Dynamic Programming to Compute Optimal Policies

Prof. Matthias Grossglauser

Information and Network Dynamics (INDY) lab
School of Computer and Communication Sciences (I&C)
EPFL

Recap

- Markov Decision Process (MDP): states that capture everything we need to know about the past, actions that can only depend on states
- Bellman equations: expresses recursive consistency of value function

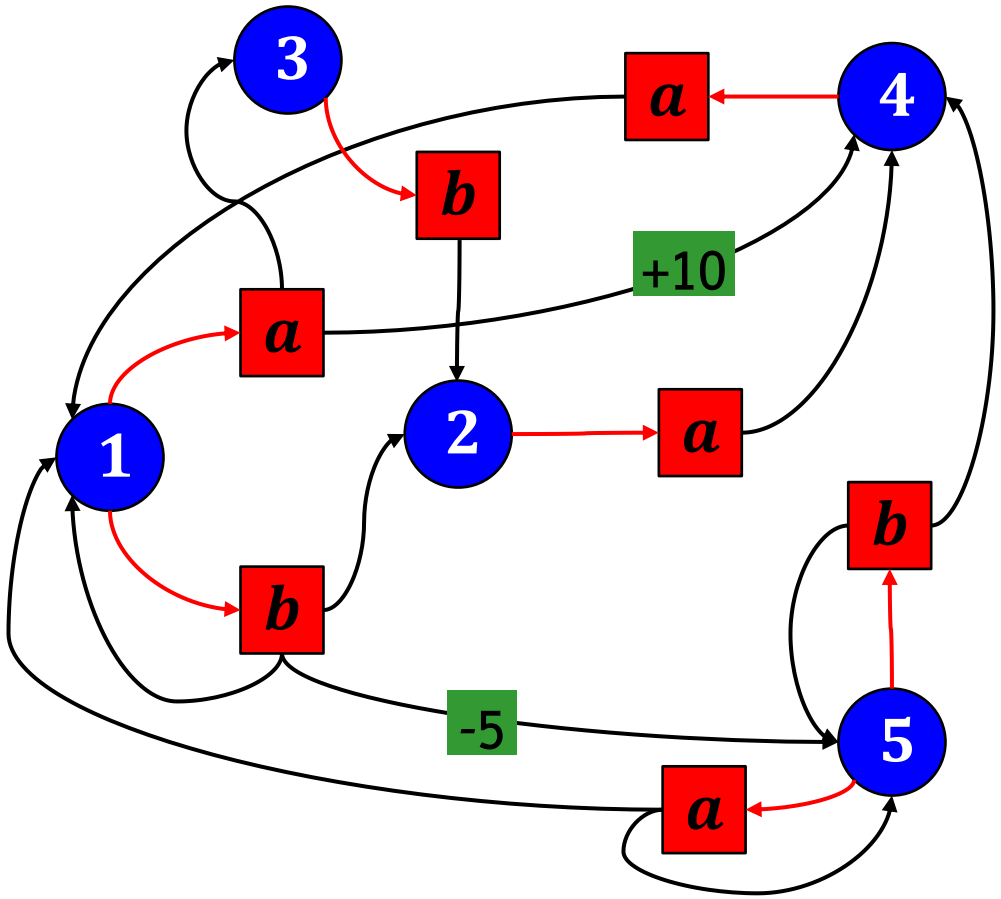
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] = \sum_a \pi(a|s)\, q_\pi(s,a)$$

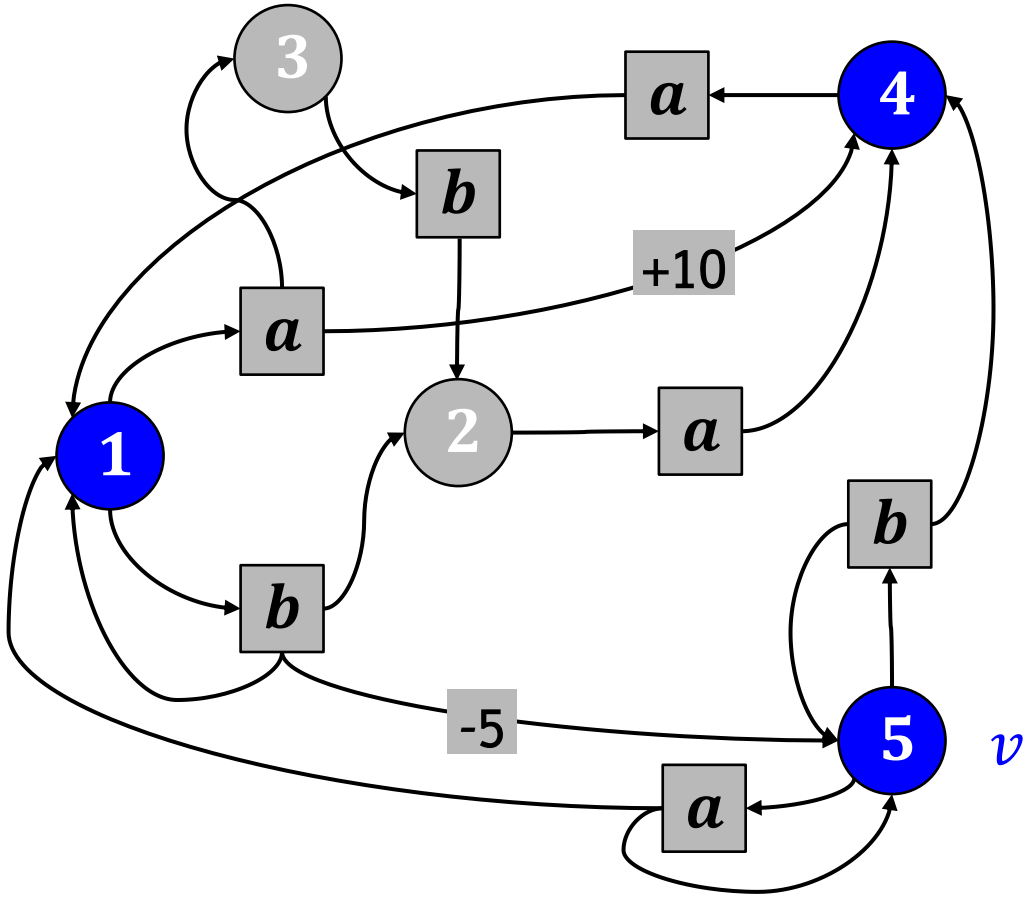
$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] = \max_a q_*(s,a)$$

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

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

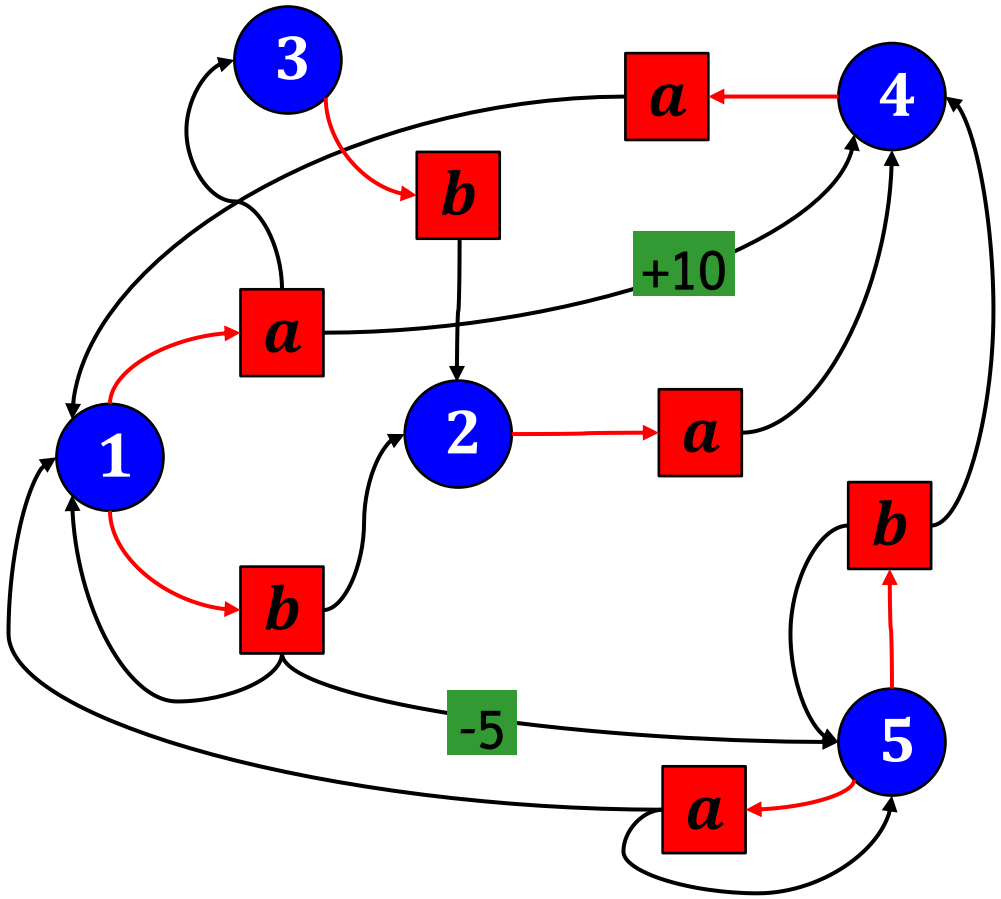
Bellman equations illustrated: state-value function

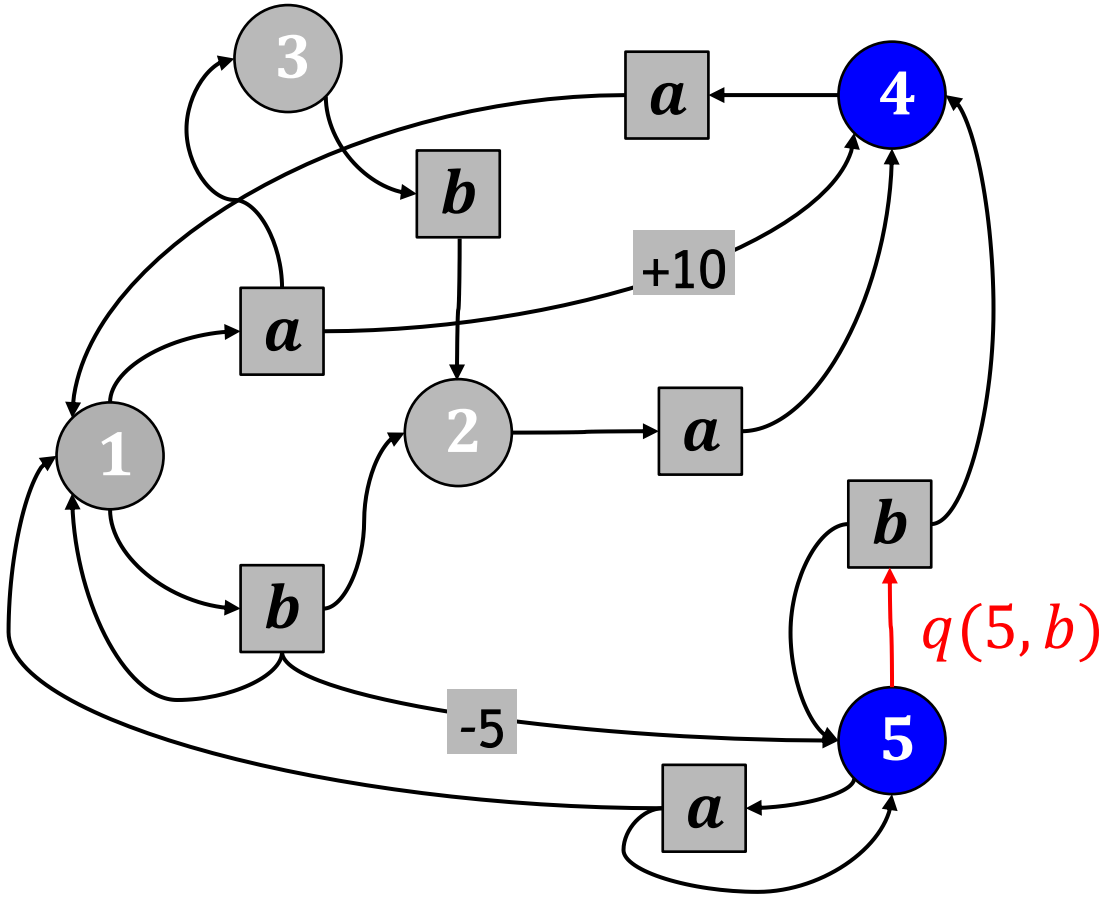




$v(5)$

Bellman equations illustrated: action-value function





Iterative application of the Bellman equation

- Policy evaluation, aka prediction problem: compute v_π for some π
- $v_\pi(s) \quad \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t|S_t = s]$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\right]$$

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

- Suggests an iterative solution:
  - $v_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s]$

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

- Note: v_π is a fixed point of this update operator

Algorithm: iterative policy evaluation

INPUT: policy π, env $p(.,.|.,.)$, threshold $\theta > 0$

Initialize: $v(s)$ arbitrary for $s \in S$, $v(\otimes) = 0$

REPEAT:

$\Delta \coloneqq 0$

For $s \in \mathcal{S}$:

$v \coloneqq v(s)$

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

$\Delta \coloneqq \max(\Delta, |v - v(s)|)$

UNTIL $\Delta < \theta$

- Note: this is slightly different from equation on previous page: "in-place" version, $v(.)$ mixes values of iterations k and $k + 1$
- This uses half the space, and usually converges faster

Gridworld example

⊗	1	2	3
4	5	6	7
8	9	10	11
12	13	14	⊗

- Actions: N,S,E,W (except where we bump into the wall)
  - $p(10, r|5, E) = 0,\ p(7, -1|7, E) = 1$
  - $p(4, -1|5, W) = 1$
- Rewards: -1 per step
  - $\gamma = 1$
- Episodic, top-left and bottom-right are end states

Example policy evaluation: Gridworld

v_k for $\pi(.|s) = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ for every s

$v_0(.)$

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

Note: any initialization would work, we set all values to 0 for simplicity

Example policy evaluation: Gridworld

$v_1(.)$

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

$v_k(\otimes) = 0$
terminal state clamped to zero

$v_2(.)$

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

$$v_2(1) = \frac{1}{4}\big((-1+0)+(-2)+(-2)+(-2)\big)$$
$$= -\frac{7}{4} = -1.75$$

Example policy evaluation: Gridworld

$v_3(.)$

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

$v_\infty(.)$

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

$v_{10}(.)$

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

Policy improvement: greedy policy

- In state s, should we choose an action $a \neq \pi(s)$?
- Idea: check whether taking action a now, and then follow π again forever, is better than following π now and forever
- Value of changed behavior:

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$

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

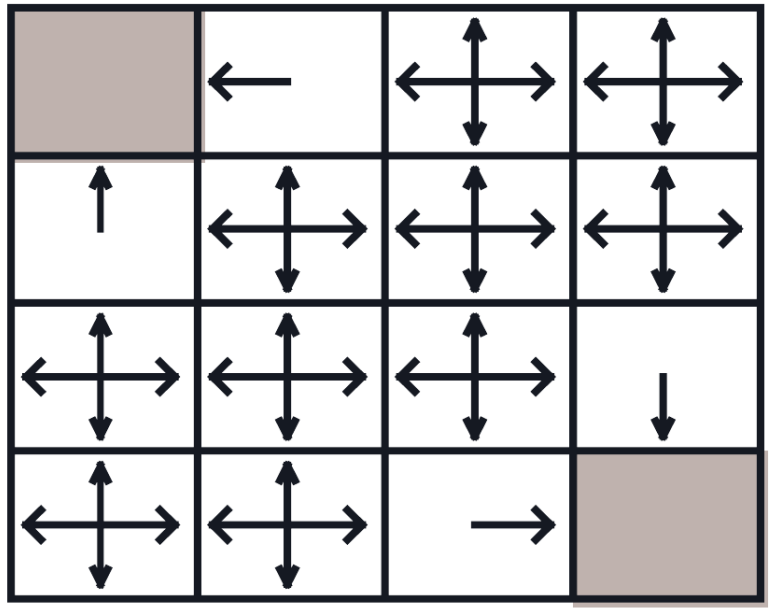
Note: PMF of this expectation does not depend on π (other than through $v_\pi(.)$) → $\mathbb{E}$ instead of $\mathbb{E}_\pi$

- Value of unchanged behavior: $v_\pi(s)$
- If $q_\pi(s,a) > v_\pi(s)$, then do a instead of $\pi(s)$
- Even better: favor a every time we are in state s → $\pi'(s) = a$

Greedy policy in Gridworld

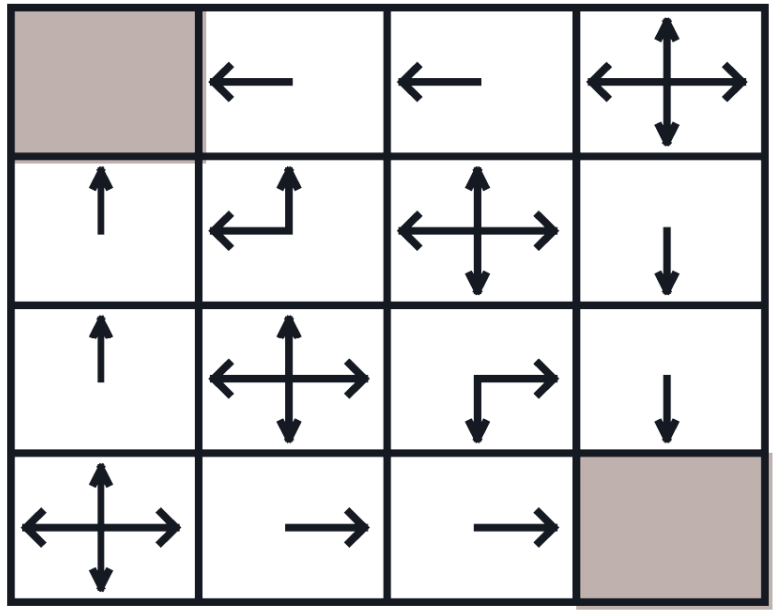
$v_1(.)$

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



$v_2(.)$

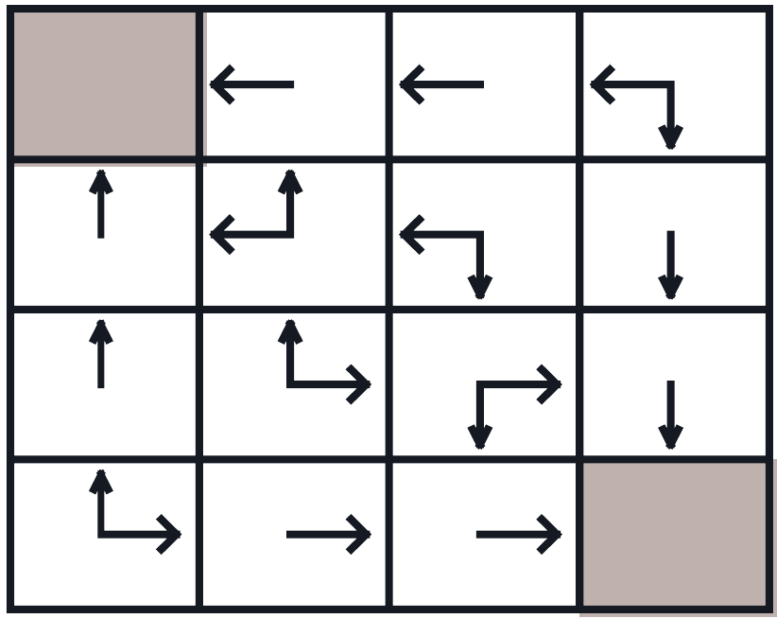
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Greedy policy in Gridworld

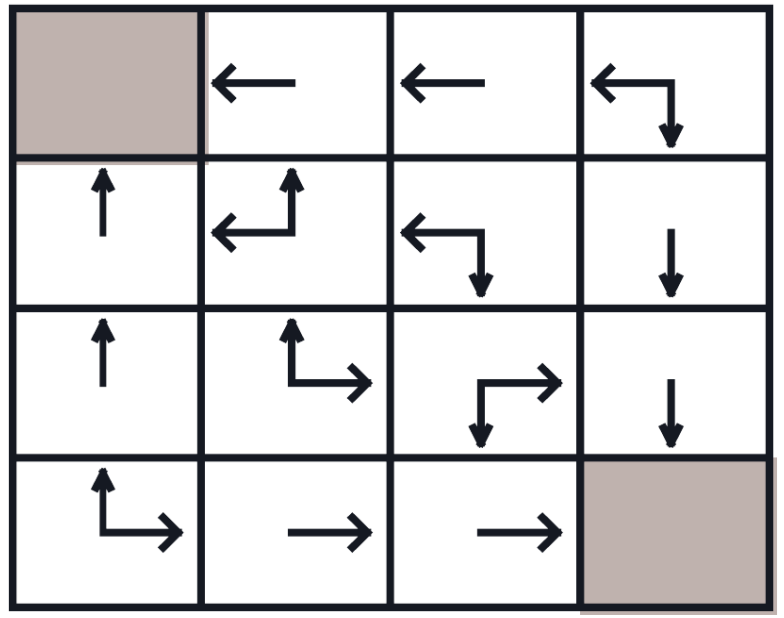
$v_3(.)$

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



$v_{10}(.)$

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

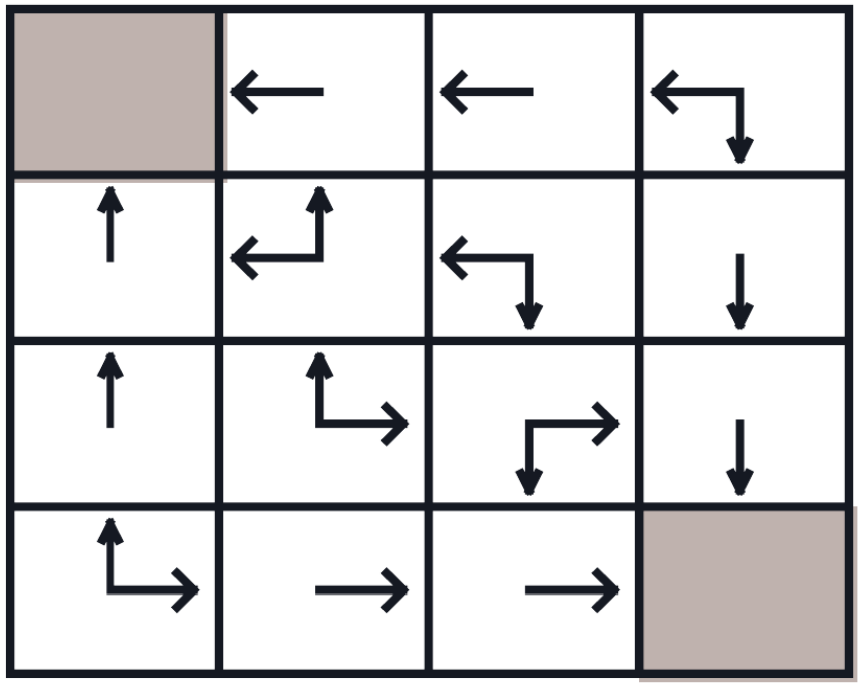


Greedy policy in Gridworld

$v_\infty(.)$

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

Greedy policy in Gridworld
Note: this is with respect to π, so not equal to the optimal policy in general!



Note: greedy policy did not change since $k = 3$

Policy improvement theorem

- Theorem: let π and π' be two deterministic (for now) policies such that, for all $s \in S$:

  $q_\pi(s, \pi'(s)) \geq v_\pi(s)$

  then $\pi' \geq \pi$ (i.e., policy π' is as good as or better than policy π)

  which means that $v_{\pi'}(s) \geq v_\pi(s)$ **for all $s \in S$**

- Intuition: choosing a better action in at least one state can increase expected cumulative reward for other/all states – and it can never hurt!

Policy improvement theorem: proof

from old policy

from new policy

- $v_\pi(s) \le q_\pi(s, \pi'(s))$

$$= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)]$$

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

① $$= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

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

$$\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s]$$

$$= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] | S_t = s]$$

$$= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}] \middle| S_t = s\right]$$

② $$= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \middle| S_t = s\right]$$

$$\mathbb{E}[\mathbb{E}[A|B]] = \mathbb{E}[A]$$

③ $$\le \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \middle| S_t = s\right] \ldots$$

Policy improvement theorem: proof

- $v_\pi(s) \leq q_\pi\ (s, \pi'(s))$

① $= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\ ]$

② $\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \big| S_t = s\ ]$

③ $\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \underbrace{\gamma^3 v_\pi(S_{t+3})}_{\to 0} \big| S_t = s\ ]$

④

⑤ …

∞ $\leq \mathbb{E}_{\pi'}[\underbrace{R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \gamma^4 R_{t+5} + \cdots}_{= G_t} \big| S_t = s\ ]$

$= v_{\pi'}(s)$

Policy improvement: all states

- Greedy policy π': maximize action-value from every state
- $\pi'(s) \coloneqq \arg\max_{a \in \mathcal{A}} q_\pi(s, a)$

$$= \arg\max_{a \in \mathcal{A}} \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$

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

- Note: to get a deterministic policy → break ties arbitrarily

If greedy policy is no improvement ⟹ policy is optimal

- Suppose the greedily updated policy π' is not better than π: $v_\pi = v_{\pi'}$, i.e.,

$$v_{\pi'}(s) \coloneqq \max_{a \in \mathcal{A}} q_\pi(s,a)$$

$$= \max_{a \in \mathcal{A}} \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$

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

- Recall: value function of an optimal policy:

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

- Therefore: **if a policy π cannot be greedily improved any more, it is optimal!**
  - This property carries over to stochastic policies $\pi(.|.)$
  - But there always exists at least one optimal deterministic policy among the optimal stochastic policies

Policy iteration

- Recap: what do we have so far?
  - (1) Policy evaluation: given π, iterative procedure to approximate v_π, q_π
  - (2) Policy improvement: greedy policy π' derived from v_π that we know is better: $\pi' \geq \pi$ (but we do not know $v_{\pi'}$ yet!)

- $\pi_0 \xrightarrow{\text{eval}} v_{\pi_0} \xrightarrow{\text{impr}} \pi_1 \xrightarrow{\text{eval}} v_{\pi_1} \xrightarrow{\text{impr}} \pi_2 \xrightarrow{\text{eval}} v_{\pi_2} \xrightarrow{\text{impr}} \dots \xrightarrow{\text{impr}} \pi_* \xrightarrow{\text{eval}} v_* \xrightarrow{\text{impr}} \pi_*$

No change → $*$ reached

- We are only considering deterministic policies → finite set of possible policies → value function is non-decreasing so no cycles → the above process terminates in a finite number of steps

Algorithm: policy iteration

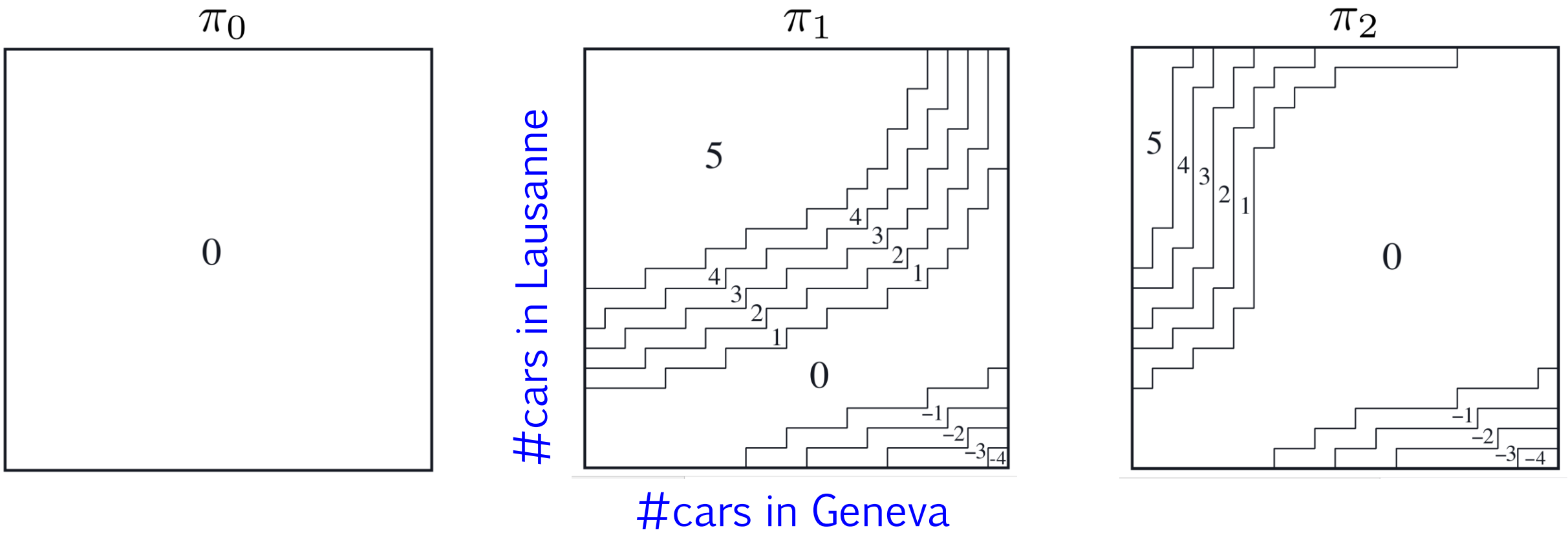
INPUT: $p(.,.|.,.), \theta > 0$

Initialize: $v(s)$ and $\pi(s)$ arbitrary (but $v(\otimes) = 0$)
REPEAT:
    REPEAT: /* policy evaluation /*
        $\Delta := 0$
        For $s \in \mathcal{S}$:
            $v := v(s)$
            $v(s) := \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma v(s')]$
            $\Delta := \max(\Delta, |v - v(s)|)$
    UNTIL $\Delta < \theta$
    policy-stable := TRUE /* policy improvement */
    For $s \in \mathcal{S}$:
        old-action := $\pi(s)$
        $\pi(s) := \arg\max_{a \in \mathcal{A}} \sum_{s',r} p(s', r|s, a)[r + \gamma v(s')]$
        IF $\pi(s) \neq$old-action, THEN policy-stable:=FALSE
UNTIL policy-stable
RETURN: $v_* = v, \pi_* = \pi$

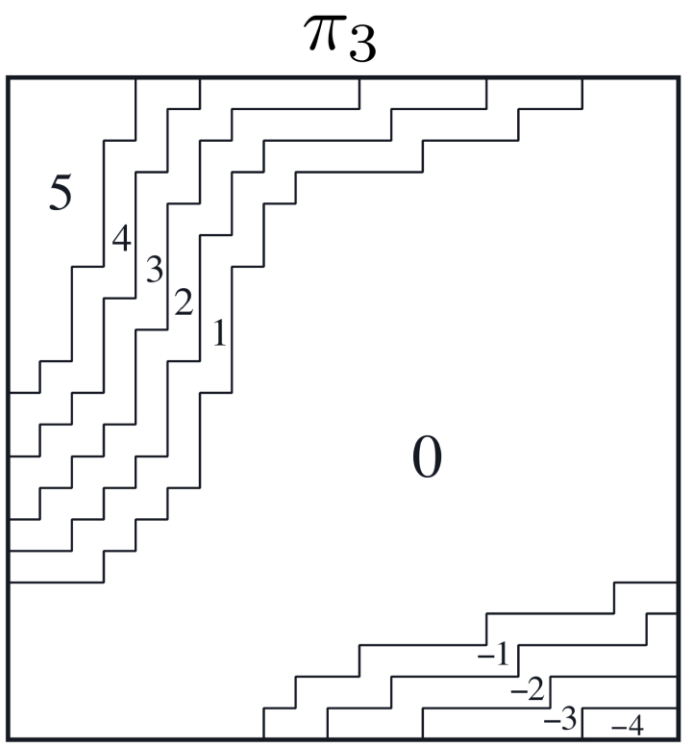
Policy iteration: example

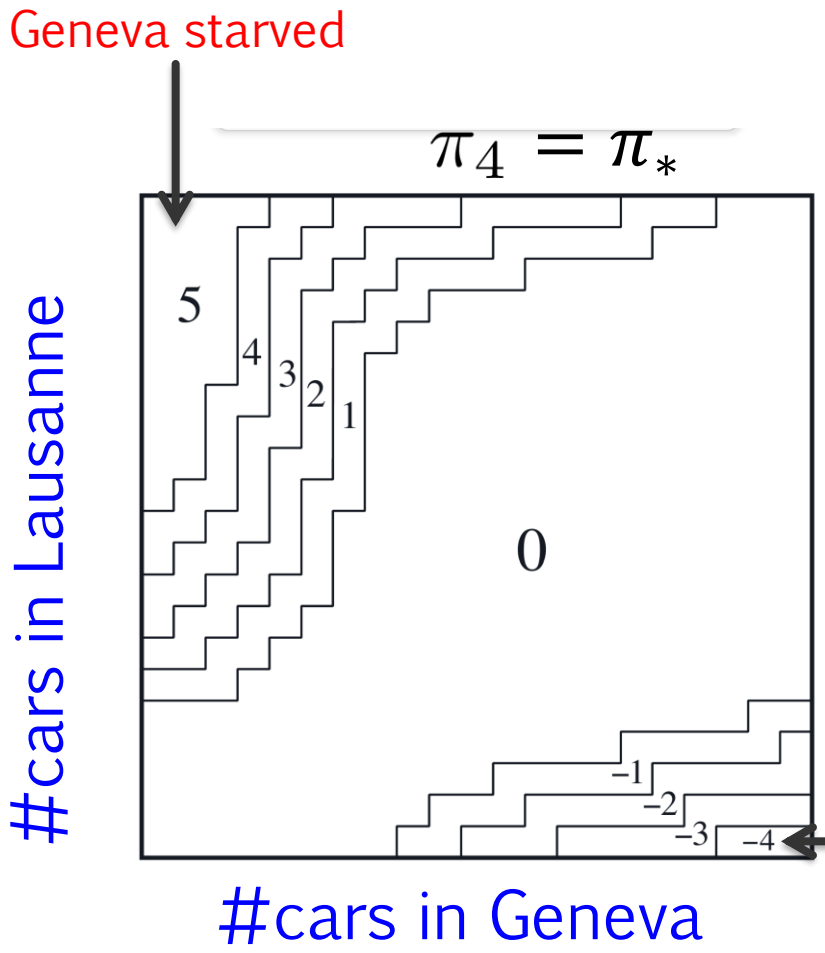
- Jacques's car rental:
  - Two rental locations with max capacity of 20 cars
  - Each day, random number of customers arrive at each
    - Lausanne: Poisson (3)
    - Geneva: Poisson (4)
  - If a car is available, the customer rents it, Jacques earns 100 CHF
  - Each day, some cars are returned (some may be returned elsewhere)
    - Lausanne: Poisson (3)
    - Geneva: Poisson (2)
  - Jacques tries to match demand by moving cars overnight
    - At most 5 cars
    - Costs CHF 20
- Model this as an MDP:
  - Iteration = day; continuing, $\gamma = 0.9$
  - State = (#cars Lausanne, #cars Geneva) $\in \{0,1,2,\dots,20\}^2$
  - Policy: # cars moved (+ L→G, - G→ L) $\pi(s) \in \{-5,-4,\dots,+4,+5\}$

Jacques's two-location car rental



Jacques's two-location car rental





Note:

- Large part of state space: do nothing → because of neg reward of moving cars
- Never move more than 4 cars to Lausanne → Geneva on avg more starved

From policy iteration to value iteration

- Disadvantage of policy iteration: potentially many costly iterations to compute v_π accurately before obtaining a new policy π'
- Can we "mesh" the two phases more tightly? → Yes: **value iteration**
  - Step (1): single policy-evaluation update step to get a (perhaps only slightly) better estimate of v_π
  - Step (2): immediately improve the policy $\pi \to \pi'$
  - Repeat until the policy is stable
- $v_{k+1}(s) \coloneqq \max_{a \in \mathcal{A}} \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a]$

  $= \max_{a \in \mathcal{A}} \sum_{s',r} p(s', r | s, a)[r + \gamma v_k(s')]$
- Value iteration is policy iteration with lazy value evaluation
  - Essentially, propagate value updates only one hop, then revisit policy already

Algorithm: value iteration

INPUT: $p(.,.|.,.), \theta > 0$
Initialize: $v(s)$ and $\pi(s)$ arbitrary (but $v(\otimes) = 0$)

REPEAT:
$\Delta \coloneqq 0$
For $s \in \mathcal{S}$:
$v \coloneqq v(s)$
$v(s) \coloneqq \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v(s')]$
$\Delta \coloneqq \max(\Delta, |v - v(s)|)$
UNTIL $\Delta < \theta$

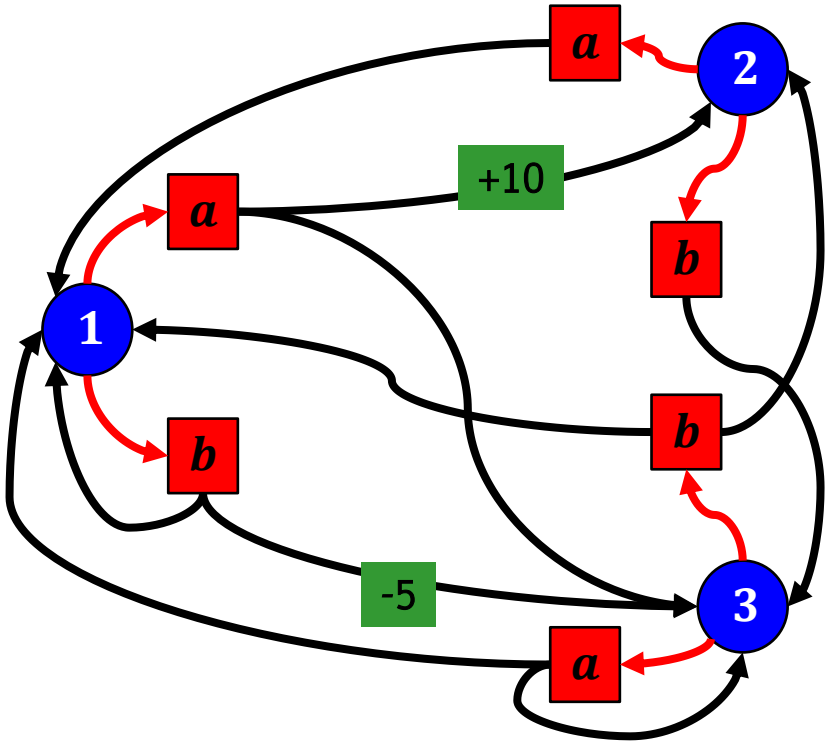
RETURN $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v(s')]$

- Note: because we do not compute v_π exactly here, but only an "intermediate estimate", we cannot make **policy-stable** a stopping criterion (as we did with PI)

Robustness to update schedule

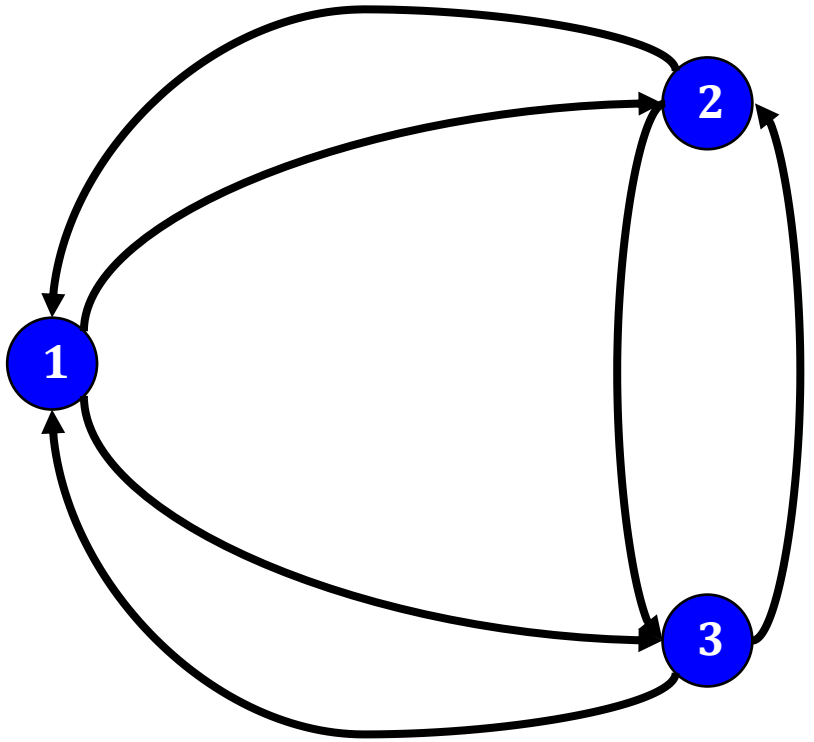
- The update rule $v(s) \coloneqq \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v(s')]$:
  - For s, we need the value of every "downstream" state s' (reachable for any action a with nonzero probability)

state-action transition graph



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

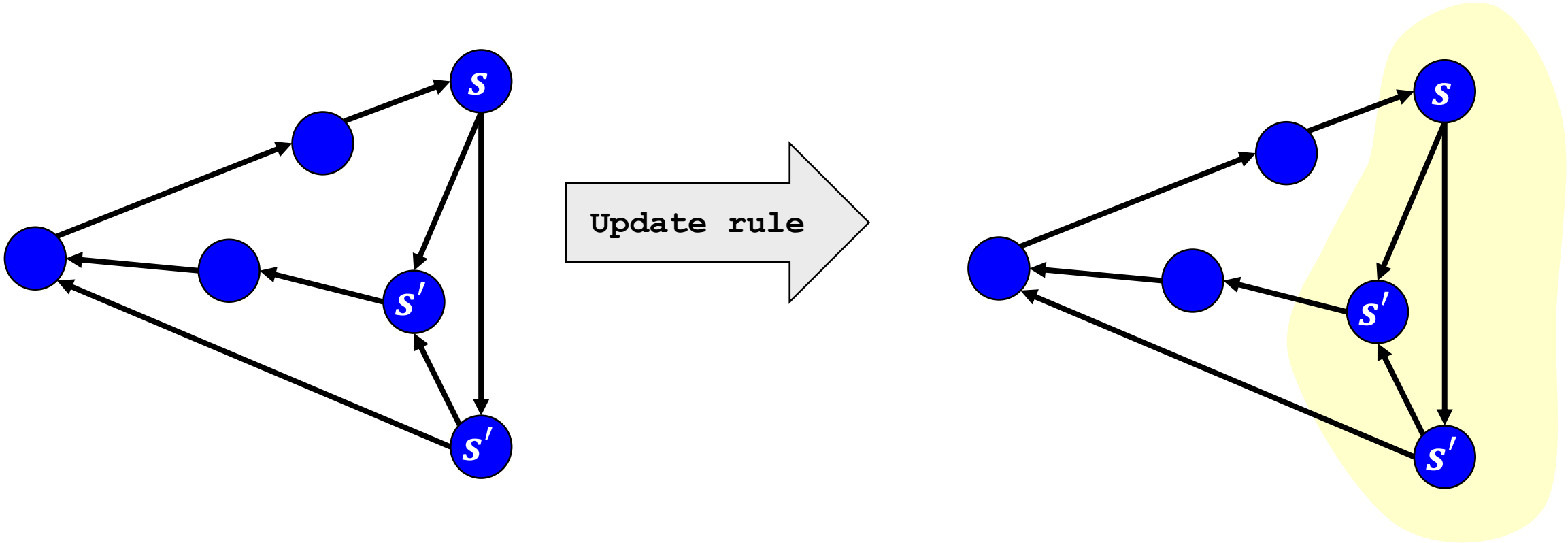
state-state transition graph



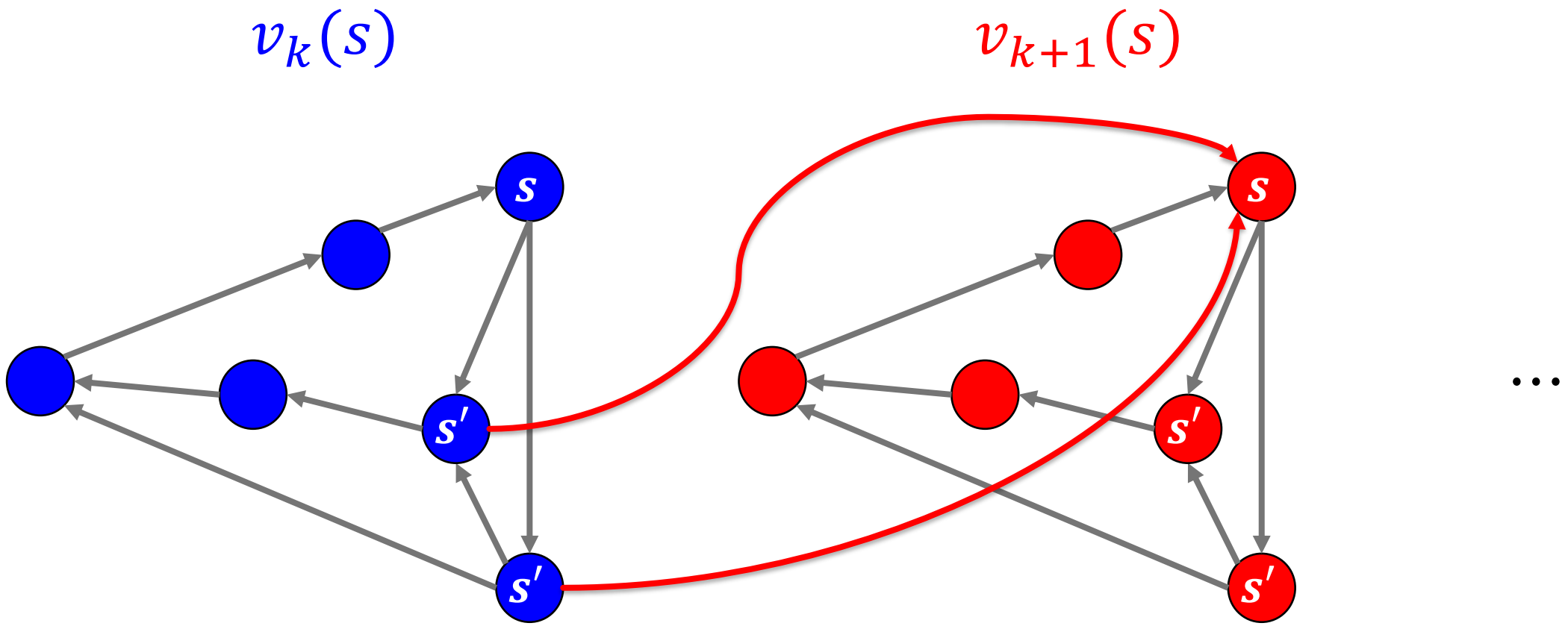
Robustness to update schedule

- The update rule $v(s) \coloneqq \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v(s')]$:
  - For s, we need the value of every "downstream" state s' (reachable for any action a with nonzero probability – this may include $s' = s$)

Let us look at a larger, sparser state-state graph

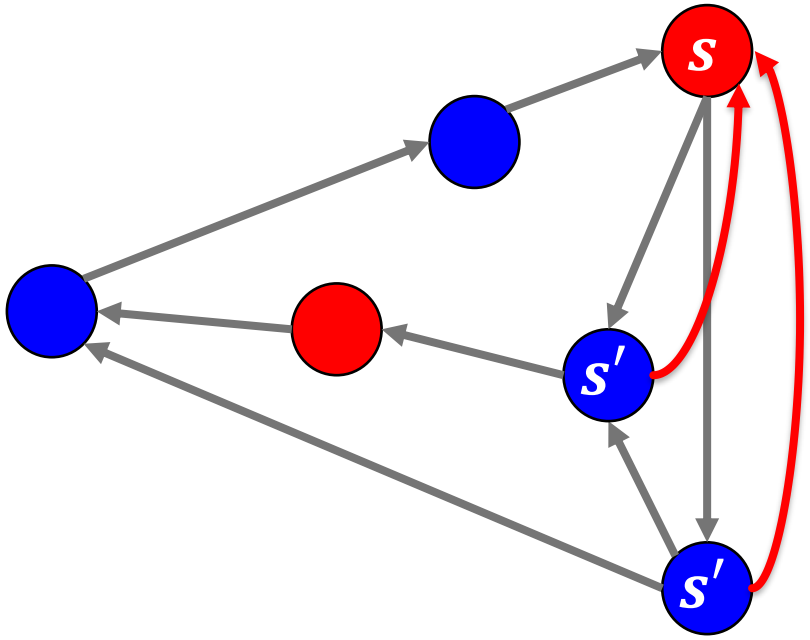


Update schedule: theory assumed $v_k(.) \rightarrow v_{k+1}(.) \rightarrow \cdots$



Update schedule: algorithm uses in-place

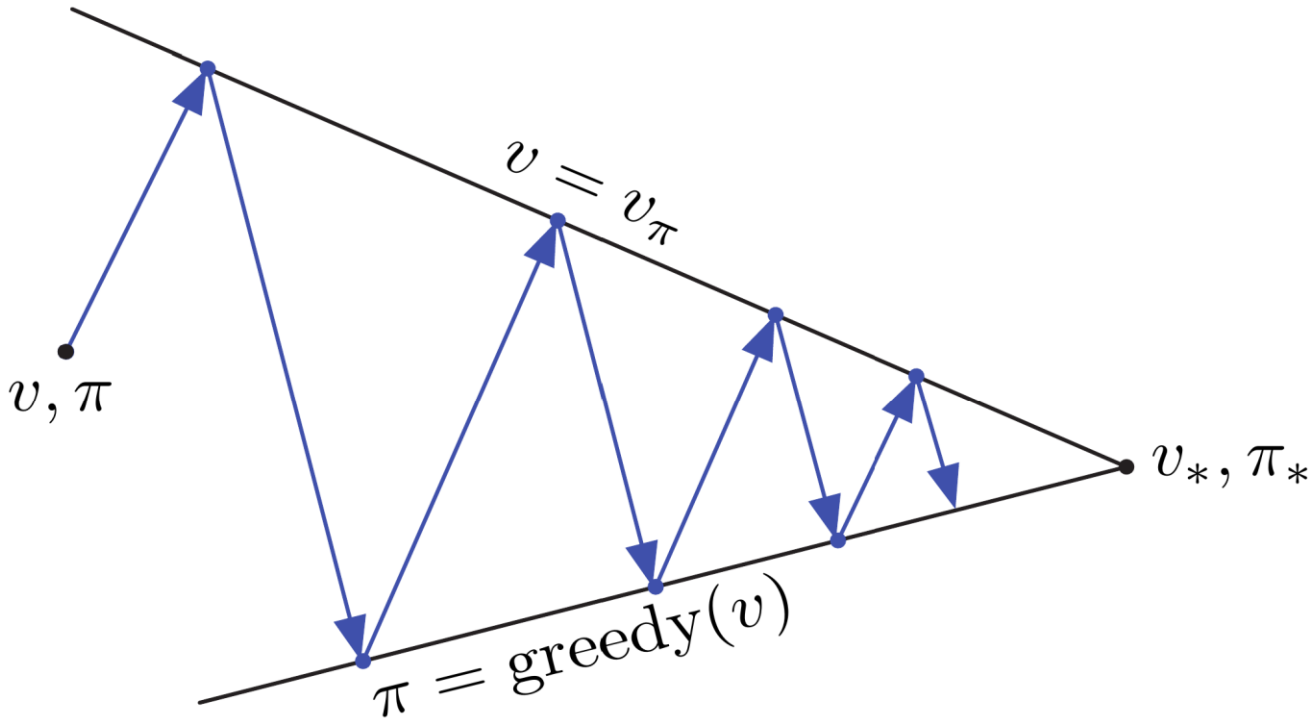
$v_k(s)$ and $v_{k+1}(s)$



- Sweep through state space in every round
  - Order does not matter for convergence
- Value iteration algorithm assumes "in-place" updates: some values $v(s)$ have seen k iterations, others already $k+1$
- DP is robust to in-place, and in fact often converges faster → use new information immediately
- This works for even more unsynchronized schedules: update one state 10 times, another once…
- As long as no state is "permanently left out" → convergence

Generalized policy iteration

- Insight from policy-iteration vs value-iteration: convergence to optimal policy is robust
- Two processes can work on estimates in a highly unsynchronized manner:
  - First process: update values of states in some arbitrary order
  - Second process: update greedy policy of states in some arbitrary order
- This robustness → room for optimization:
  - Focus on important states that matter (occur often, bad estimate,…)



Summary

- Dynamic Programming: a general method to solve problems that lend themselves to subdivision into smaller problems
- Here: optimal value of a state can be expressed as a function of optimal values of adjacent states
- Bellman equations give rise to iterative approximation of value functions
- Greedy policy: local improvement in a state, based on value function of previous policy
- We have a lot of freedom in how to combine value estimation and policy updates:
  - Iterative policy evaluation: approximate v_π (multiple sweeps), then update π
  - Value iteration: one-step update to v_π (one sweep), then update π
  - More asynchronous merging of these processes is not problematic
- Reading assignment: [S&B] chapter 4