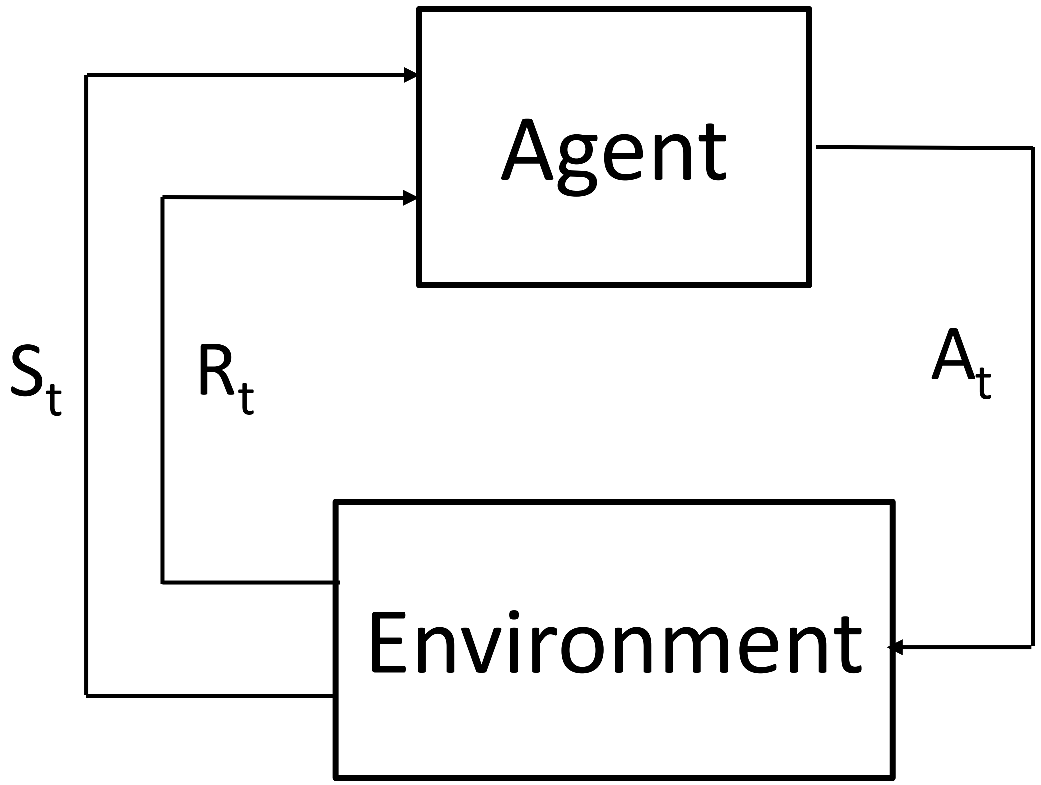


- Reward: $R_t \in \mathbb{R}$

-1 or 2

- Action: $A_t \in A(S_t)$

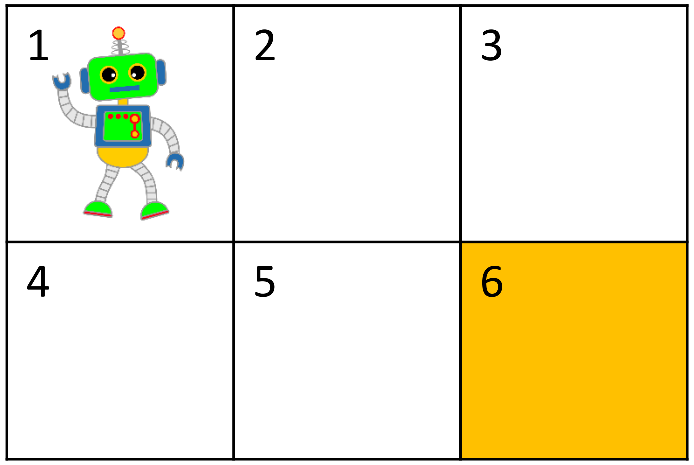


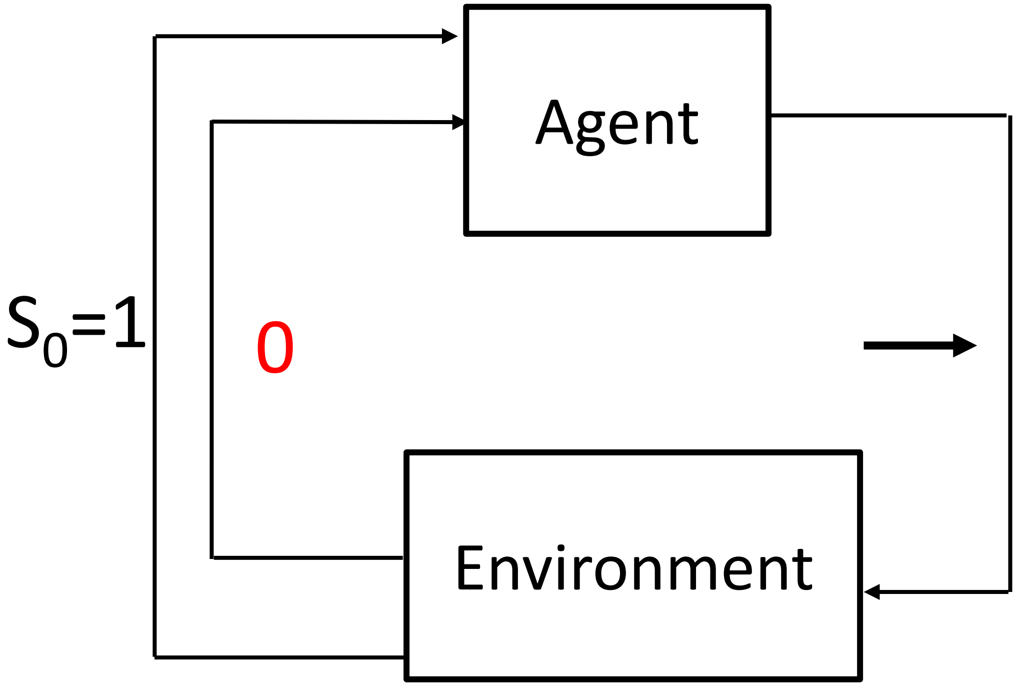


Discrete time steps

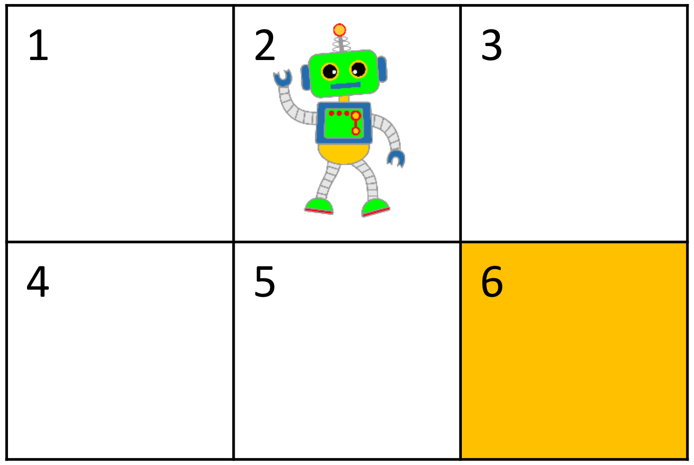
- $t = 0{,}1{,}2{,}3 \ldots$

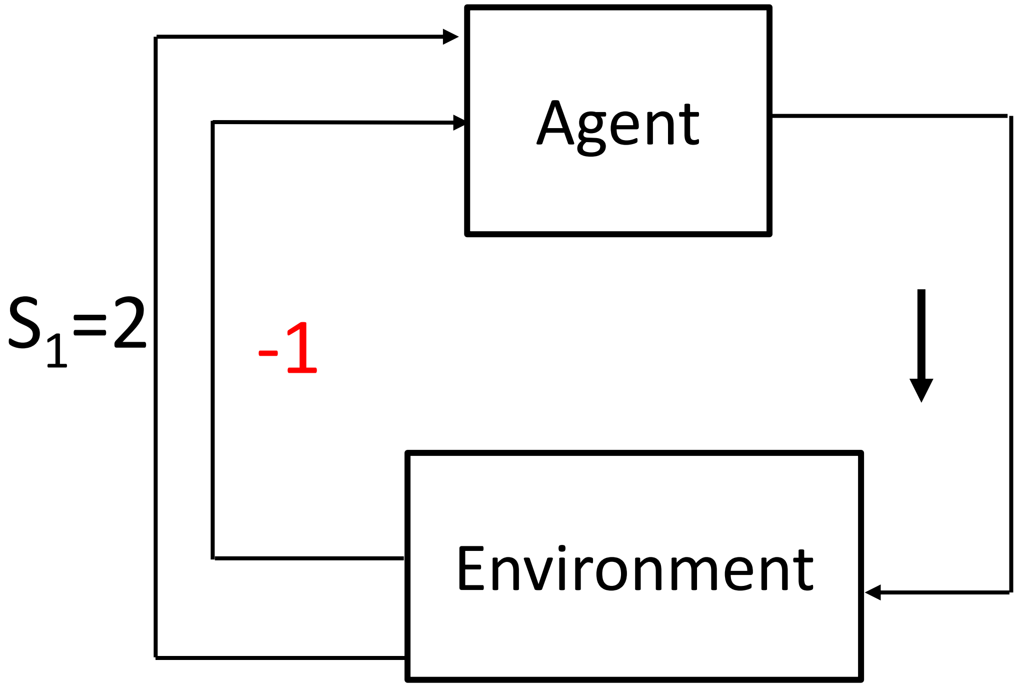
Example Robot



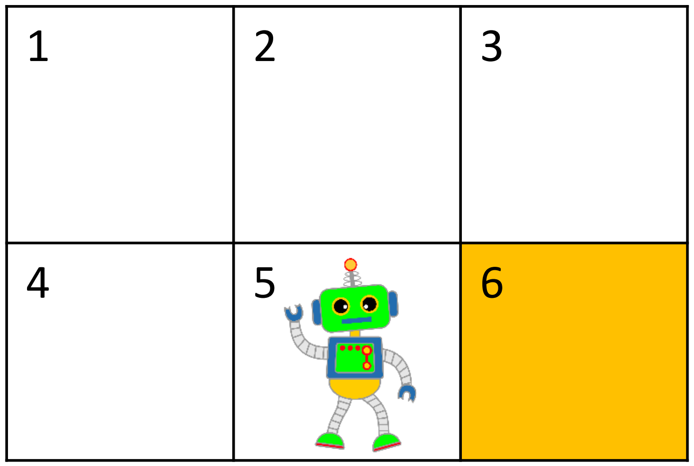


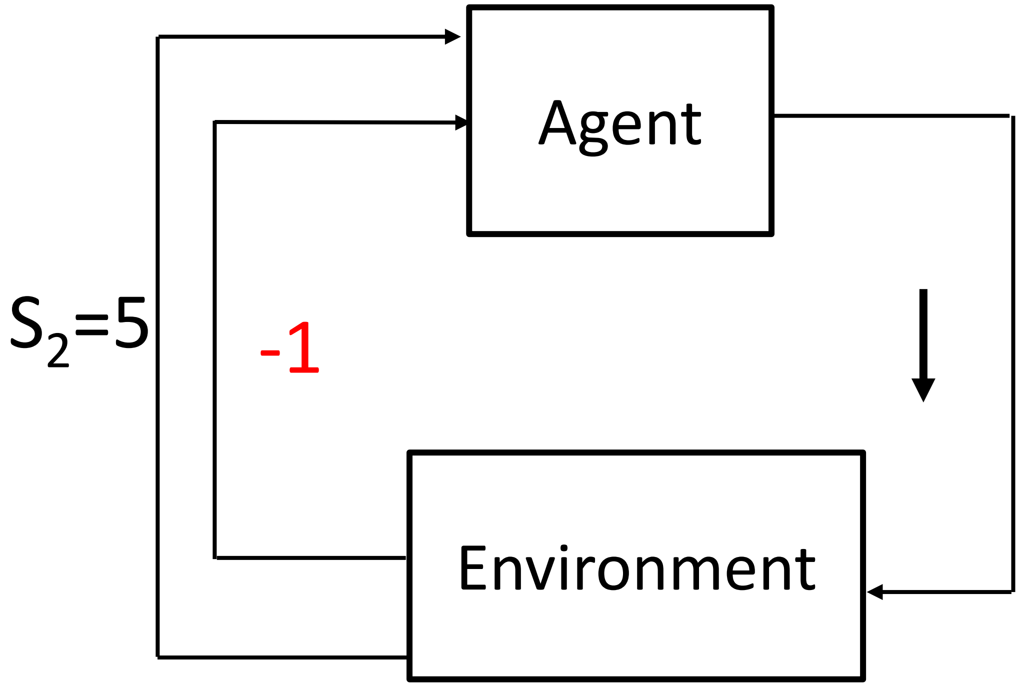
Example Robot



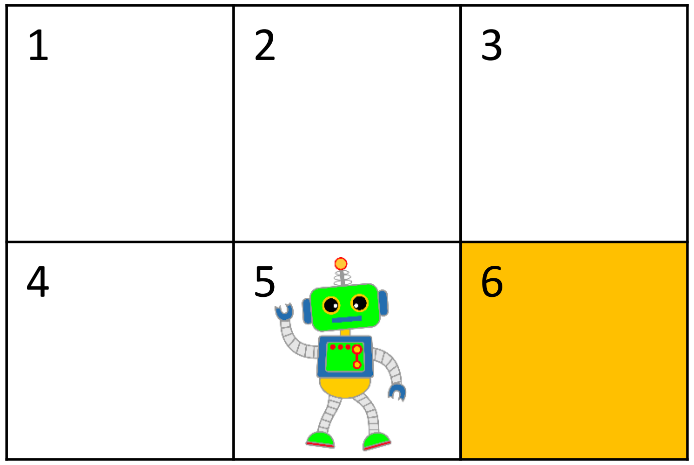


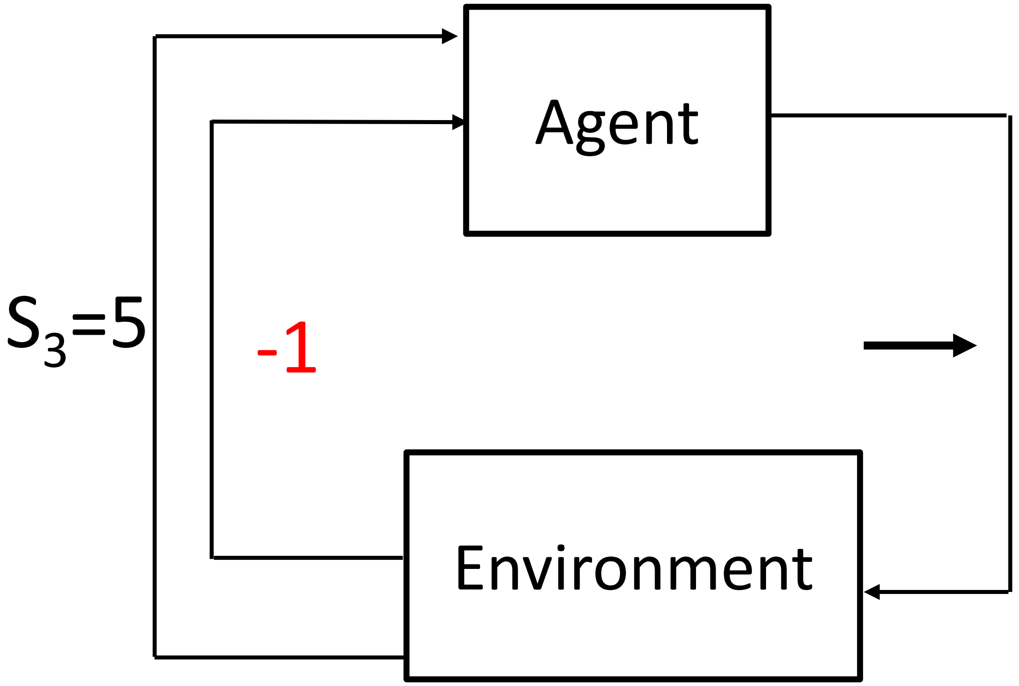
Example Robot



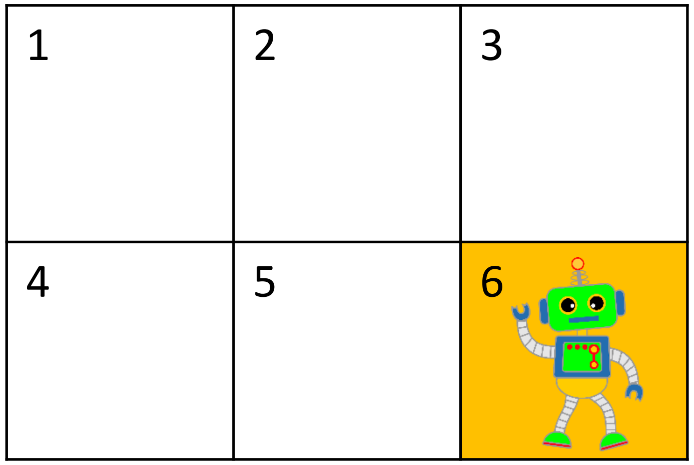


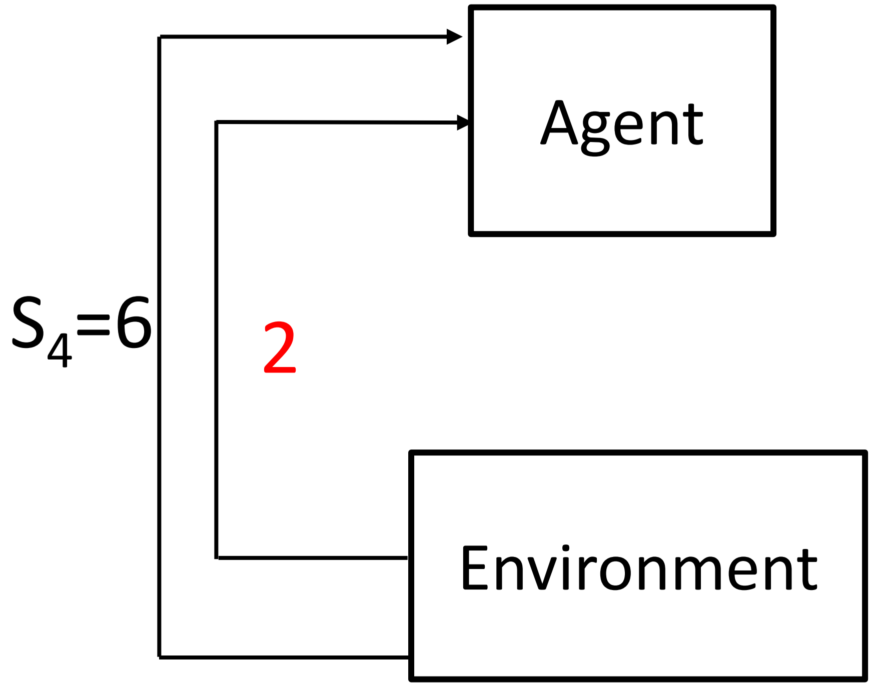
Example Robot



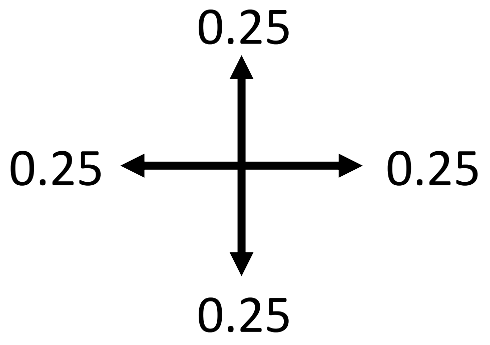


Example Robot





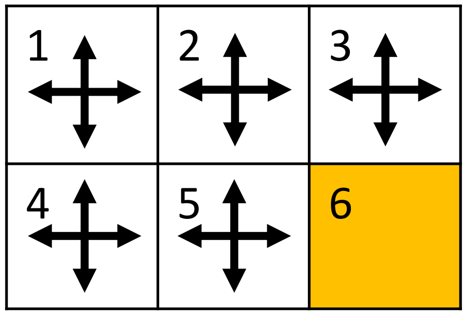
Policy

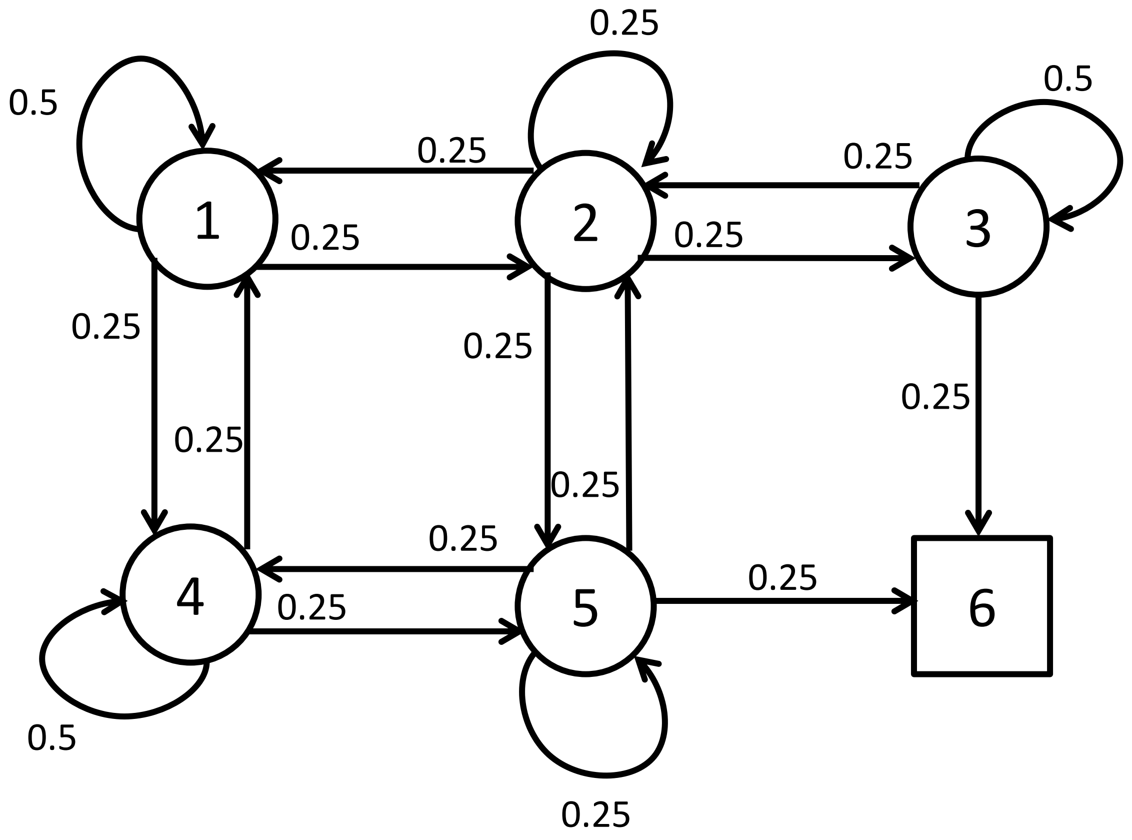


$$\pi_t(up|s_i) = 0.25$$
$$\pi_t(left|s_i) = 0.25$$
$$\pi_t(down|s_i) = 0.25$$
$$\pi_t(right|s_i) = 0.25$$

- In each state, the agent can choose between different actions. The probability that the agent selects a possible action is called policy.
- $\pi_t(a|s)$: probability that $A_t = a$ if $S_t = s$
- In reinforcement learning: the agent changes the policy as a result of the experience

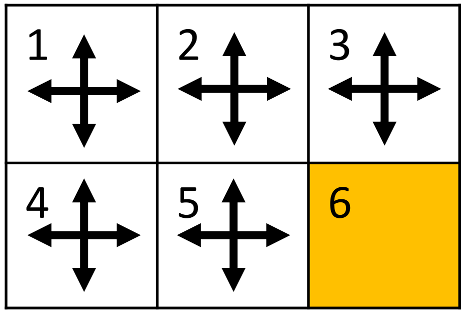
Example Robot: Diagram

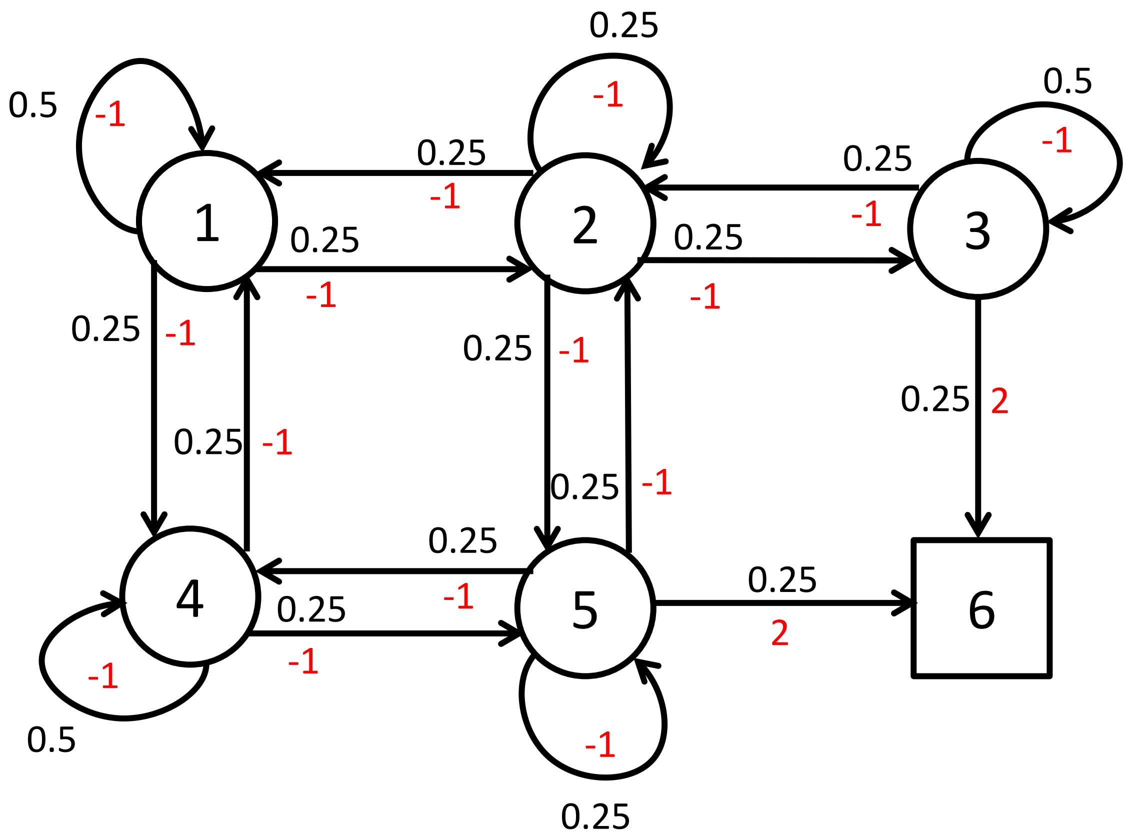




Reward signal

- Goal: Maximizing the total amount of cumulative reward over the long run



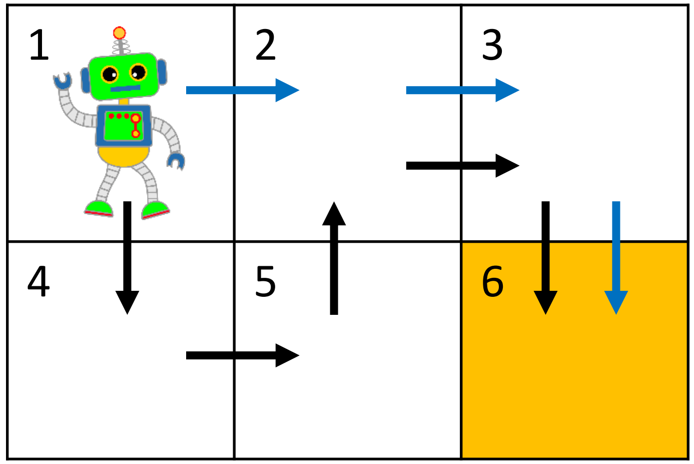


Return

Sum of the rewards

- $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$, where T is a final step

Maximize the expected return



t=0

G_0=-1-1-1-1+2=-2

G_0=-1-1+2=0

Discounting

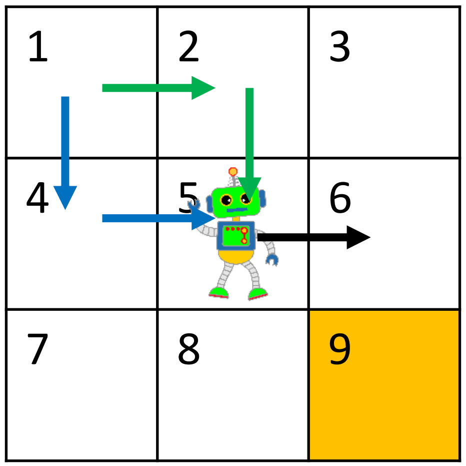
- If the task is a continuing task, a discount rate for the return is needed

Discount rate determines the present value of the future rewards in a continuing task

- $G_t = R_{t+1} + \gamma * R_{t+2} + \gamma^2 * R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ where γ is called the discount rate: $0 \leq \gamma \leq 1$

Unified Notation: $\boldsymbol{G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1}}$

The Markov Property



- $Pr\{R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_1, \dots, S_{t-1}, A_{t-1}, R_1, S_t, A_t$
  $Pr\{R_{t+1} = r, S_{t+1} = s' | S_t, A_t\}$
- State signal summarizes past sensations compactly such that all relevant information is retained
- Decisions are assumed to be a function of the current state only

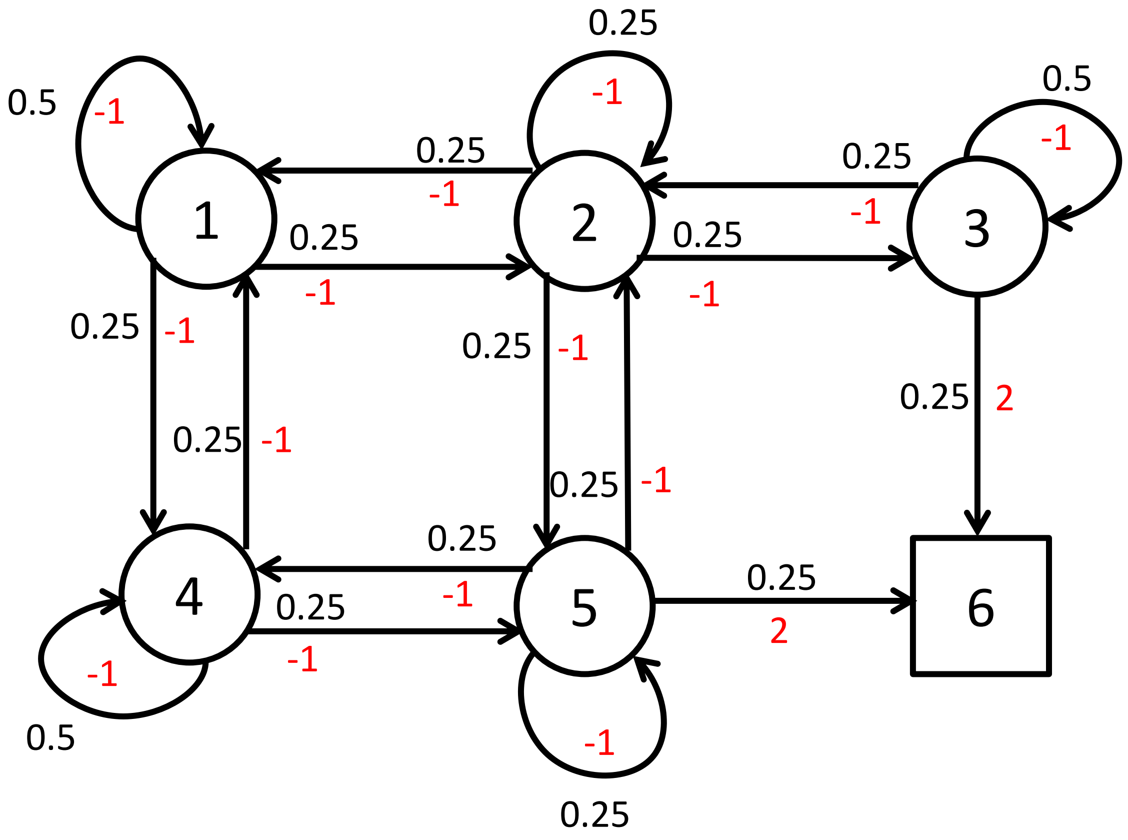
The Markov Decision Processes

Task has to satisfy the Markov Property

- If the state and action spaces are finite, then it is called a finite Markov decision process
- Given any state and action, s and a, the probability of each possible next state and reward, s’, r, is: $p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$

Example robot

$$p(s', r | s, a) = Pr\{S_{t+1} = s', R_{t+1} | S_t = s, A_t = a\}$$



$p(2, -1 | 1, right) = 1$

$p(4, -1 | 1, down) = 1$

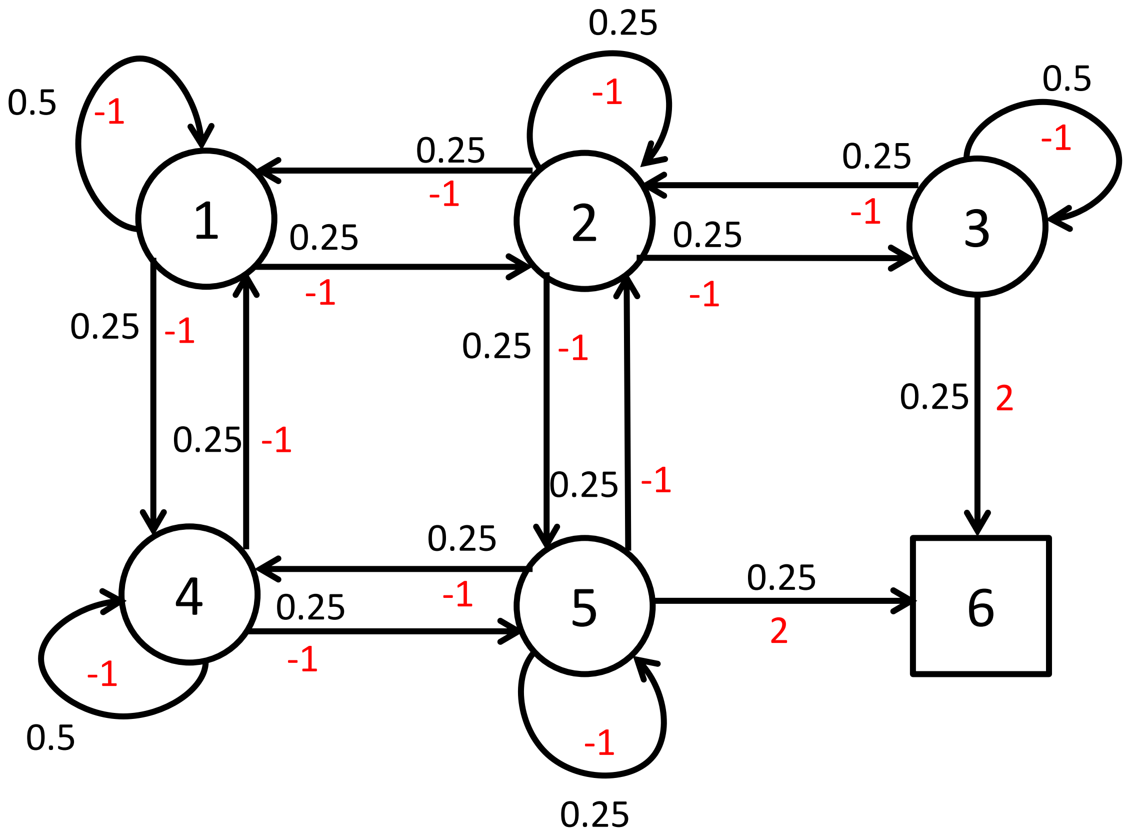
$p(4, -1 | 1, up) = 0$

The Markov Decision Processes

- Given any current state and action, s and a, together with any next state, s', the expected value of next reward is:
  $r(s,a,s') = E[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s']$

Example robot

$$r(s,a,s') = E[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s']$$



$$r(1, right, 2) = -1$$

$$r(1, down, 4) = -1$$

$$r(5, right, 6) = 2$$

Value functions

- Value functions estimate how good it is for the agent to be in a given state (state-value function) or how good it is to perform a certain action in a given state (action-value function)
- Value functions are defined with respect to particular policies
- The value of a state s under a policy π is the expected return when starting in s and following π thereafter: $v_\pi(s) = E_\pi[G_t | S_t = s]$

- v_π is called the **state-value function** for policy π

State-value function

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \Big[r + \gamma v_\pi(s')\Big]
\end{aligned}$$

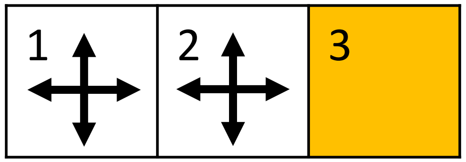
Property of state-value function

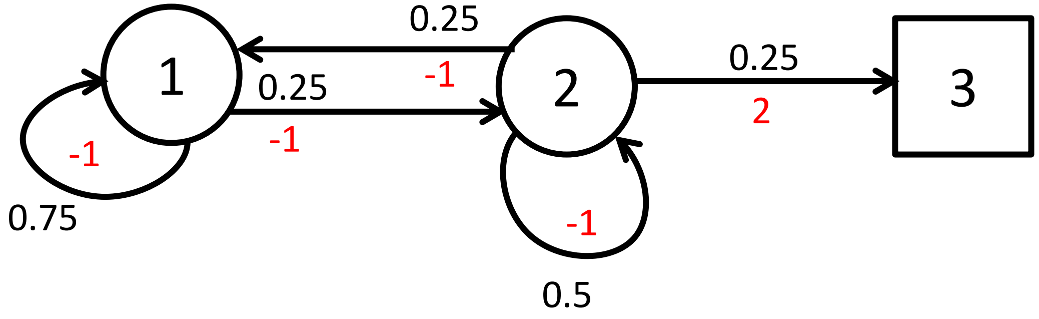
$$v_\pi(s) = E_\pi[G_t | S_t = s] = \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a)[r + \gamma v_\pi(s')]$$

- Bellman equation for v_π
- Expresses a relationship between the value of a state and the value of its successor states

Example state-value function

$$v_\pi(s) = E_\pi[G_t|S_t = s] = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$





$\gamma = 1$

$v_\pi(3) = 0$

$v_\pi(1) = 3 * (0.25 * 1 * (-1 + v_\pi(1))) + 0.25 * 1 * (-1 + v_\pi(2))$

$v_\pi(2) = 2 * (0.25 * 1 * (-1 + v_\pi(2))) + 0.25 * 1 * (-1 + v_\pi(1)) + 0.25 * 1 * (2 + v_\pi(3))$

$\boldsymbol{v_\pi(1) = -9}$ $\boldsymbol{v_\pi(2) = -5}$ $\boldsymbol{v_\pi(3) = 0}$

Action-value function

- The value of the expected return taking action a in state s under policy π
- $q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$
- q_π is called the **action-value function** for policy π

Optimal policy

A policy π is better or equal to a policy π' if the state-value function is greater or equal to that of π'

- $\pi \geq \pi' if\ and\ only\ if\ v_\pi(s) \geq v_{\pi'}(s)\ for\ all\ s \in S$

Optimal state-value function

- $v_*(s) = \max_\pi v_\pi(s), for\ all\ s \in S$

Optimal action-value function

- $q_*(s, a) = \max_\pi q_\pi(s, a), for\ all\ s \in S\ \ and\ a \in A(s)$

Bellman optimality equation

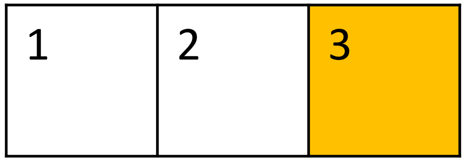
- Without a reference to any specific policy

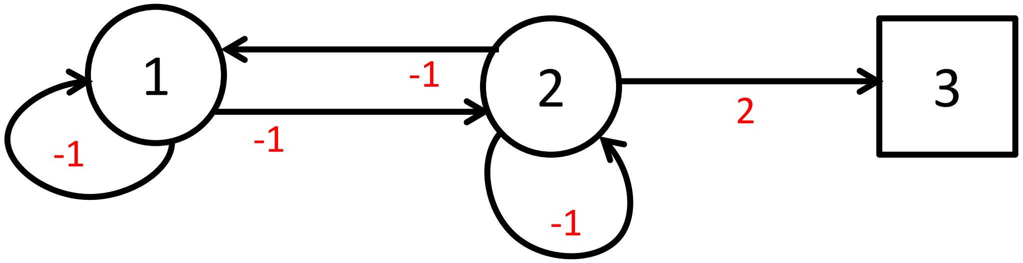
Bellman optimality equation for $\mathbf{v}_*$

- $v_*(s) = \max\limits_{a \in A(s)} \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$

Bellman optimality equation for v$_*$

$$v_*(s) = \max_{a \in A(s)} \sum_{s',r} p(s', r | s, a)[r + \gamma v_*(s')]$$





$\underline{\gamma = 1}$

$v_*(3) = 0$

$$v_*(1) = \max \begin{Bmatrix} 1 * (-1 + v_*(1)) \\ 1 * (-1 + v_*(1)) \\ 1 * (-1 + v_*(1)) \\ 1 * (-1 + v_*(2)) \end{Bmatrix}$$

<u>actions:</u>

up
down
left
right

$$\boldsymbol{v_*(1) = ?}$$

$$v_*(2) = \max \begin{Bmatrix} 1 * (-1 + v_*(2)) \\ 1 * (-1 + v_*(2)) \\ 1 * (-1 + v_*(1)) \\ 1 * (2 + v_*(3)) \end{Bmatrix}$$

up
down
left
right

$$\boldsymbol{v_*(2) = ?}$$

Bellman optimality equation

Bellman optimality equation for $\mathbf{q}_*$

- $q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$

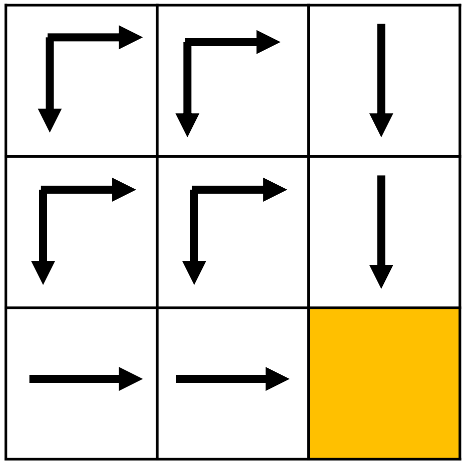
Bellman optimality equation

- System of nonlinear equations, one for each state
- N states: there are N equations and N unknowns
- If we know $p(s',r|s,a)$ and $r(s,a,s')$ then in principle one can solve this system of equations
- If we have v_* it is relatively easy to determine an optimal policy

v_*

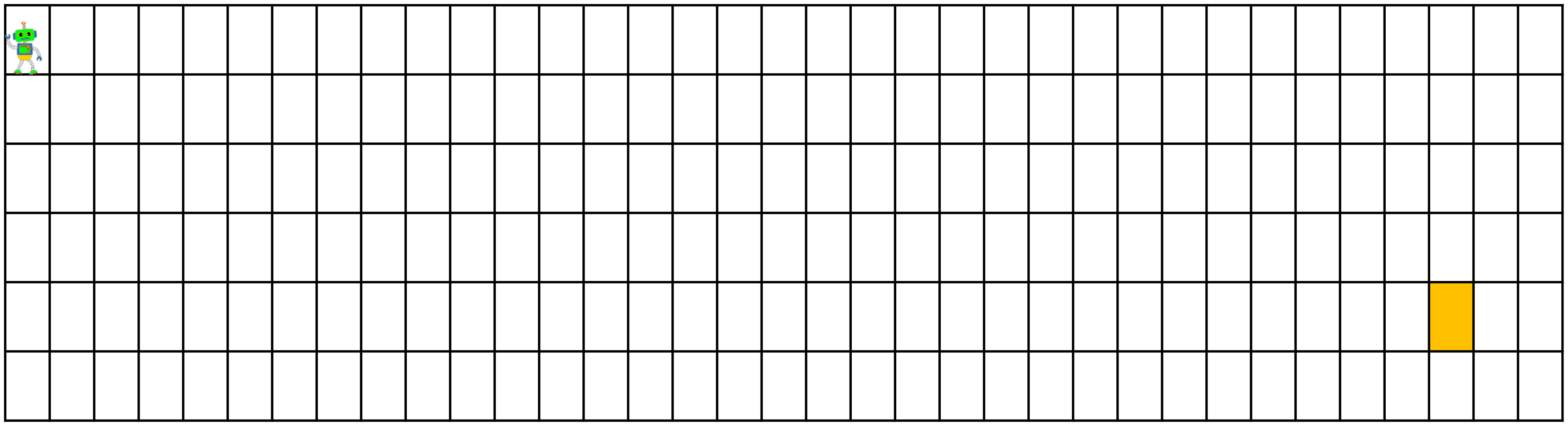
-9	-5	-3
-5	-3	-2
-3	-2	0

π_*



Assumptions for solving the Bellman optimality equation

- Markov property
- We know the dynamics of the environment
- We have enough computational resources to complete the computation of the solution
- Problem: Long computational time
- Solution: Dynamic programming



Dynamic Programming

Dynamic Programming

Collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process

Problem of classic DP algorithms: They are only of limited utility in reinforcement learning:

- Assumption of perfect model
- Great computational expense

Key Idea of Dynamic Programming

Goal: Find optimal policy

Problem: Solve the Bellman optimality equation

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$$

Solution methods:

- Direct search
- Linear programming
- Dynamic programming

Key Idea of Dynamic Programming

Key idea of DP (and of reinforcement learning in general):

Use of value functions to organize and structure the search for good policies

Dynamic programming approach:

Introduce two concepts:

- Policy evaluation
- Policy improvement

Use those concepts to get an optimal policy

Assumptions

We always assume that the environment is a finite MDP, i.e:

- State, action and reward sets $\mathcal{S}$, $\mathcal{A}(s)$ and $\mathcal{R}$, for $s \in \mathcal{S}$, are finite
- Dynamics are given by a set of probabilities $p(s', r|s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s), r \in \mathcal{R}$, and $s' \in S^{+}$ (S^{+} is $\mathcal{S}$ plus a terminal state if the problem is episodic)

Policy Evaluation

How to compute state-value function v_π for an arbitrary policy π:

Recall Bellman equation:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

Existence and uniqueness of v_π are guaranteed if:

- Either $\gamma < 1$ or
- Eventual termination is guaranteed from all states under policy π

Iterative Policy Evaluation

Consider iterative solution methods for Bellman equation:

Consider a sequence of approximate value functions $v_0, v_1, v_2, \dots$, each mapping S^+ to $\mathbb{R}$.

Initial approximation, v_0, is chosen arbitrarily (except that the terminal states, if any, must be given value 0) .

Subsequently, use the Bellman equation for v_π as an update rule:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

for all $s \in \mathcal{S}$.

Iterative Policy Evaluation

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

Convergence:

One can show that the sequence $\{v_k\}$ converges to v_π as $k \to \infty$ under the same conditions that guarantee the existence of v_π, i.e.

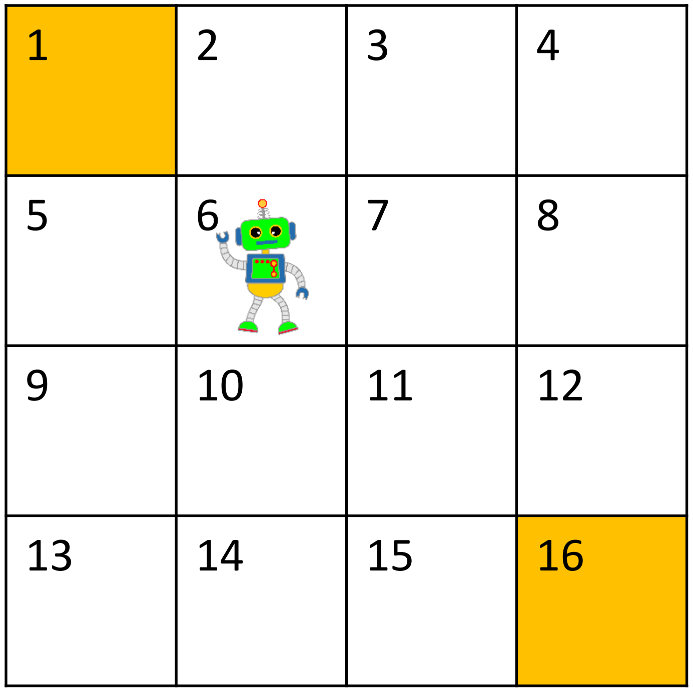
- Either $\gamma < 1$ or
- Eventual termination is guaranteed from all states under policy π

Example: Iterative Policy Evaluation

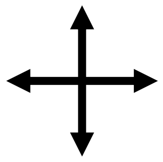
Consider the robot example:

Goal: reach top left or bottom right corner

$\rightarrow$(Nonterminal states are S = {2, 3, …, 15})



actions



Reward is -1 for all transition

Example: Iterative Policy Evaluation

Recall: can choose initial approximation arbitrarily (except for terminal state)

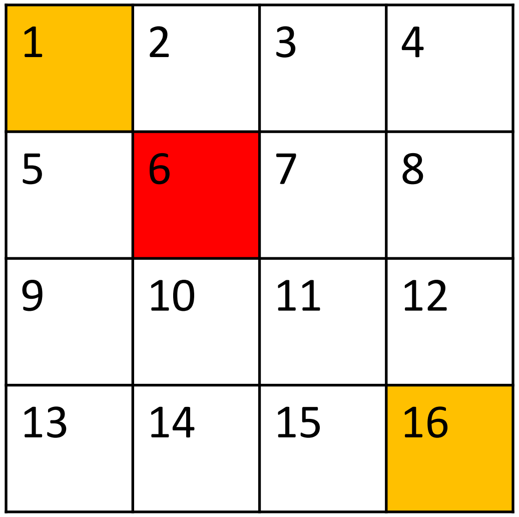
→ choose $v_0(s) = 0$ for all states $s \in S^+ = \{1,2,\ldots,16\}$

v_0 for the random policy:

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Example: Iterative Policy Evaluation

Let’s calculate v_1:



Example: Iterative Policy Evaluation

Let's calculate v_1: $s = 6$

$$v_1(6) = \sum_{a \in \{u,d,l,r\}} \pi(a|6) \sum_{s',r} p(s',r|6,a)[r + \gamma v_0(s')]$$

$$= \sum_{a \in \{u,d,l,r\}} \underbrace{\pi(a|6)}_{= 0.25\ \forall a} \sum_{s'} p(s'|6,a)[\underbrace{r}_{= -1} + \gamma \underbrace{v_0(s')}_{= 0\ \forall s'}]$$

$$= 0.25 * \{-p(2|6,u) - p(10|6,d) - p(5|6,l) - p(7|6,r)\}$$

$$= 0.25 * \{-1 - 1 - 1 - 1\}$$

$$= -1$$

$$\Rightarrow v_1(6) = -1$$

Analogously for all non-terminal states $s \in S$: $v_1(s) = -1$

Example: Iterative Policy Evaluation

Let's calculate v_1:

For the terminal states 1 and 16 the process terminates, i.e. for $s \in \{1, 16\}$:

$$p(s'|s,a) = 0 \; \forall s' \in S, a \in \{u, d, l, r\}$$

$$\boxed{\Rightarrow v_k(1), v_k(16) = 0 \;\; \forall k}$$

$\Rightarrow v_1$ for the random policy:

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

Example: Iterative Policy Evaluation

Let’s calculate v_2:

$s = 6$:

$$v_2(6) = \sum_{a \in \{u,d,l,r\}} \underbrace{\pi(a|6)}_{= 0.25\ \forall a} \sum_{s'} p(s'|6,a)[\underbrace{r}_{= -1} + \gamma \underbrace{v_1(s')}_{= \begin{cases} -1, s' \in S \\ 0, s' \in S^+ \backslash S \end{cases}}]$$

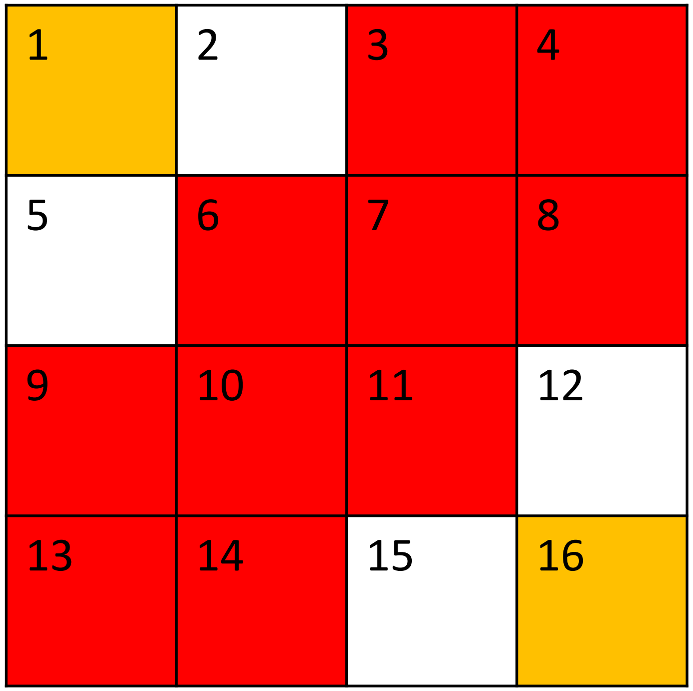
$$= 0.25 * \{p(2|6,u)[-1-\gamma] + p(10|6,d)[-1-\gamma] + p(5|6,l)[-1-\gamma] + p(7|6,r)[-1-\gamma]\}$$

$$\overset{\gamma = 1}{=} 0.25 * \{-2-2-2-2\}$$

$$= -2$$

$$\Rightarrow v_2(6) = -2$$

Example: Iterative Policy Evaluation

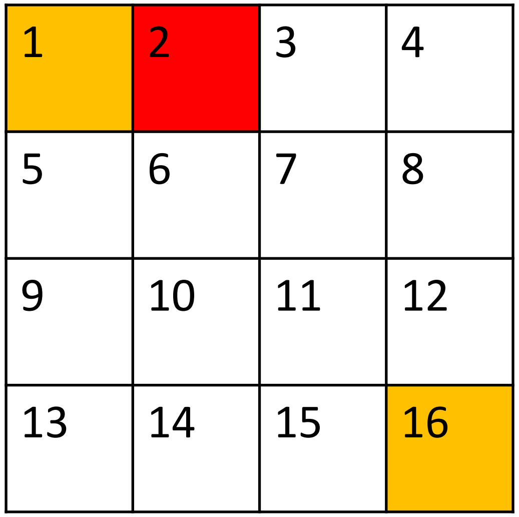


Analogously, we get for all red colored states s

$$\boldsymbol{v_2(s) = -2}$$

Example: Iterative Policy Evaluation

Let’s calculate v_2:



Example: Iterative Policy Evaluation

Let's calculate v_2:

$s = 2$:

$$v_2(2) = \sum_{a \in \{u,d,l,r\}} \underbrace{\pi(a|2)}_{= 0.25\ \forall a} \sum_{s'} p(s'|2,a)[\underbrace{r}_{= -1} + \gamma \underbrace{v_1(s')}_{= \begin{cases} -1, s' \in S \\ 0, s' \in S^+ \backslash S \end{cases}}]$$

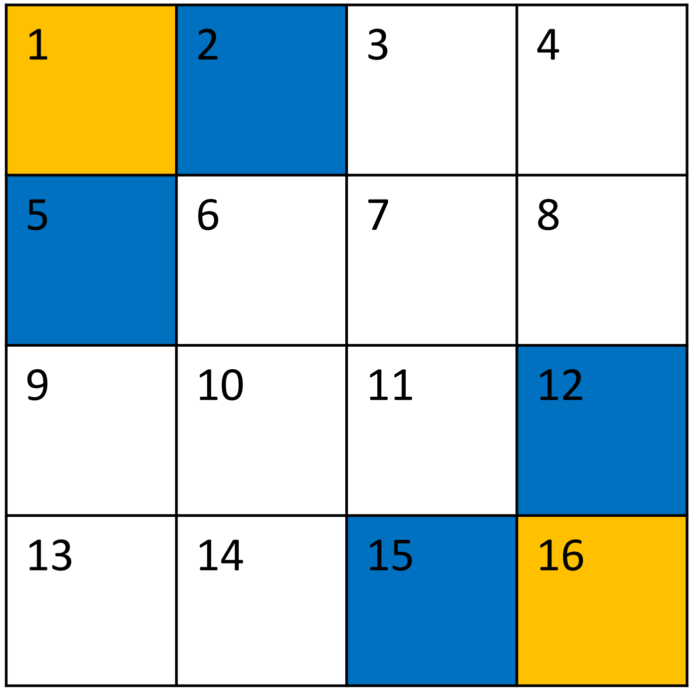
$$= 0.25 * \{p(2|2,u)[-1-\gamma] + p(6|2,d)[-1-\gamma] + p(1|2,l)[-1-\gamma*0] + p(3|2,r)[-1-\gamma]\}$$

$$\overset{\gamma = 1}{=} 0.25 * \{-2-2-1-2\}$$

$$= -1.75$$

$$\Rightarrow v_2(2) = -1.75$$

Example: Iterative Policy Evaluation



Analogously, we get for all blue colored states s

$$\boldsymbol{v_2(s) = -1.75}$$

Example: Iterative Policy Evaluation

$\Rightarrow v_2$ for the random policy:

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

Example: Iterative Policy Evaluation

v_k for the random policy:

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k =3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$\cdots$

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$\cdots$

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$\leftarrow v_\pi$

Policy Evaluation

Reason for computing value function v_π for a policy π:

$\rightarrow$ Finding better policies

$\rightarrow$ **Policy improvement**

Policy Improvement

- Suppose we have determined the value function v_π for an arbitrary deterministic policy π
- Should we change the policy to deterministically choose an action $a \neq \pi(s)$ for some state s?
- What we know: how good it is to follow the current policy from $s: v_\pi(s)$
- What we want to know: would it be better or worse to change to the new policy?

Policy Improvement

Would it be better or worse to change to the new policy ?

(new policy: for some s choose action $a \neq \pi(s)$)

→ Consider selecting a in s and thereafter following the existing policy π: value of this way of behaving is:

$$q_\pi(s,a) = E_\pi[\, R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s', r \mid s, a)\, [r + \gamma v_\pi(s')]$$

→ If this is greater than $v_\pi(s)$, i.e., if it is better to select a once in s and thereafter follow π than it would be to follow π all the time, then we would expect the new policy to be better than π

Policy Improvement Theorem

Let π and π' be any pair of deterministic policies such that, for all $s \in S$,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{1}$$

Then the policy π' must be as good as, or better than, π. That is, it must obtain greater or equal expected return from all states $s \in S$:

$$v_{\pi'}(s) \geq v_\pi(s). \tag{2}$$

Moreover, if there is strict inequality of (1) at any state then there must be strict inequality of (2) at at least one state.

Policy Improvement

For situation before:

- Suppose we have a deterministic policy π, and a new policy π' that equals π except for one state s for which $\pi'(s) = a \neq \pi(s)$
- Suppose $q_\pi(s,a) \geq v_\pi(s)$, i.e. (1) is satisfied

$\xRightarrow[\substack{policy\\ improv.\\ thm.}]{}$ π' is as good as, or better than, π

Policy Improvement Theorem: Proof

Claim: $q_\pi\big(s, \pi'(s)\big) \geq v_\pi(s) \qquad (1)$

$$\Rightarrow v_{\pi'}(s) \geq v_\pi(s)$$

Proof: $$\begin{aligned}
v_\pi(s) &\overset{(1)}{\leq} q_\pi\big(s, \pi'(s)\big) \\
&= \boldsymbol{E}_\pi[R_{t+1} + \gamma\, v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\
&= \boldsymbol{E}_{\pi'}[R_{t+1} + \gamma\, v_\pi(S_{t+1}) | S_t = s] \\
&\overset{(1)}{\leq} \boldsymbol{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\
&= \boldsymbol{E}_{\pi'}[R_{t+1} + \gamma\, \boldsymbol{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2})] | S_t = s] \\
&= \boldsymbol{E}_{\pi'}[R_{t+1} + \gamma\, R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s] \\
&\overset{(1)}{\leq} \boldsymbol{E}_{\pi'}[R_{t+1} + \gamma\, R_{t+2} + \gamma^2 q_\pi\, (S_{t+2}, \pi'(S_{t+2})) | S_t = s] \\
&= \boldsymbol{E}_{\pi'}[R_{t+1} + \gamma\, R_{t+2} + \gamma^2 \boldsymbol{E}_{\pi'}[R_{t+3} + \gamma v_\pi(S_{t+3})] | S_t = s] \\
&= \boldsymbol{E}_{\pi'}[R_{t+1} + \gamma\, R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) | S_t = s] \\
&\vdots \\
&= \boldsymbol{E}_{\pi'}[\overbrace{R_{t+1} + \gamma\, R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots}^{= G_t} | S_t = s] \\
&= \boldsymbol{E}_{\pi'}\ [G_t | S_t = s] \\
&= v_{\pi'}(s)
\end{aligned}$$

■

Policy Improvement

- What we have seen: Given a (deterministic) policy and its value function we can easily evaluate a change in the policy at a **single** state
- What if we allow changes at **all** states?

  →For a given (deterministic) policy π select at each state $s \in S$ the action that appears best according to $q_\pi(s,a)$

  → i.e., consider the new greedy policy π', given by

$$\pi'(s) = \underset{a}{\operatorname{argmax}}\, q_\pi(s,a) \tag{3}$$

  → take action that looks best in the short term – after one step of lookahead – according to v_π

Policy Improvement

By construction, the greedy policy π' fulfills the condition

$$q_\pi(s, \pi'(s)) \geq \; v_\pi(s) \qquad (1)$$

$\xRightarrow{\substack{\text{policy}\\ \text{impr.}\\ \text{theorem}}}$ the policy π' is as good as, or better than, the original policy

Policy Improvement

The process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy, is called **policy improvement.**

Policy Improvement

Suppose the new greedy policy, π', is as good as, but not better than, the old policy π.

Then $v_\pi = v_{\pi'}$, and from

$$\pi'(s) = \underset{a}{\operatorname{argmax}}\, q_\pi(s, a) \tag{3}$$

it follows that for all $s \in S$:

$$\begin{aligned} v_{\pi'}(s) &= \max_a E[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_{\pi'}(s)]. \end{aligned}$$

This is the Bellman optimality equation, and therefore, $v_{\pi'}$ must be v_*, and both π and π' must be optimal policies.

Policy Improvement

- All the ideas of policy improvement can be extended to stochastic policies.

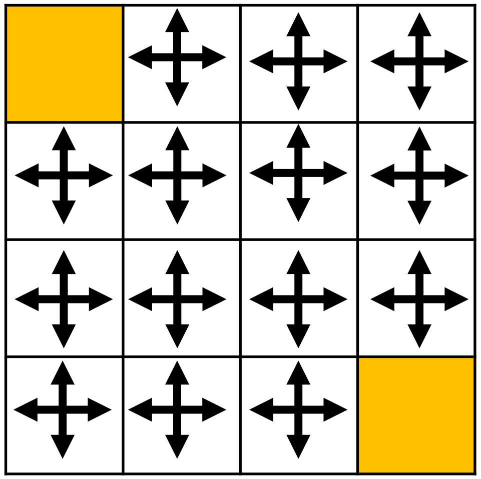
  (A stochastic policy π specifies probabilities $\pi(a|s)$ for taking each action a in each state s.)

- In particular, the policy improvement theorem holds also for stochastic policies, under the natural definition:

$$q_\pi(s, \pi'(s)) = \sum_a \pi'(a|s) q_\pi(s, a).$$

Example: Policy Improvement

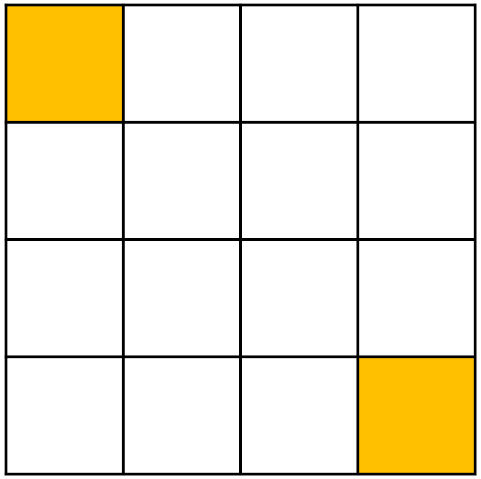
random policy π



value function v_π

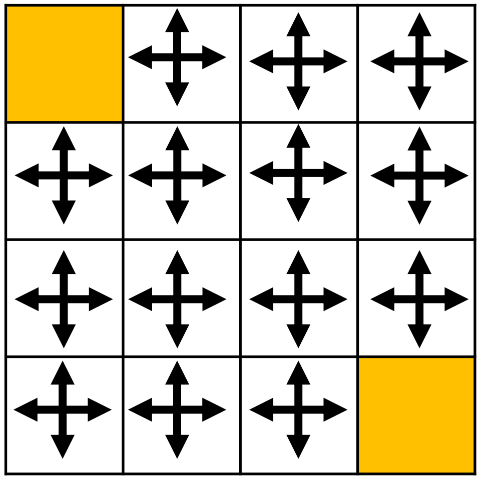
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$\xrightarrow{\text{policy improvement}}$



Example: Policy Improvement

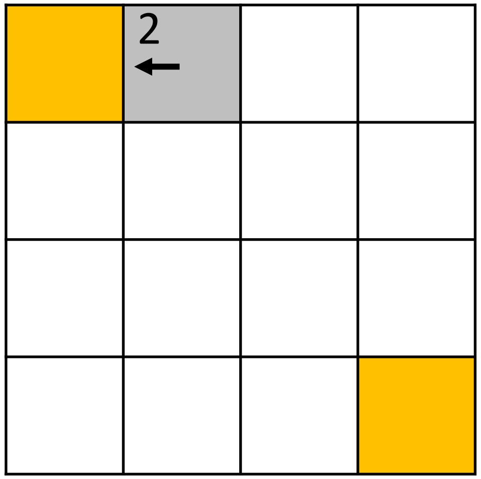
random policy π



value function v_π

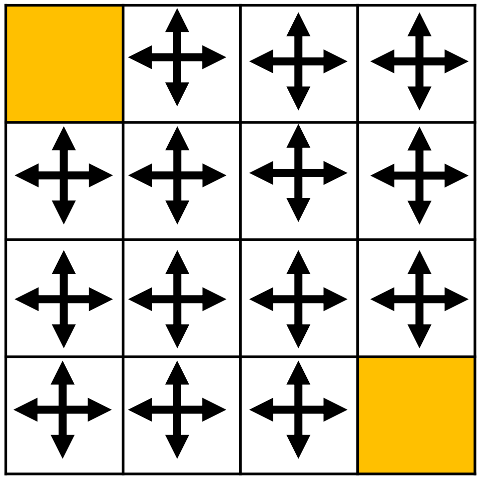
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

policy improvement $\longrightarrow$



Example: Policy Improvement

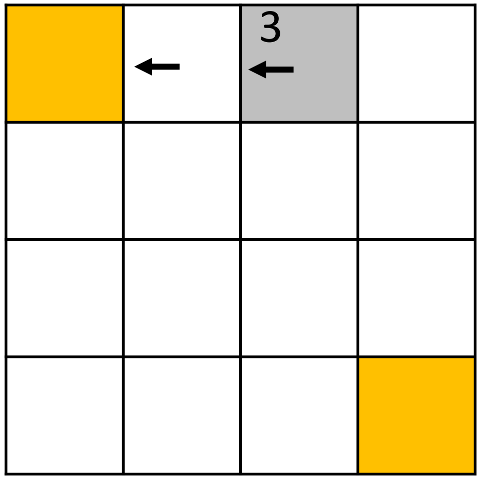
random policy π



value function v_π

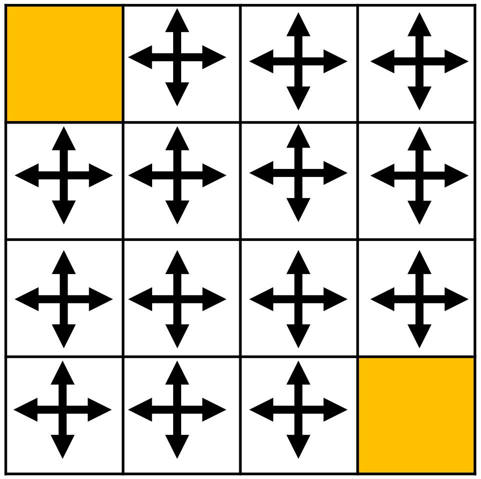
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

policy improvement $\longrightarrow$



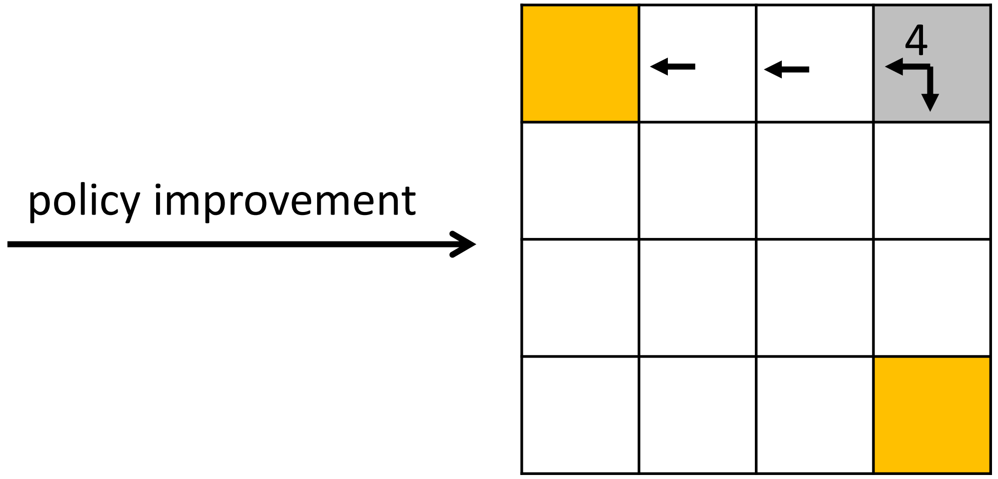
Example: Policy Improvement

random policy π



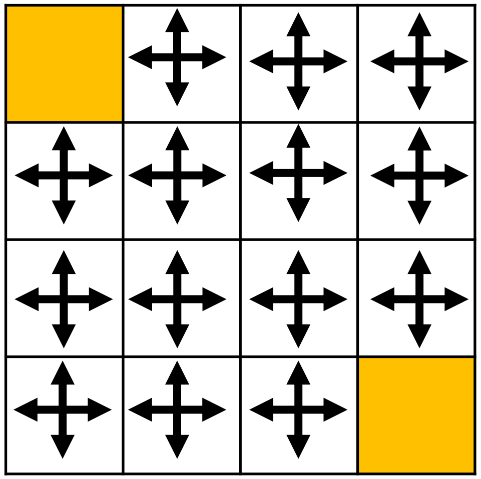
value function v_π

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Example: Policy Improvement

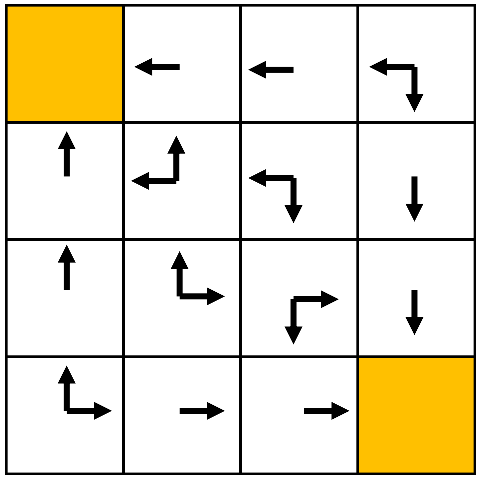
random policy π



value function v_π

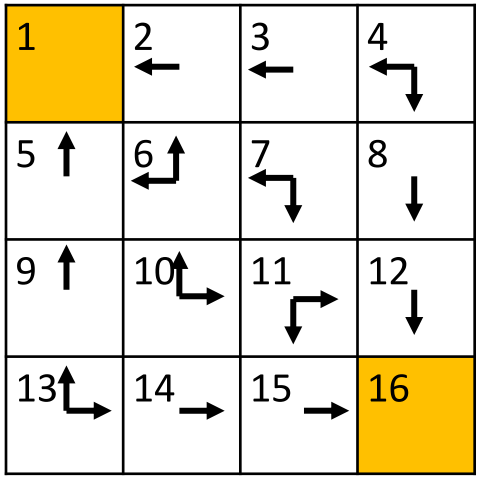
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

policy improvement $\longrightarrow$



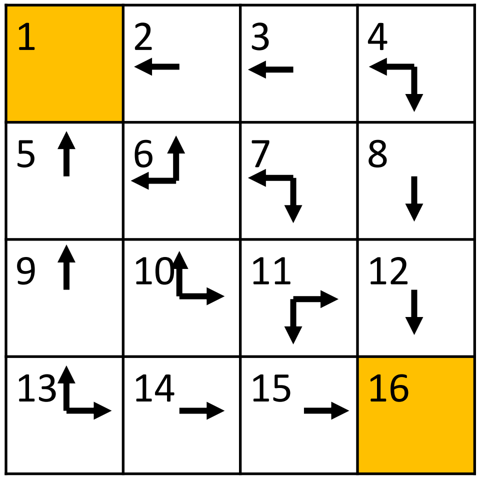
Example: Policy Improvement

New policy π':



Example: Policy Improvement

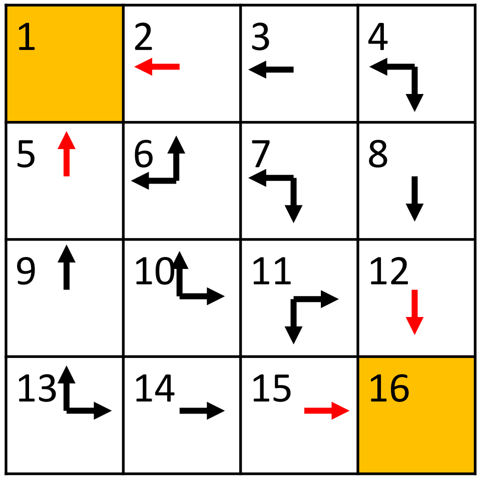
New policy π':



Is π' a better policy than the random policy π?

Example: Policy Improvement

New policy π':



Is π' a better policy than the random policy π?

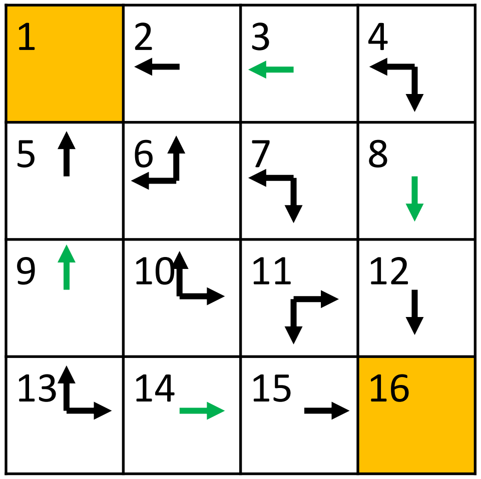
Let's calculate $v_{\pi'}$:

$$\color{red}{v_{\pi'}(2)} = \sum_{a\in\{l\}} \underbrace{\pi'(a|2)}_{=1} \sum_{s'} \underbrace{p(s'|2,a)}_{=\begin{cases}0\ for\ s' \in \{2,3,\dots,16\}\\ 1\ for\ s' = 1\end{cases}} [-1 + v_{\pi'}(s')]$$

$$= \pi'(l|2) \textstyle\sum_{s'} p(s'|2,l)[-1 + v_{\pi'}(s')]$$
$$= -1 + v_{\pi'}(1)$$
$$\color{red}{= -1}$$

Example: Policy Improvement

New policy π':



Is π' a better policy than the random policy π?

Let's calculate $v_{\pi'}$:

$$v_{\pi'}(3) = \sum_{a\in\{l\}} \underbrace{\pi'(a|3)}_{=1} \sum_{s'} \underbrace{p(s'|3,a)}_{=\begin{cases}0 \; for\; s' \in \{1,3,4 \dots, 16\} \\ 1\; for\; s' = 2\end{cases}} [-1 + v_{\pi'}(s')]$$

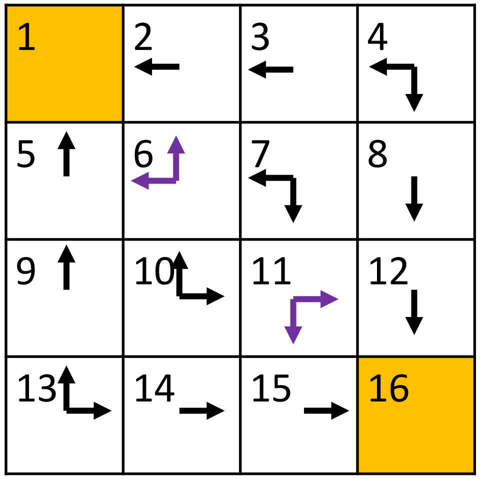
$$= -1 + v_{\pi'}(2)$$

$$= -1 - 1$$

$$= -2$$

Example: Policy Improvement

New policy π':



Is π' a better policy than the random policy π?

Let's calculate $v_{\pi'}$:

$$v_{\pi'}(6) = \sum_{a\in\{l,u\}} \underbrace{\pi'(a|6)}_{=\,0.5} \sum_{s\prime} \underbrace{p(s'|6,a)}_{=\begin{cases}0 \; for \; s' \in S\{2,5\} \\ 1 \; for \; s' \in \{2,5\}\end{cases}} [-1 + v_{\pi'}(s')]$$

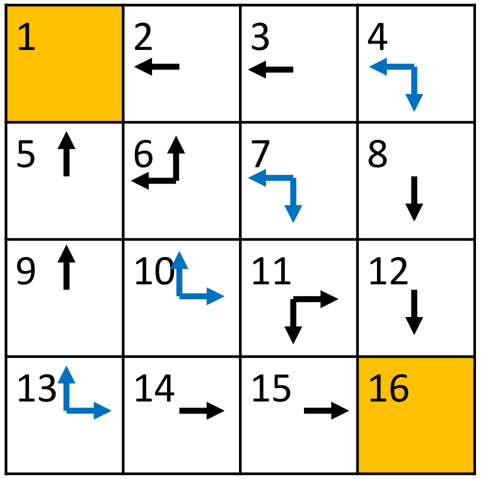
$$= 0.5 * \{p(5|6,l) * [-1 + \overbrace{v_{\pi'}(5)]}^{=\,-1}$$
$$+p(2|6,u) * [-1 + \underbrace{v_{\pi'}(2)}_{=\,-1}]\}$$

$$= 0.5 * \{-2 - 2\}$$

$$= -2$$

Example: Policy Improvement

New policy π':



Is π' a better policy than the random policy π?

Let's calculate $v_{\pi'}$:

$$v_{\pi'}(4) = \sum_{a \in \{l,d\}} \underbrace{\pi'(a|4)}_{=0.5} \sum_{s'} p(s'|4,a)[-1 + v_{\pi'}(s')]$$

$$= 0.5 * \{p(3|4,l) * [-1 + \overbrace{v_{\pi'}(3)}^{=-2}]$$

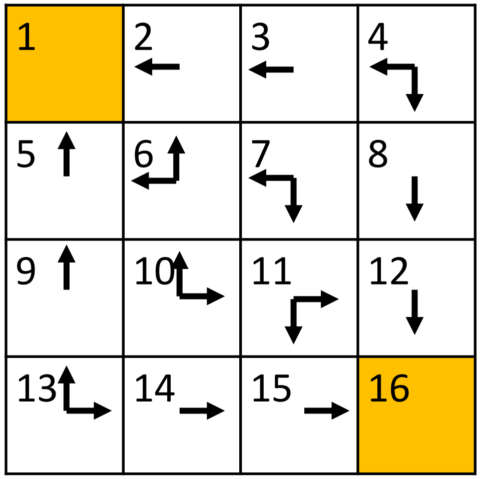
$$+p(8|4,d) * [-1 + \underbrace{v_{\pi'}(8)}_{=-2}]\}$$

$$= 0.5 * \{-3 - 3\}$$

$$= -3$$

Example: Policy Improvement

New policy π':



Value function $v_{\pi'}$:

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

Value function v_{π}:

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$$\left.\begin{array}{l}\text{Since } v_\pi(s) \leq -14 \text{ for all non-terminal states,}\\ \text{and } v_{\pi'}(s) \geq -3 \text{ for all non-terminal states}\end{array}\right\} \quad \text{Clearly } v_{\pi'}(s) \geq v_\pi(s) \quad \forall\, s \in S$$

$\Rightarrow$ π' is better than π

Policy Iteration

Policy iteration is a way of finding an optimal policy:

Once a policy π has been improved using v_π to yield a better policy π' we can then compute $v_{\pi'}$, and improve it again to yield an even better policy π''

Thus, we can obtain a sequence of monotonically improving policies and value functions:

$$\pi_0 \overset{E}{\rightarrow} v_{\pi_0} \overset{I}{\rightarrow} \pi_1 \overset{E}{\rightarrow} v_{\pi_1} \overset{I}{\rightarrow} \pi_2 \overset{E}{\rightarrow} \ldots \overset{I}{\rightarrow} \pi_* \overset{E}{\rightarrow} v_*$$

$\overset{E}{\rightarrow}$ denotes a policy evaluation and

$\overset{I}{\rightarrow}$ denotes a policy improvement

Policy Iteration

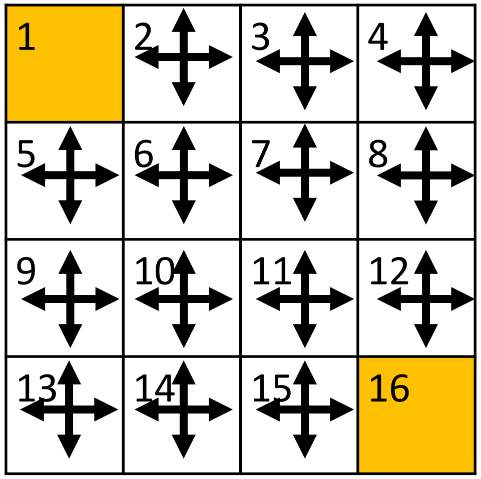
$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

Because a finite MDP has only a finite number of policies, the policy iteration has to converge to an optimal policy and optimal value function in a finite number of iterations.

Example: Policy Iteration

Policy iteration often converges in surprisingly few iterations:

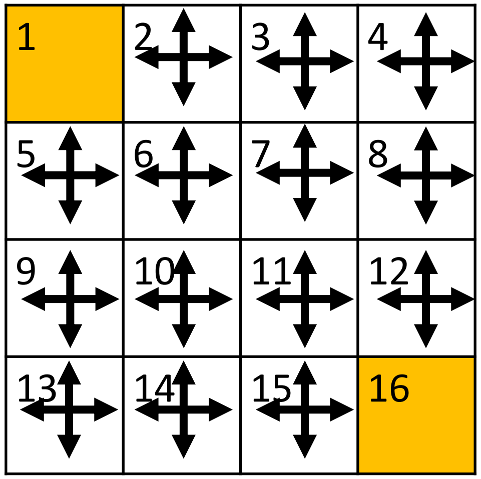
Take the random policy as π_0



Example: Policy Iteration

Policy iteration often converges in surprisingly few iterations:

π_0



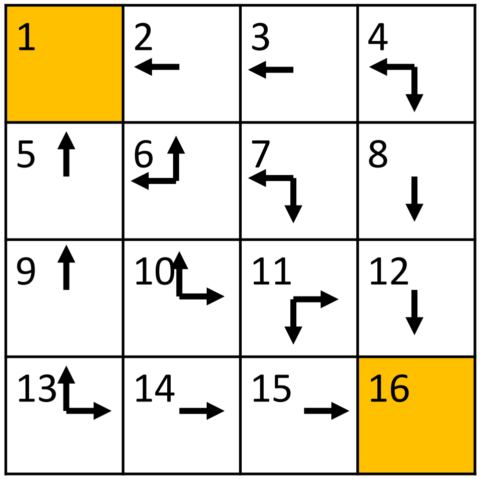
$\xrightarrow{E}$

v_{π_0}

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$\xrightarrow{I}$

π_1



Policy Iteration: Drawback

Each of its iterations involves policy evaluation, which itself is an iterative process, that may require multiple sweeps through the state set.

Policy Iteration: Drawback

Exact convergence to v_π occurs only in the limit in iterative policy evaluation.

Do we really need exact convergence?

→ No

Value iteration: stop policy evaluation after just one sweep of the state set.

Value Iteration

Value iteration can be written as a simple backup operation that combines the policy improvement and truncated policy evaluation steps:

$$\begin{aligned} v_{k+1}(s) &= \max_a E[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \textstyle\sum_{s',r} p(s', r | s, a)[r + \gamma v_k(s')], \end{aligned} \tag{4}$$

for all $s \in S$.

The sequence $\{v_k\}$ converges to v_* under the same assumptions that guarantee the existence of v_*, i.e.

- Either $\gamma < 1$ or
- Eventual termination is guaranteed from all states under the optimal policy

Asynchronous Dynamic Programming

Major drawback of DP methods discussed so far:

→ Require sweeps over the whole state set

→ If state set is very large: single sweep already prohibitively expensive

«Solution»: Asynchronous DP algorithms

Asynchronous Dynamic Programming

Asynchronous DP algorithms:

- Are iterative DP algorithms that are **not** organized in terms of systematic sweeps of the state set.
- Back up the values of states in any order whatsoever, using whatever values of other states happen to be available.
- Must continue to back up the values of all the states to converge correctly (can’t ignore any state after some point in the computation).

Asynchronous Dynamic Programming

Version of asynchronous value iteration:

On each step k it only backs up the value of one state s_k, using the value iteration backup:

$$v_{k+1}(s_k) = \max_a E[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s_k, A_t = a] \quad (4)$$

If $0 \leq \gamma < 1$, convergence to v_* is guaranteed given only that all states occur in the sequence $\{s_k\}$ infinitely often.

Asynchronous Dynamic Programming

Asynchronous algorithms make it easier to intermix computation with real-time interaction:

To solve a given MDP, we can run an iterative DP algorithm at the same time that an agent is actually experiencing the MDP
→ Experience can be used to determine states to which DP algorithm applies its backups

At the same time, the latest value and policy information from the algorithm can guide the agent's decision-making.

Generalized Policy Iteration

Policy iteration consists of two interacting processes:

- Policy evaluation: making value function consistent with the current policy
- Policy improvement: making the policy greedy w.r.t. the current value function

Generalized Policy Iteration

Generalized policy iteration (GPI) refers to the general idea of letting policy evaluation and policy improvement processes interact, independent of the granularity and other details of the two processes.

Generalized Policy Iteration

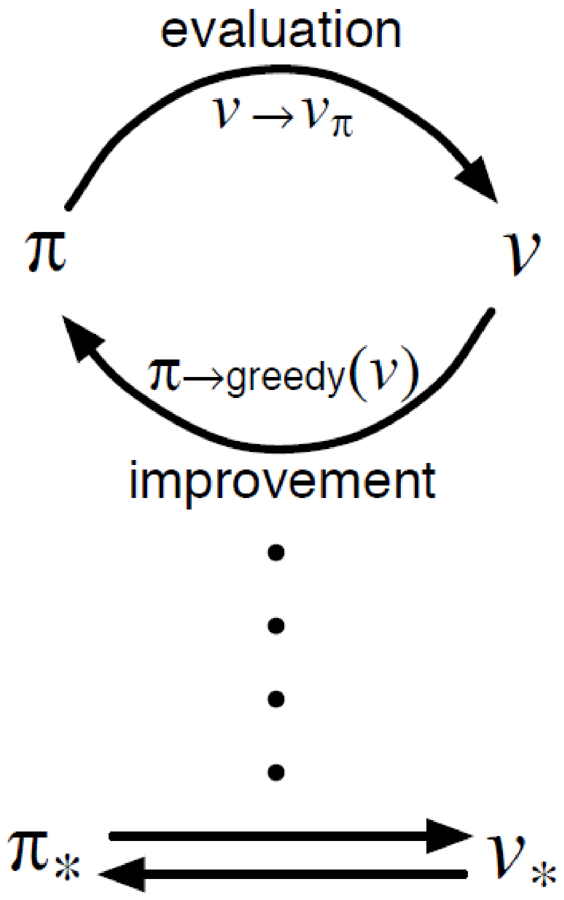
Interacting processes: Policy evaluation & policy improvement

- In policy iteration, theses two processes alternate, each completing before the other begins.
- In value iteration, only one iteration of policy evaluation is performed in between each policy improvement.
- In asynchronous DP methods, the evaluation and improvement processes are interleaved at an even finer grain.

As long as both processes continue to update all states, the ultimate result is typically the same: convergence to optimal value function and an optimal policy.

Generalized Policy Iteration

Almost all reinforcement learning methods are well described as GPI:



Generalized Policy Iteration

It is easy to see that if both the evaluation process and the improvement process stabilize, then the value function and policy must be optimal:

- Value function stabilizes only when it is consistent with current policy
- Policy stabilices only when it is greedy w.r.t. the current value function

→ Both processes stabilize only when a policy has been found that is greedy w.r.t. its own value function

→ Bellman optimality equation holds

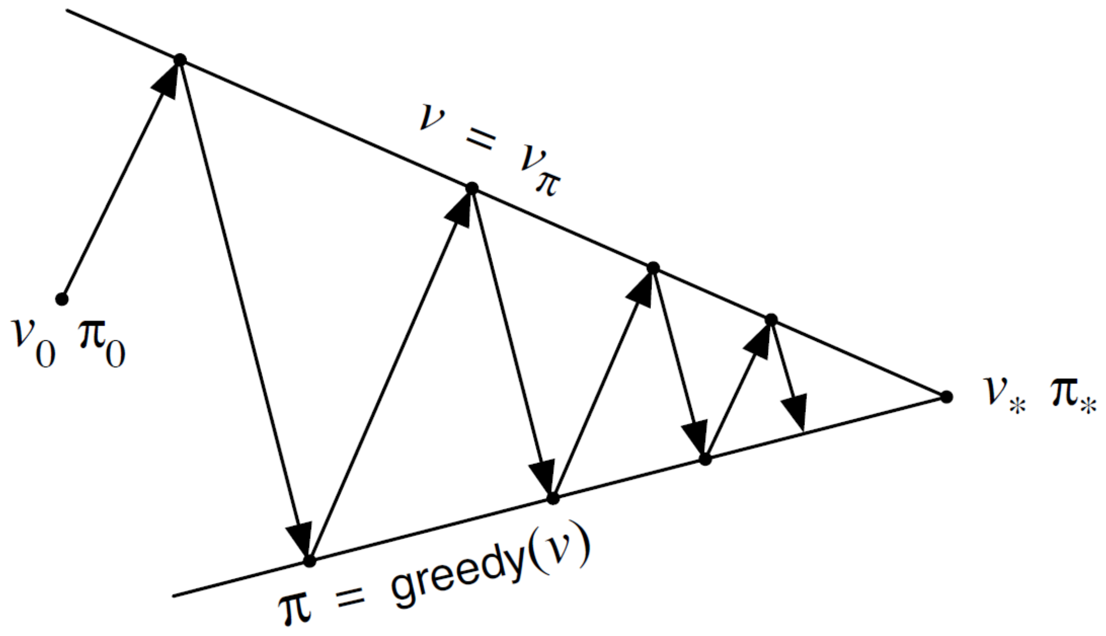
→ Policy and value funtion are optimal

Generalized Policy Iteration

Evaluation and improvement processes in GPI:
Both competing and cooperating

Pull in opposing directions

Interact to find optimal solution



Efficiency of Dynamic Programming

A DP method is guaranteed to find an optimal policy in polynomial time even though the total number of (deterministic) policies is m^n

- n = number of states
- m = number of actions

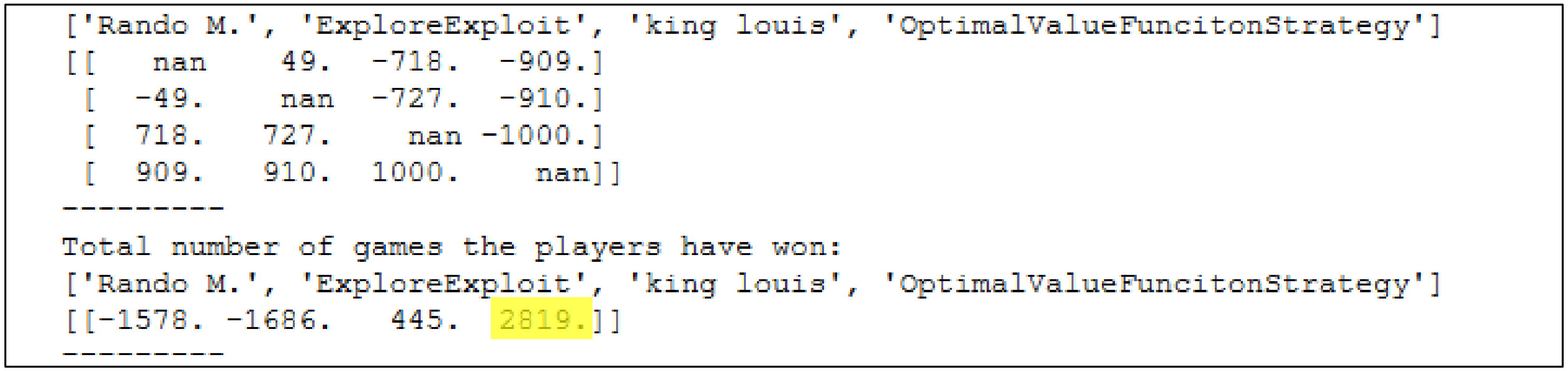
$\rightarrow$ DP is exponentially faster than any direct search in policy space could be

Efficiency of Dynamic Programming

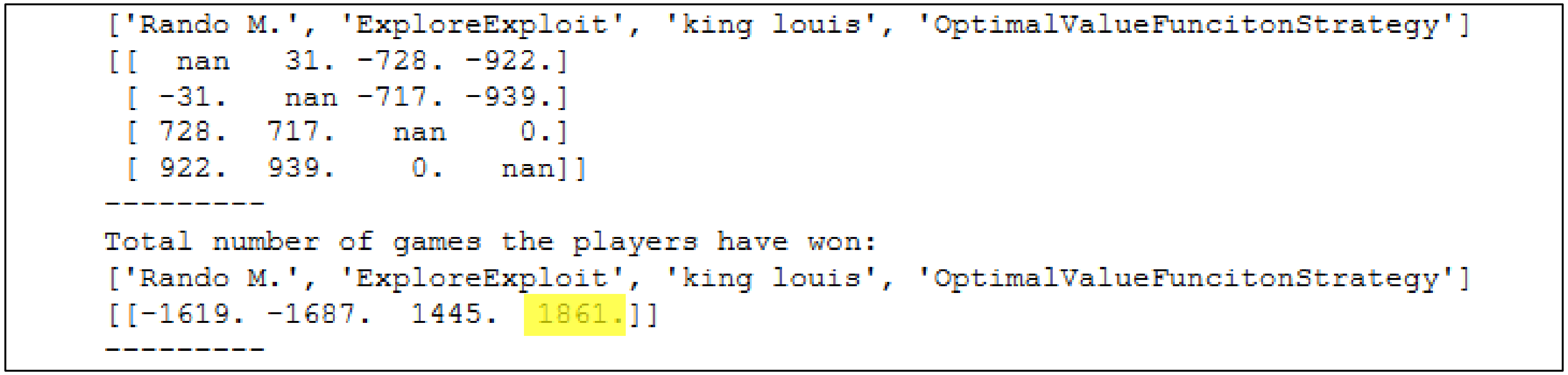
- In practice, DP methods can be used with today's computers to solve MDPs with millions of states.
- Both policy and value iteration are widely used, and it is not clear which, if either, is better in general.
- In practice, these methods usually converge much faster than their theoretical worst-case run times.
- On problems with large state spaces, asynchronous DP methods are often preferred.

Tic Tac Toe with Dynamic Programming

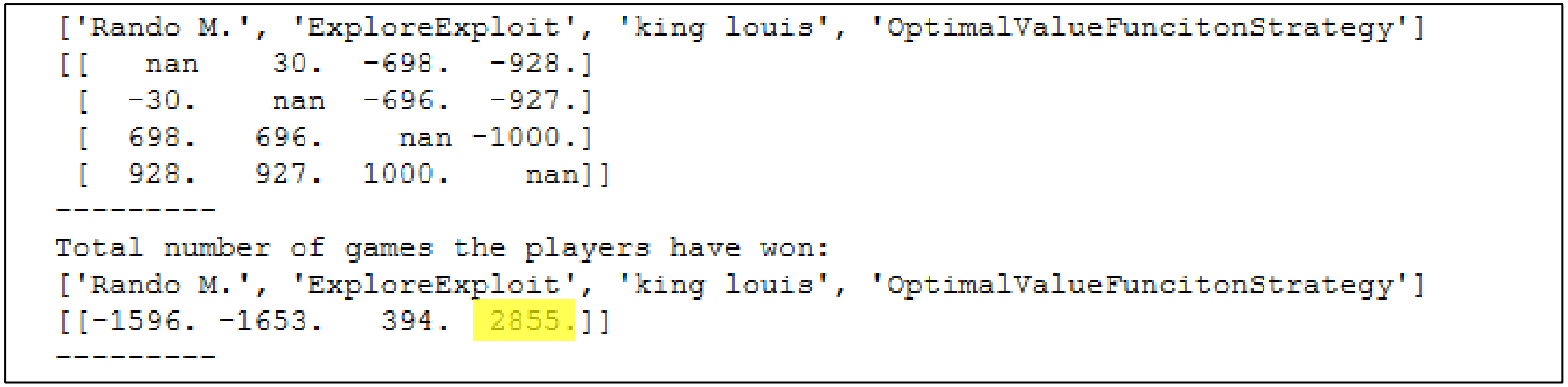
```
Initialize array $V$ arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that
    $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
```



```
['Rando M.', 'ExploreExploit', 'king louis', 'OptimalValueFuncitonStrategy']
[[   nan    49.  -718.  -909.]
 [  -49.    nan  -727.  -910.]
 [  718.   727.    nan -1000.]
 [  909.   910.  1000.    nan]]
----------
Total number of games the players have won:
['Rando M.', 'ExploreExploit', 'king louis', 'OptimalValueFuncitonStrategy']
[[-1578. -1686.   445.  2819.]]
----------
```



```
['Rando M.', 'ExploreExploit', 'king louis', 'OptimalValueFuncitonStrategy']
[[  nan   31. -728. -922.]
 [ -31.   nan -717. -939.]
 [ 728.  717.   nan    0.]
 [ 922.  939.    0.   nan]]
----------
Total number of games the players have won:
['Rando M.', 'ExploreExploit', 'king louis', 'OptimalValueFuncitonStrategy']
[[-1619. -1687.  1445.  1861.]]
----------
```



```
['Rando M.', 'ExploreExploit', 'king louis', 'OptimalValueFuncitonStrategy']
[[   nan    30.  -698.  -928.]
 [  -30.    nan  -696.  -927.]
 [  698.   696.    nan -1000.]
 [  928.   927.  1000.    nan]]
----------
Total number of games the players have won:
['Rando M.', 'ExploreExploit', 'king louis', 'OptimalValueFuncitonStrategy']
[[-1596. -1653.   394.  2855.]]
----------
```