TÉCNICO LISBOA

Planning, Learning and Intelligent Decision Making

Lecture 5

PADInt 2024

Policies

(or “ways of selecting actions”)

Policy

- A **policy** is a (maybe random) "rule" that tells an agent/decision-maker what actions to choose in each step
- If the agent follows the rules prescribed by a policy, we say that the **agent is following the policy**

Policy

- Policies can select actions...
  - ... at random
  - ... deterministically
  - ... using information from the past
  - ... using only current information
  - ... always in the same way
  - ... in different ways as time goes by

History

- The **history** at time step t...
  - ... is a random variable, h_t
  - ... contains all that the agent saw up to time step t:

$$\mathrm{h}_t = \{\mathrm{x}_0, \mathrm{a}_0, \mathrm{x}_1, \mathrm{a}_1, \ldots, \mathrm{x}_{t-1}, \mathrm{a}_{t-1}, \mathrm{x}_t\}$$

- The set of t-length histories (histories up to time t) is denoted as $\mathcal{H}_t$

Policies

- A **policy** is a (random?) mapping π from **histories** to **actions**
- A **policy** is a mapping $\pi : \mathcal{H} \to \Delta(\mathcal{A})$

Distributions over actions

- A policy chooses each action $a \in \mathcal{A}$ with a probability that depends on the history ht:

$$\pi(a \mid h_t) = \mathbb{P}[\mathrm{a}_t = a \mid \mathrm{h}_t = h_t]$$

- In general, policies are **stochastic** (random) and **history-dependent**

Policies

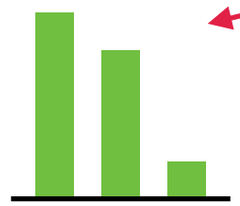
History-dependent

x_0	a_0	...	x_t

History up to time t

$\pi(\cdot \mid h_t)$

Stochastic

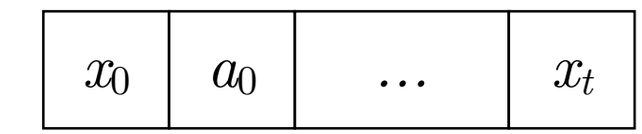


$a_1 \quad a_2 \quad a_3$

Distribution over actions

Types of policies

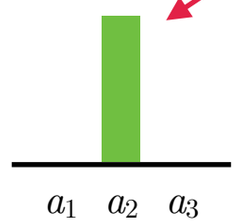
- A policy is **deterministic**...
  - ... if there is one action that is selected with probability 1



History up to time t

$\pi(\cdot \mid h_t)$

All probability mass is in a single action



Only one action has positive probability

Types of policies

- A policy is **Markov**...
  - ... if the distribution over actions given the history **depends only on the last state** (and t)
    - If $h_t = \{x_0, a_0, \ldots, a_{t-1}, x_t\}$, then

$$\pi(a \mid h_t) = \mathbb{P}[\mathrm{a}_t = a \mid \mathrm{h}_t = h_t] = \mathbb{P}[\mathrm{a}_t = a \mid \mathrm{x}_t = x_t]$$

Depends only on x_t
(and eventually t)

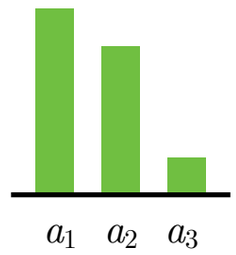
Types of policies

- **Markov policy:**

History up to time t

$\pi_t(\cdot \mid x_t)$

Depends on t



Distribution over actions

Types of policies

- A policy is **stationary**...
  - ... if the distribution over actions given the history depends only on the last state (and **not on t**)
    - If $h_t = \{x_0, a_0, \ldots, a_{t-1}, x_t\}$, then

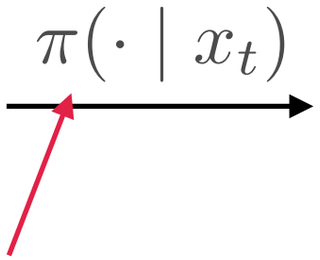
$$\pi(a \mid h_t) = \mathbb{P}[\mathrm{a}_t = a \mid \mathrm{h}_t = h_t] = \mathbb{P}[\mathrm{a}_t = a \mid \mathrm{x}_t = x_t]$$

Depends only on x_t
(but not on t)

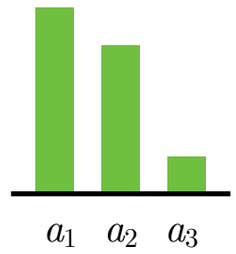
Types of policies

- **Stationary policy:**

History up to time t



Does not depend on t



Distribution over actions

Types of policies

- Non-deterministic vs deterministic (simpler)

Chooses actions randomly

Does not choose actions randomly

- Non-Markov vs Markov (simpler)

Depends on the whole past

Depends only on the last state

- Non-stationary vs stationary (simpler)

Changes with time

Fixed through time

The ideal case

- The policy that we are looking for is (ideally):

Deterministic

Stationary

Markov cost process

- If the agent follows a **stationary policy** π, the actions are fully determined by the state
- The resulting state process is a **Markov chain** $(\mathcal{X}, \mathbf{P}_\pi)$

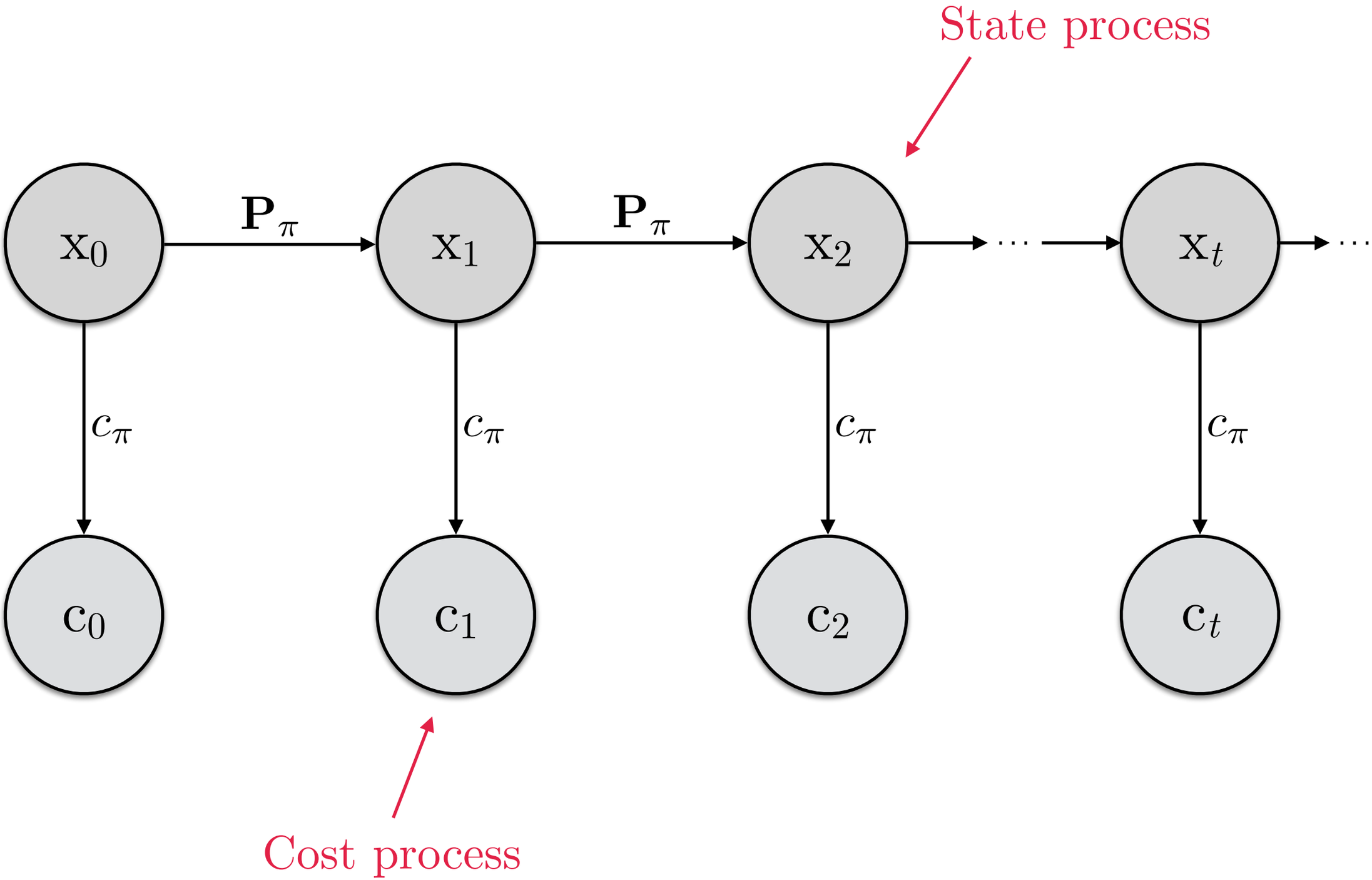
$$\mathbf{P}_\pi(y \mid x) = \mathbb{E}_{\mathrm{a}\sim\pi(x)}[\mathbf{P}(y \mid x, \mathrm{a})] = \sum_{a\in\mathcal{A}} \pi(a \mid x)\mathbf{P}(y \mid x, a)$$

Markov cost process

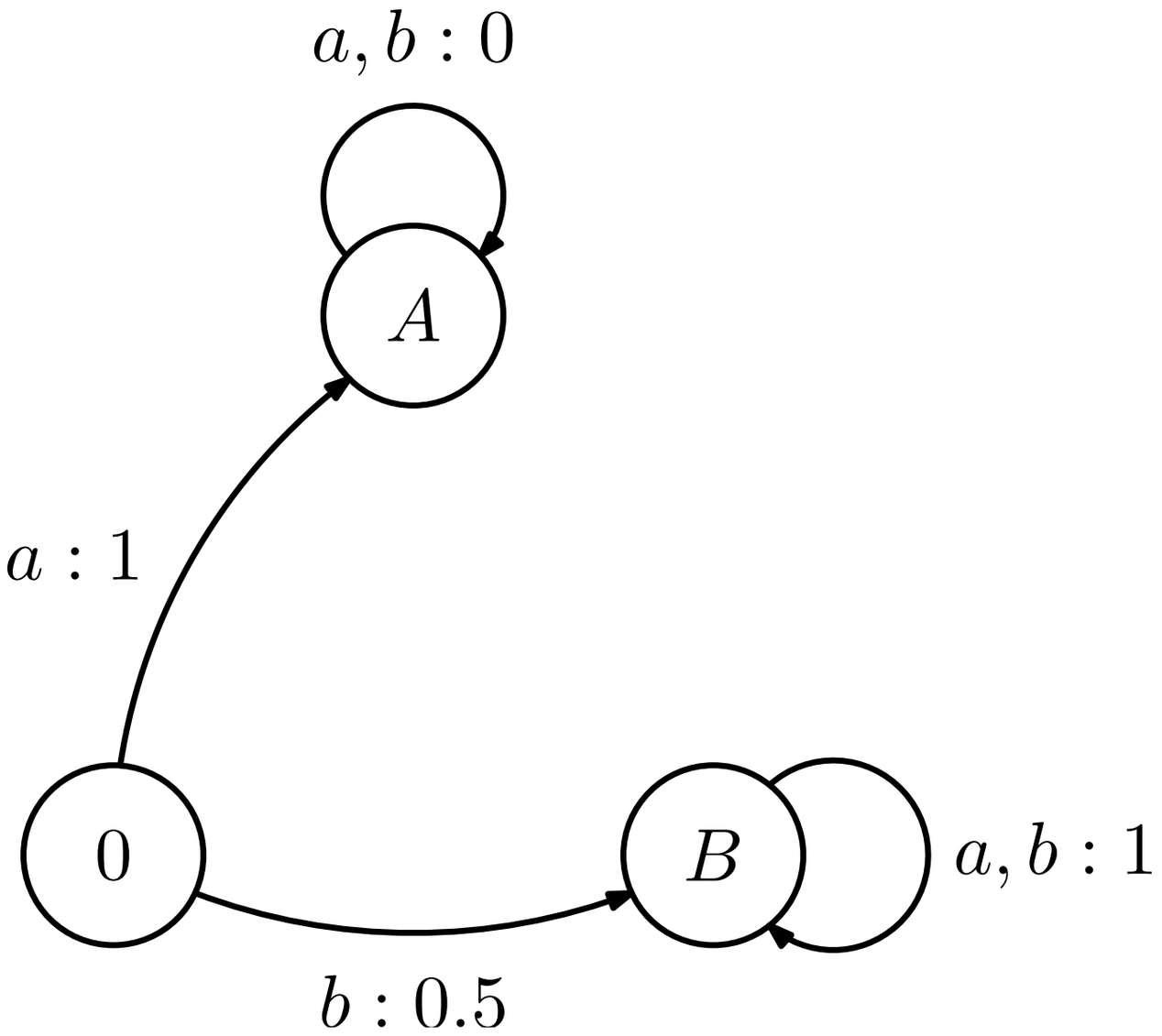
- The **costs** can also be fully determined from the states visited by the chain
- Whenever the chain visits a state x, it pays an expected cost of

$$c_\pi(x) = \mathbb{E}_{\mathrm{a}\sim\pi(x)}[c(x,\mathrm{a})] = \sum_{a\in\mathcal{A}} \pi(a \mid x) c(x,a)$$

Markov cost process



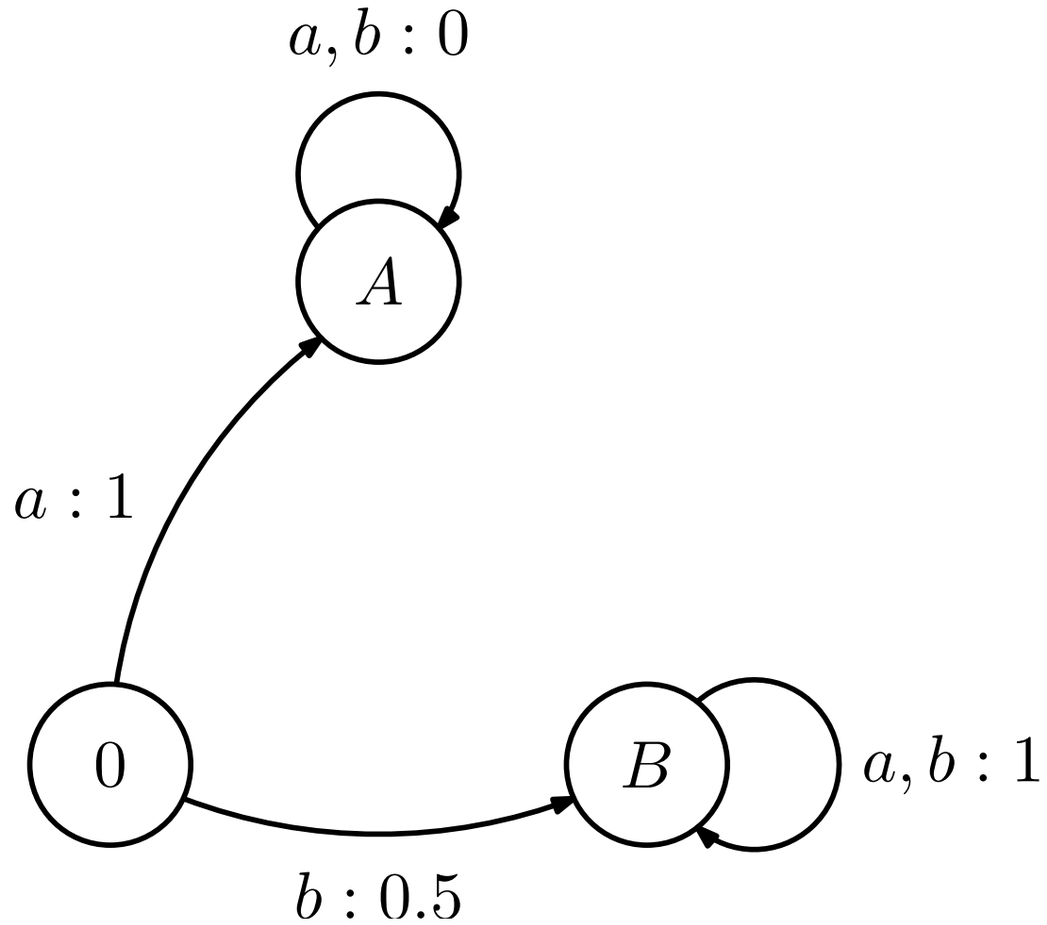
Examples



The 3-state MDP

Example

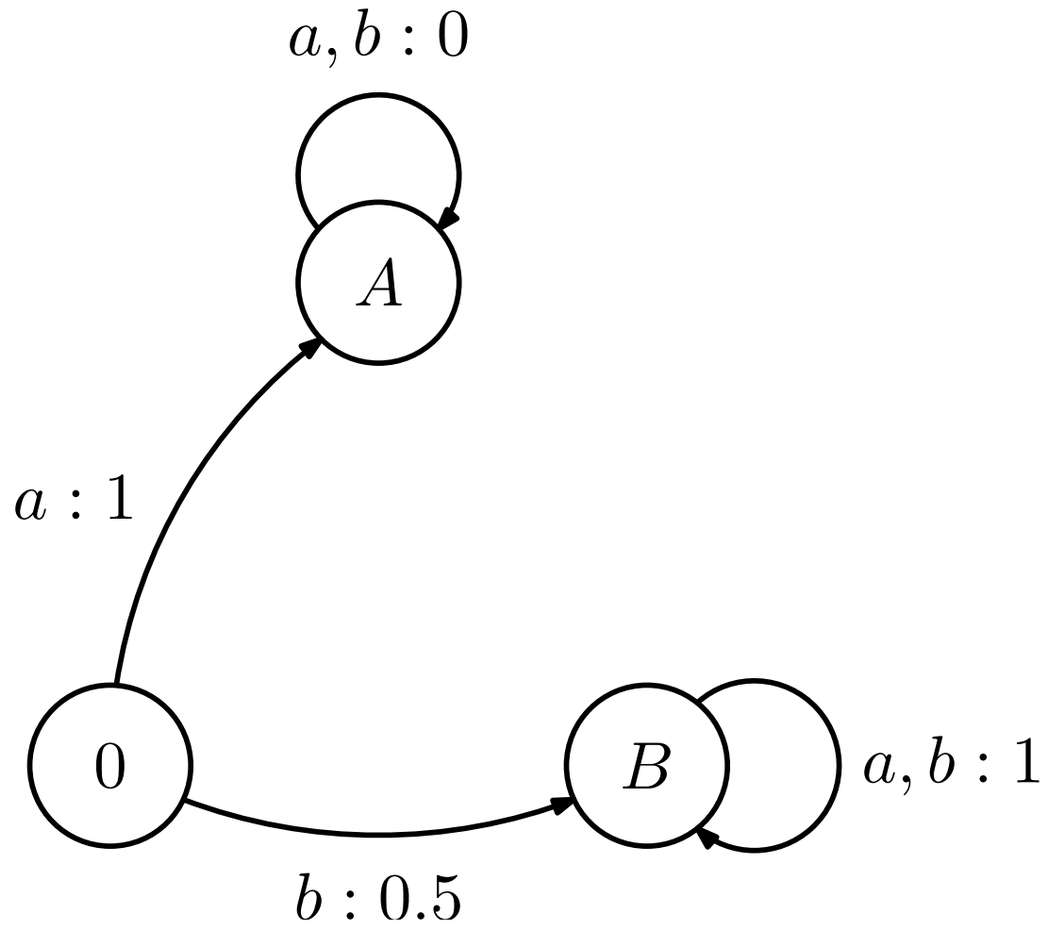
- Let us consider the policy π that:
  - Selects action a in A
  - Selects action b in B
  - Selects uniformly between a and b in 0



Example

- Policy π:

$$\pi = \begin{bmatrix} 0.5 & 0.5 \\ 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$$



Example

- What is $\mathbf{P}_\pi$?

$$\mathbf{P}_a = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{P}_b = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiply

Multiply

$$\pi = \begin{bmatrix} 0.5 & 0.5 \\ 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix} \qquad \mathbf{P}_\pi = \begin{bmatrix} & \\ & \\ & \end{bmatrix}$$

Example

- What is $\mathbf{P}_\pi$?

$$\mathbf{P}_a = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{P}_b = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiply

Multiply

$$\pi = \begin{bmatrix} 0.5 & 0.5 \\ 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix} \qquad \mathbf{P}_\pi = \begin{bmatrix} 0.0 & 0.5 & 0.5 \\ & & \end{bmatrix}$$

Example

- What is $\mathbf{P}_\pi$?

$$\mathbf{P}_a = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{P}_b = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiply

Multiply

$$\pi = \begin{bmatrix} 0.5 & 0.5 \\ 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix} \qquad \mathbf{P}_\pi = \begin{bmatrix} 0.0 & 0.5 & 0.5 \\ 0.0 & 1.0 & 0.0 \\ & & \end{bmatrix}$$

Example

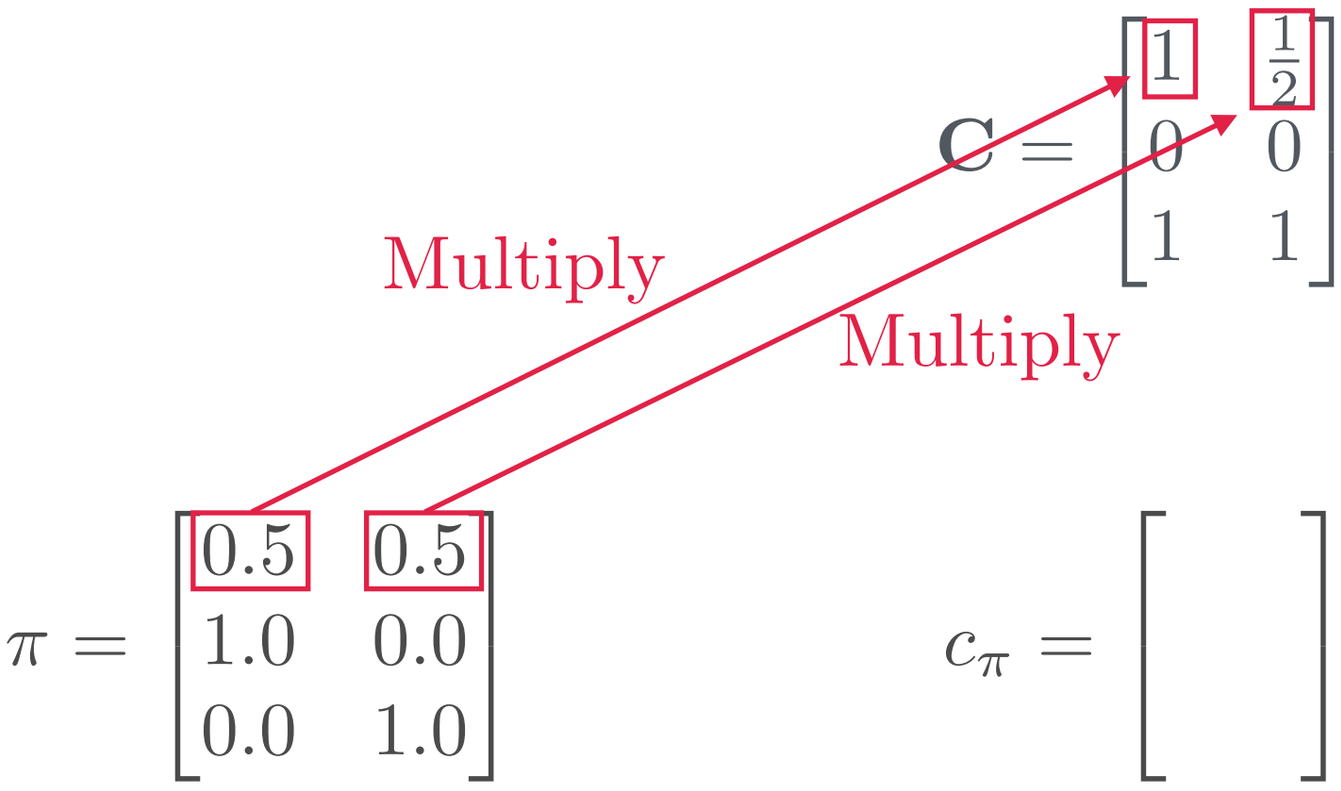
- What is $\mathbf{P}_\pi$?

$$\mathbf{P}_a = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{P}_b = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\pi = \begin{bmatrix} 0.5 & 0.5 \\ 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix} \qquad \mathbf{P}_\pi = \begin{bmatrix} 0.0 & 0.5 & 0.5 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Example

- What is c_π?



Example

- What is c_π?

$$\mathbf{C} = \begin{bmatrix} 1 & \frac{1}{2} \\ \boxed{0} & \boxed{0} \\ 1 & 1 \end{bmatrix}$$

Multiply

Multiply

$$\pi = \begin{bmatrix} 0.5 & 0.5 \\ \boxed{1.0} & \boxed{0.0} \\ 0.0 & 1.0 \end{bmatrix} \qquad c_\pi = \begin{bmatrix} 0.75 \\ \\ \end{bmatrix}$$

Example

- What is c_π?

$$\mathbf{C} = \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & 0 \\ \boxed{1} & \boxed{1} \end{bmatrix}$$

Multiply

Multiply

$$\pi = \begin{bmatrix} 0.5 & 0.5 \\ 1.0 & 0.0 \\ \boxed{0.0} & \boxed{1.0} \end{bmatrix} \qquad c_\pi = \begin{bmatrix} 0.75 \\ 0.0 \\ \end{bmatrix}$$

Example

- What is c_π?

$$\mathbf{C} = \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\pi = \begin{bmatrix} 0.5 & 0.5 \\ 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix} \qquad c_\pi = \begin{bmatrix} 0.75 \\ 0.0 \\ 1.0 \end{bmatrix}$$

How do we use Markov decision processes to make decisions?

Optimality

Cost-to-go function

- Imagine the following experiment:
  - Fix a (stationary) policy π
  - Start the agent in some state x
  - Let the agent go
  - Keep track of all costs to pay

Cost-to-go function

- How much will the agent pay (in average)?

$$\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t c_t\right]$$

Discounted cost-to-go

Expectation (the "in average")

Cost-to-go function

- How much will the agent pay (in average)?
  - Depends on the initial state x

$$\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \mathrm{c}_t \mid \mathrm{x}_0 = x\right]$$

Initial state

Cost-to-go function

- How much will the agent pay (in average)?
  - Depends on the initial state x
  - Depends on the policy π

$$\mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}\mathrm{c}_{t}\mid \mathrm{x}_{0}=x\right]$$

Policy

Cost-to-go function

- How much will the agent pay (in average)?
  - Depends on the initial state x
  - Depends on the policy π

$$J^{\pi}(x) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_t \mid \mathrm{x}_0 = x\right]$$

Cost-to-go
at state x
for policy π

Cost-to-go function

- J^π maps each state in $\mathcal{X}$ to a number (the discounted cost-to-go)
- J^π is the **cost-to-go function** associated with policy π

Optimality

- A policy π^* is optimal if

$$J^{\pi^*}(x) \leq J^{\pi}(x)$$

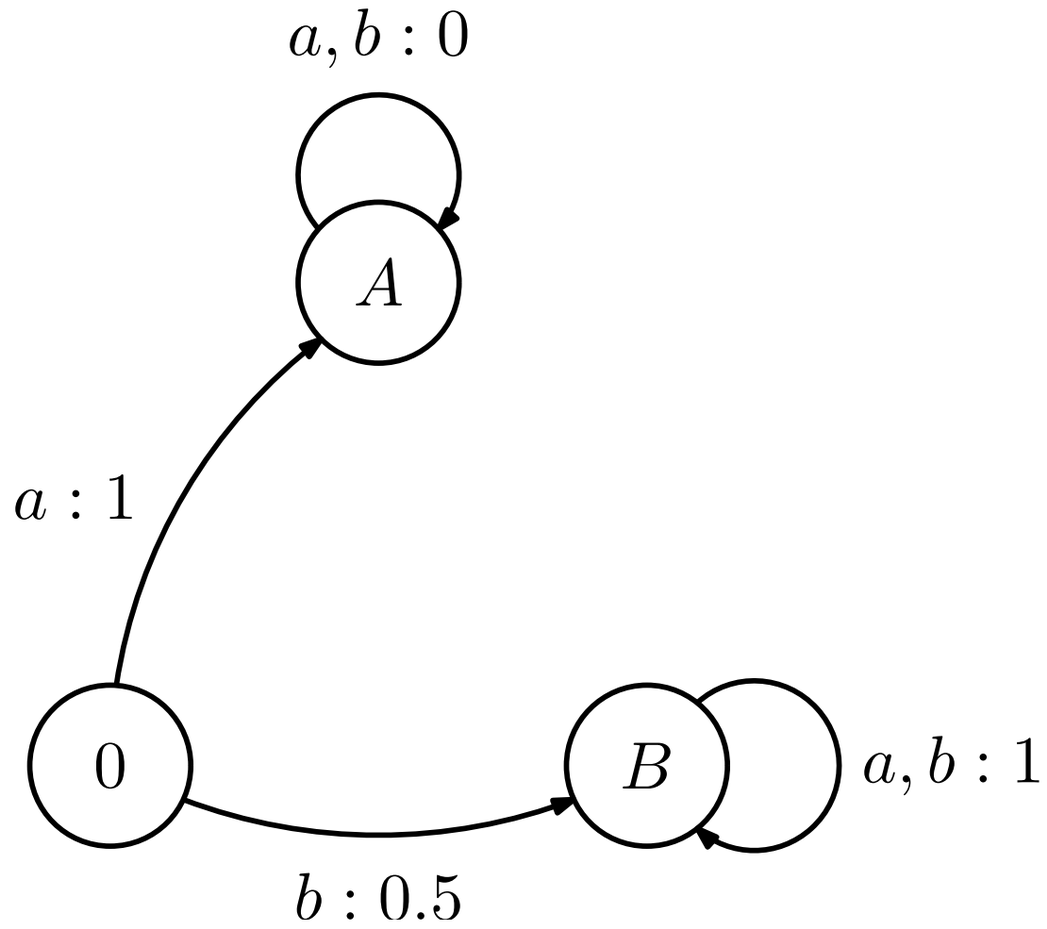
for all states

- In other words:

No matter where you start, the agent cannot attain a better value by following any other policy

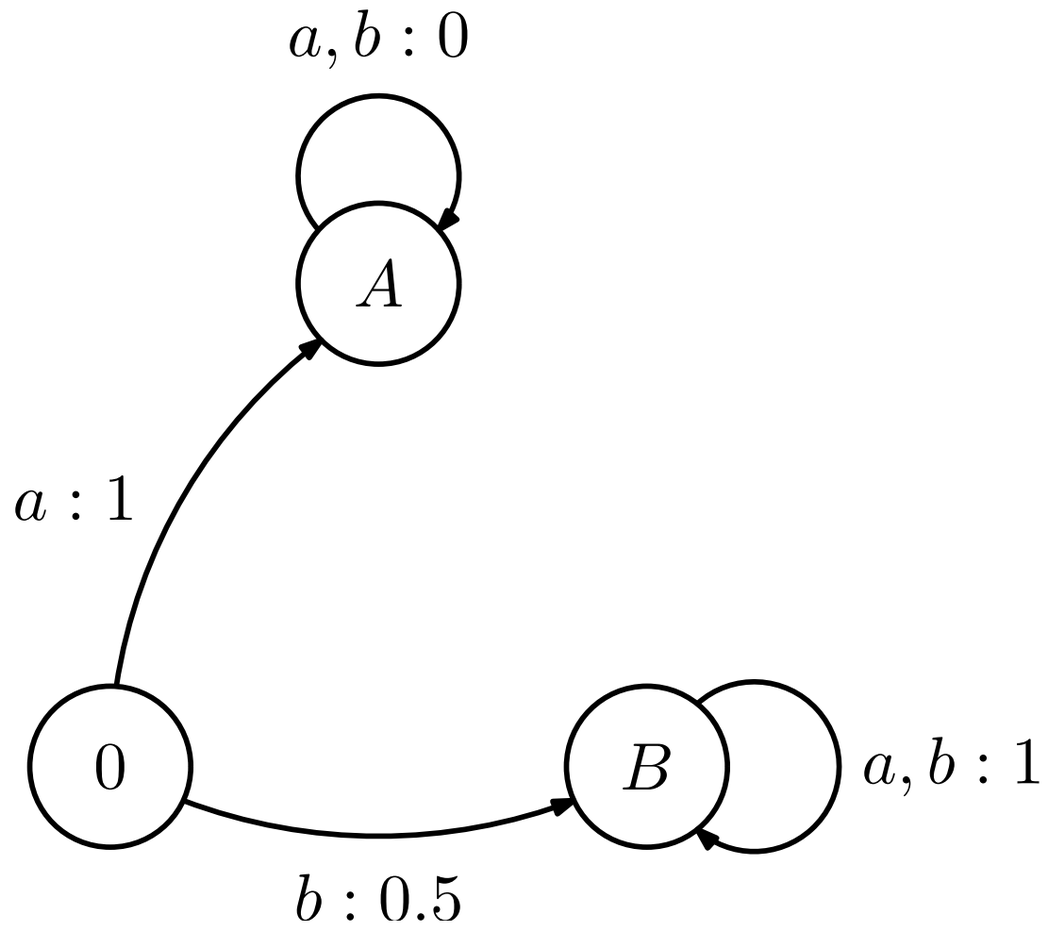
Example

- Consider the example:



Example

- We compare two policies:
  - A policy π that always selects action a
  - A policy π' that always selects action b



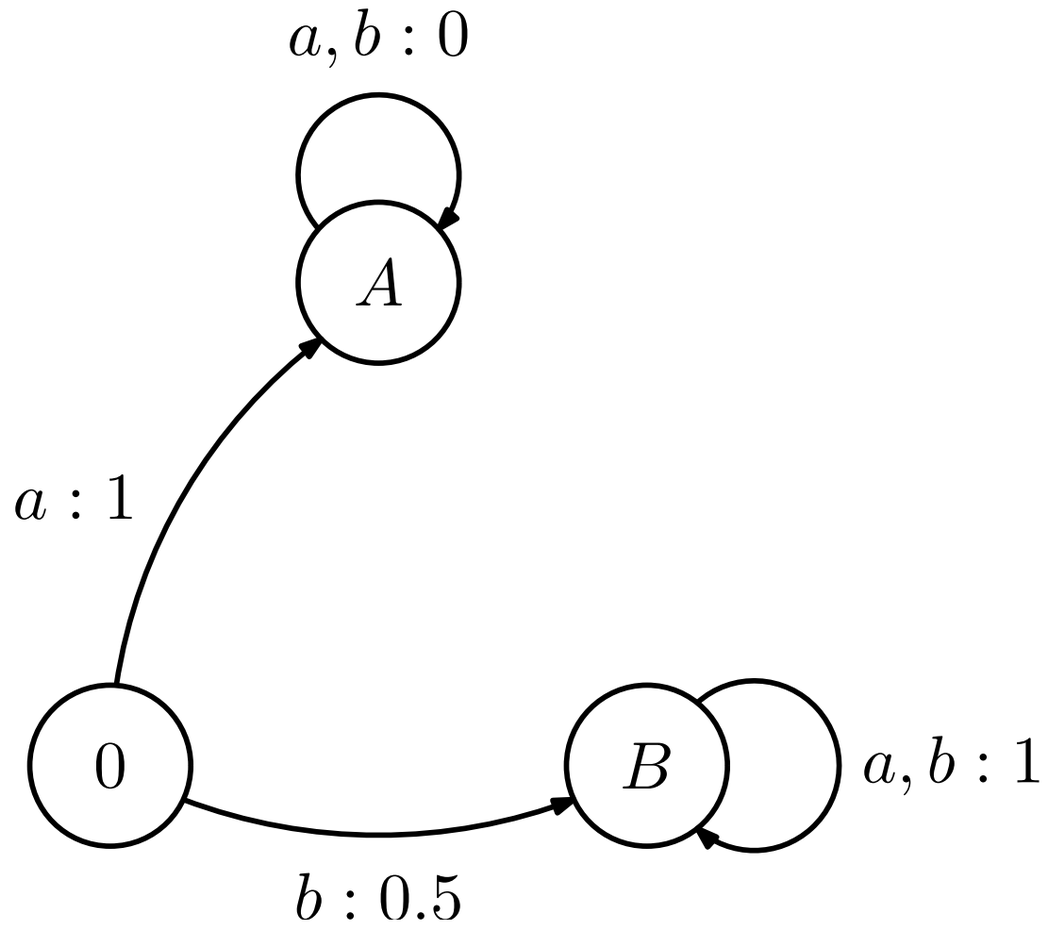
Example

- Policy π (always selects a):

$$\mathbf{J}^{\pi} = \begin{bmatrix} 1 \\ 0 \\ \frac{1}{1-\gamma} \end{bmatrix}$$

- Policy π' (always selects b):

$$\mathbf{J}^{\pi'} = \begin{bmatrix} \frac{1}{2} \cdot \frac{1+\gamma}{1-\gamma} \\ 0 \\ \frac{1}{1-\gamma} \end{bmatrix}$$



Example

- Policy π is better if

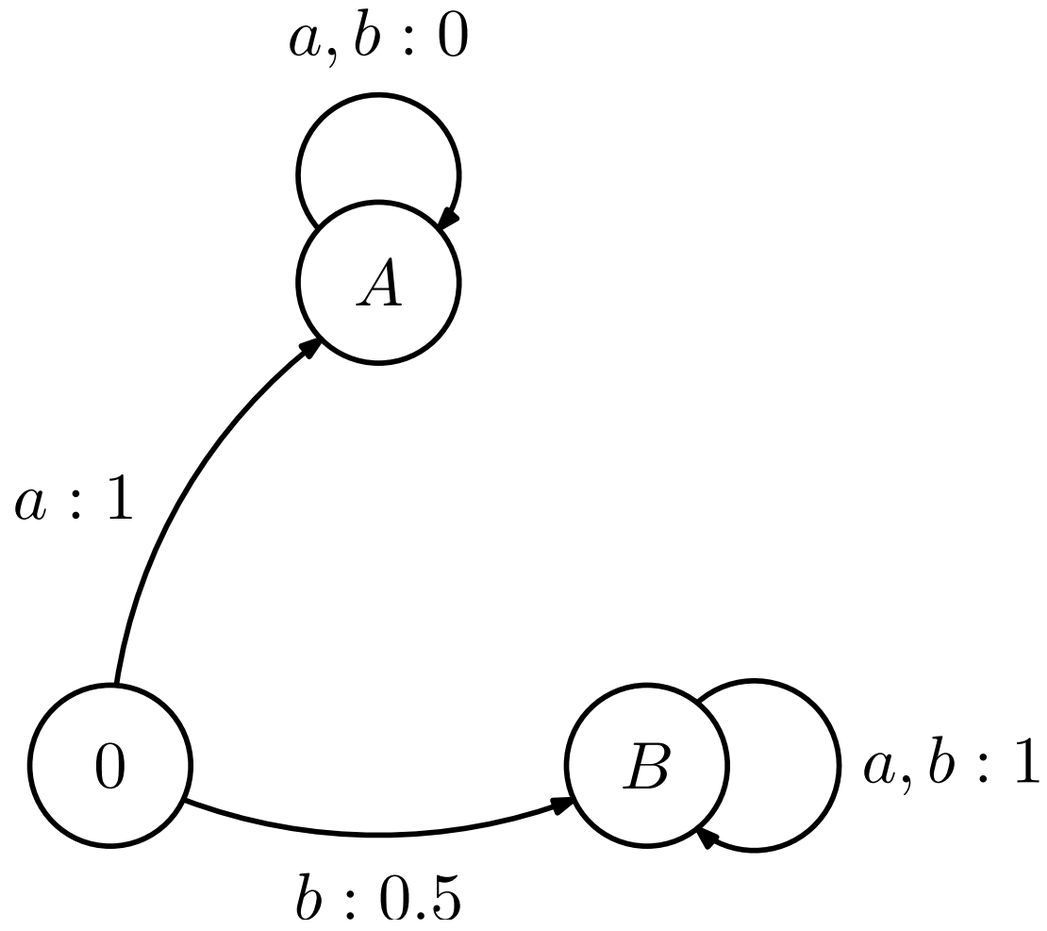
$$J^{\pi}(0) < J^{\pi'}(0)$$

- Equivalently, if

$$1 < \frac{1}{2} \cdot \frac{1+\gamma}{1-\gamma}$$

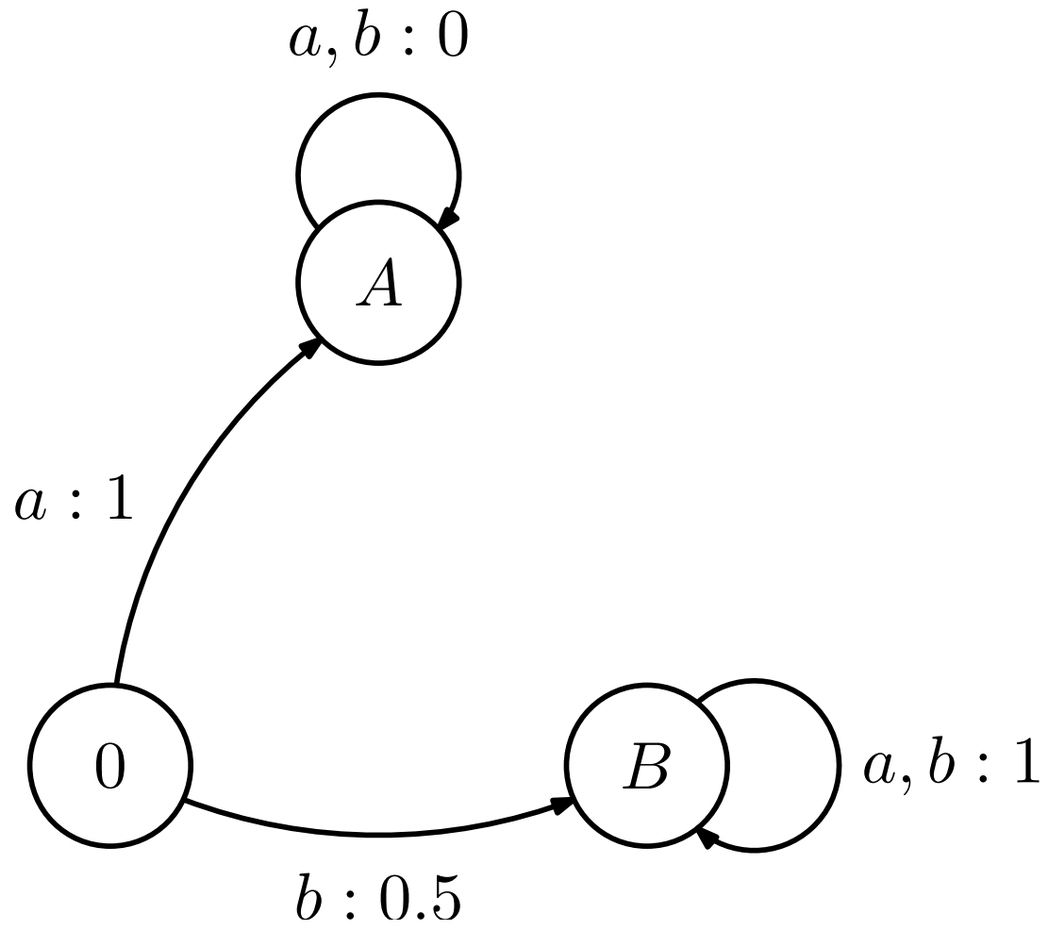
or

$$\gamma > \frac{1}{3}$$



Example

- For example, if $\gamma = 0.99$,
  - $J^{\pi}(0) = 1$
  - $J^{\pi'}(0) = 99.5$



Existence of optimal policies

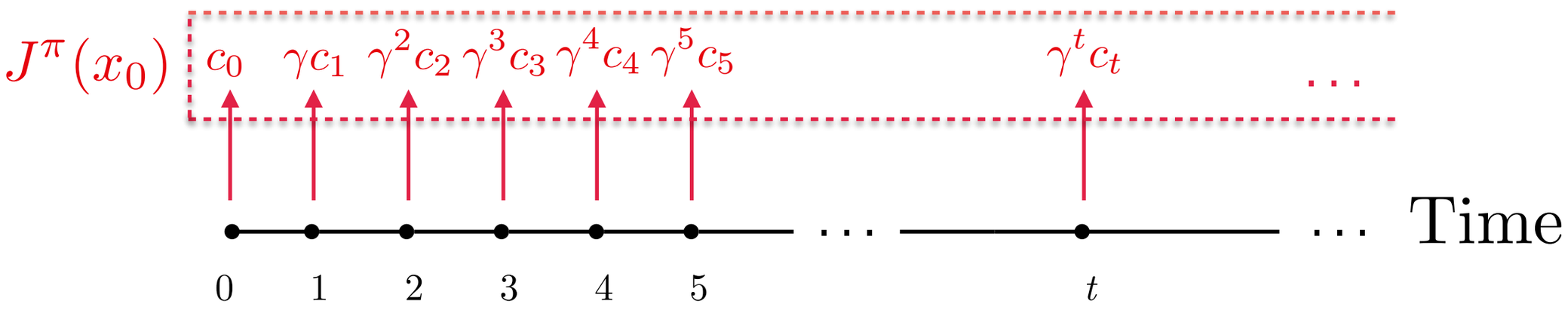
Does an optimal policy exist??

If so, how can we compute it?

A simpler problem

- Fix some stationary policy π
- What is the cost to go, J^π? ← Prediction problem

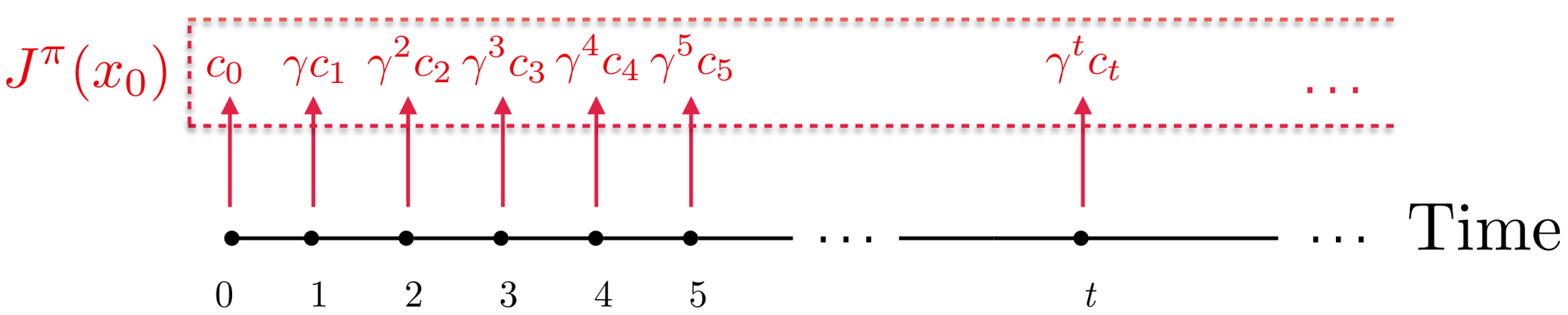
$$J^\pi(x_0) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_t \mid \mathrm{x}_0 = x_0 \right]$$



A simpler problem

- Fix some stationary policy π
- What is the cost to go, J^π?

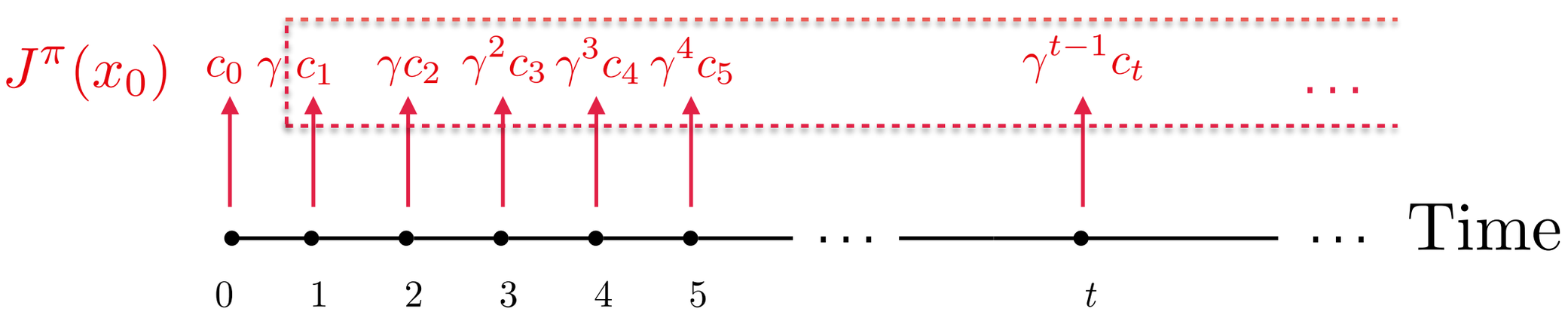
$$J^\pi(x_0) = \mathbb{E}_\pi \left[\mathrm{c}_0 + \gamma \mathrm{c}_1 + \gamma^2 \mathrm{c}_2 + \ldots \mid \mathrm{x}_0 = x_0 \right]$$



A simpler problem

- Fix some stationary policy π
- What is the cost to go, J^π?

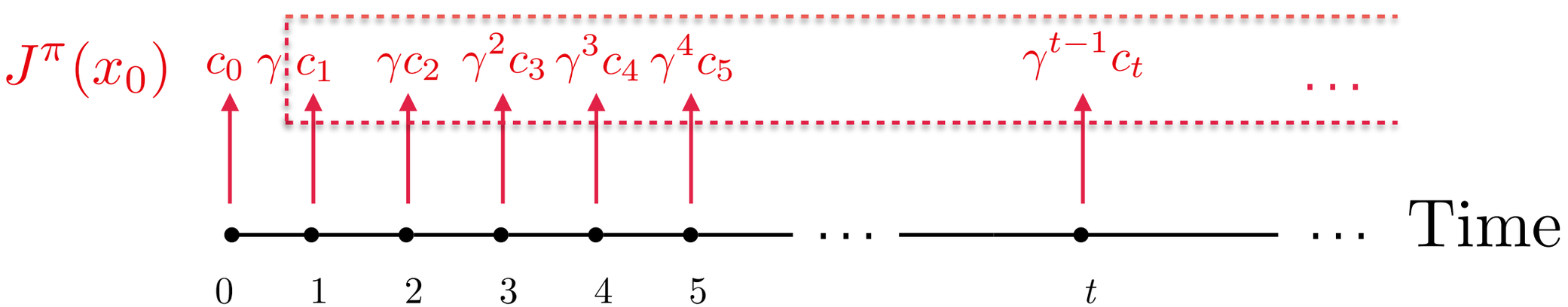
$$J^\pi(x_0) = \mathbb{E}_\pi \left[\mathrm{c}_0 + \gamma(\mathrm{c}_1 + \gamma \mathrm{c}_2 + \ldots) \mid \mathrm{x}_0 = x_0 \right]$$



A simpler problem

- Fix some stationary policy π
- What is the cost to go, J^π?

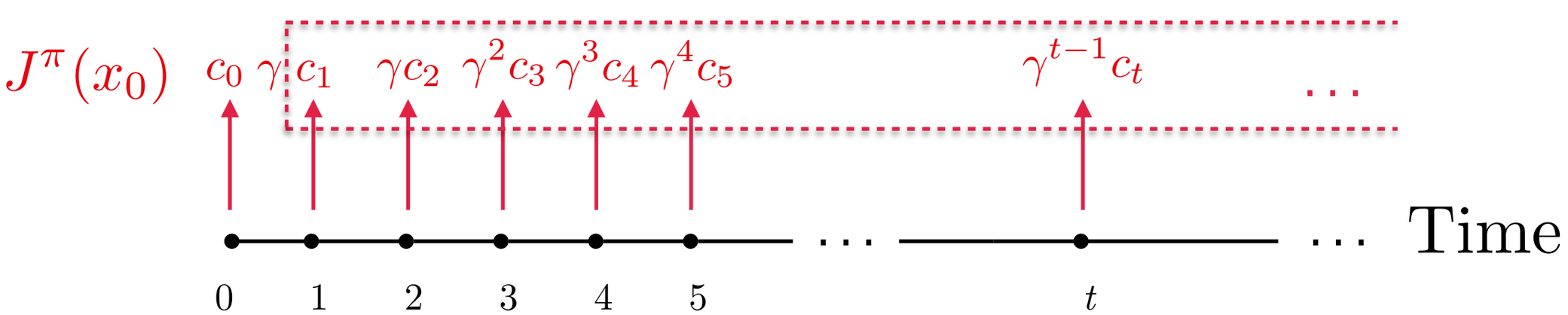
$$J^\pi(x_0) = \underbrace{\mathbb{E}_\pi\left[\mathrm{c}_0 \mid \mathrm{x}_0 = x_0\right]}_{c_\pi(x_0)} + \gamma \mathbb{E}_\pi\left[\mathrm{c}_1 + \gamma \mathrm{c}_2 + \ldots \mid \mathrm{x}_0 = x_0\right]$$



A simpler problem

- Fix some stationary policy π
- What is the cost to go, J^π?

$$J^\pi(x_0) = c_\pi(x_0) + \gamma \mathbb{E}_\pi \left[\mathrm{c}_1 + \gamma \mathrm{c}_2 + \ldots \mid \mathrm{x}_0 = x_0 \right]$$

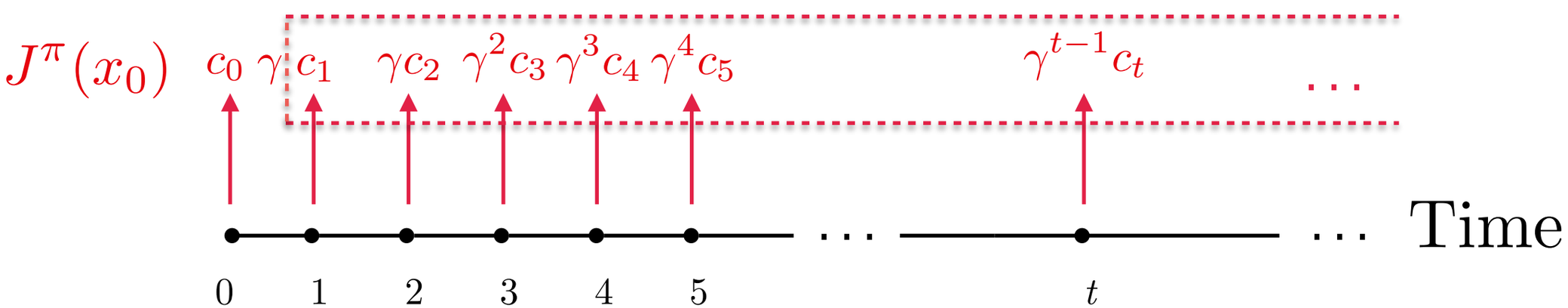


A simpler problem

- Fix some stationary policy π
- What is the cost to go, J^π?

$$J^\pi(x_0) = c_\pi(x_0) + \gamma \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_{t+1} \mid \mathrm{x}_0 = x_0 \right]$$

This also looks like a cost-to-go function



A simpler problem

- Fix some stationary policy π
- What is the cost to go, J^π?

$$J^\pi(x_0) = c_\pi(x_0) + \gamma \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_{t+1} \mid \mathrm{x}_0 = x_0 \right]$$

First cost is c_1

We'd like to condition on x_1

Total probability law

A simpler problem

- Fix some stationary policy π
- What is the cost to go, J^π?

$$J^\pi(x_0) = c_\pi(x_0) + \gamma \sum_{x \in \mathcal{X}} \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_{t+1} \mid \mathrm{x}_1 = x, \mathrm{x}_0 = x_0 \right] \mathbb{P}_\pi \left[\mathrm{x}_1 = x \mid \mathrm{x}_0 = x_0 \right]$$

$$\mathbf{P}_\pi(x \mid x_0)$$

We can ignore x_0
(Markov property)

A simpler problem

- Fix some stationary policy π
- What is the cost to go, J^π?

$$J^\pi(x_0) = c_\pi(x_0) + \gamma \sum_{x \in \mathcal{X}} \underbrace{\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_{t+1} \mid \mathrm{x}_1 = x \right]}_{J^\pi(x)} \mathbf{P}_\pi(x \mid x_0)$$

A simpler problem

- Fix some stationary policy π
- What is the cost to go, J^π?

$$J^\pi(x_0) = c_\pi(x_0) + \gamma \sum_{x \in \mathcal{X}} J^\pi(x) \mathbf{P}_\pi(x \mid x_0)$$

A simpler problem

- Fix some stationary policy π
- What is the cost to go, J^π?

$$J^\pi(x_0) = c_\pi(x_0) + \gamma \sum_{x \in \mathcal{X}} \mathbf{P}_\pi(x \mid x_0) J^\pi(x)$$

The prediction problem

- Fix some stationary policy π
- The cost-to-go function, J^π, verifies

$$J^\pi(x_0) = c_\pi(x_0) + \gamma \sum_{x \in \mathcal{X}} \mathbf{P}_\pi(x \mid x_0) J^\pi(x)$$

- Using vector notation...

$$\mathbf{J}^\pi = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{J}^\pi$$

Vector Vector Matrix Vector

Computing $\mathbf{J}^\pi$

- We can easily solve the system

$$\mathbf{J}^\pi = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{J}^\pi$$

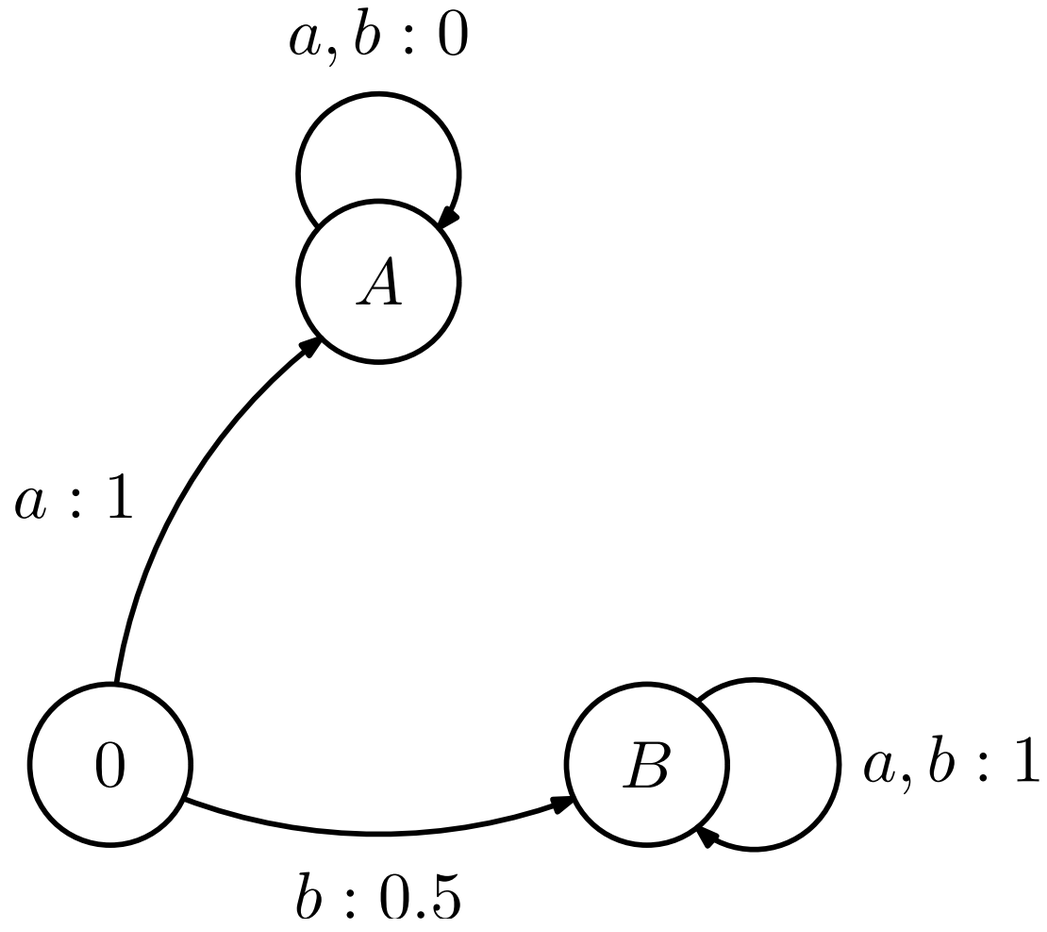
- We get

$$\mathbf{J}^\pi = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \mathbf{c}_\pi$$

Example

- Consider the policy π that always selects a
- What is c_π?

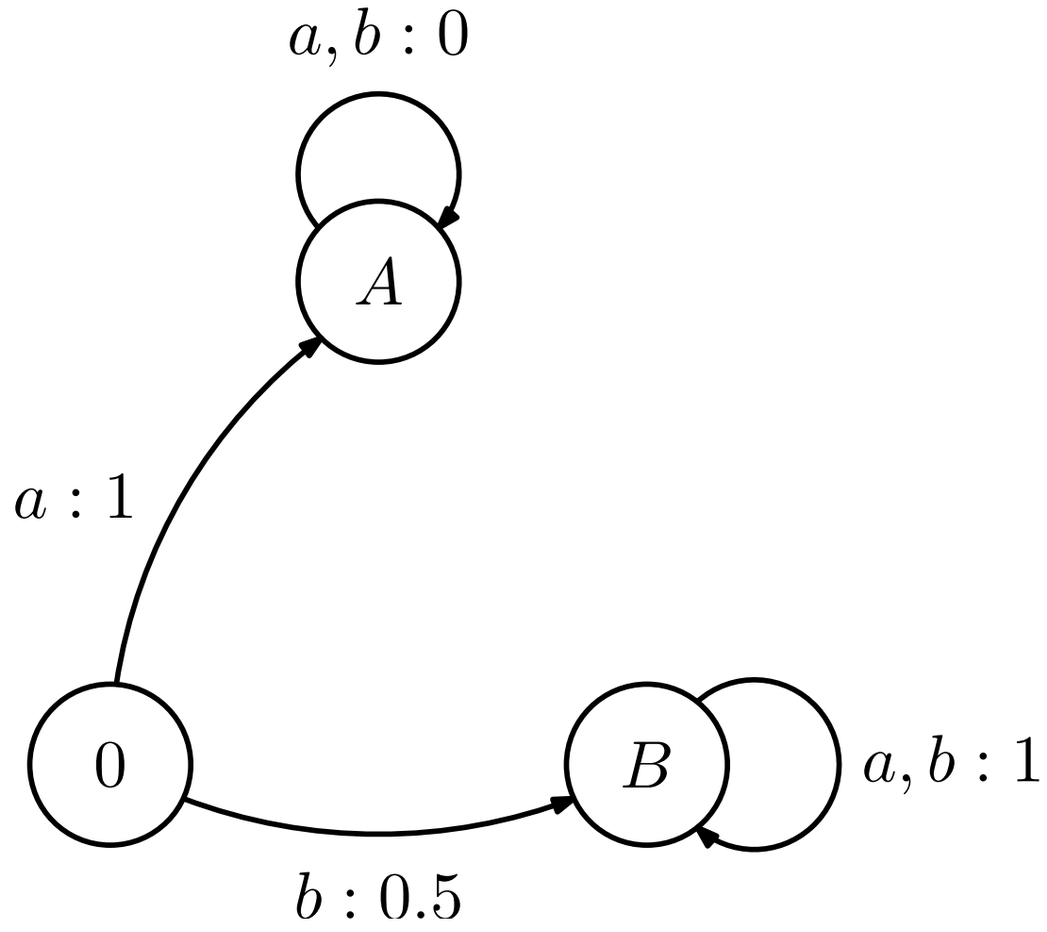
$$\boldsymbol{c}_\pi = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$



Example

- Consider the policy π that always selects a
- What is $\mathbf{P}_\pi$?

$$\mathbf{P}_\pi = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$



Example

- We have

$$\mathbf{J}^{\pi} = (\mathbf{I} - \gamma \mathbf{P}_{\pi})^{-1} \mathbf{c}_{\pi}$$

$$= \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - 0.99 \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Example

- We have

$$\mathbf{J}^{\pi} = (\mathbf{I} - \gamma \mathbf{P}_{\pi})^{-1} \mathbf{c}_{\pi}$$

$$= \begin{bmatrix} 1 & -0.99 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Example

- We have

$$\mathbf{J}^{\pi} = (\mathbf{I} - \gamma \mathbf{P}_{\pi})^{-1} \mathbf{c}_{\pi}$$

$$= \begin{bmatrix} 1 & 99 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 0 \\ 100 \end{bmatrix}$$

Values computed before!

Is this efficient?

- For large problems, matrix inversion is inefficient
- Alternatively, we can use the recursive expression:

$$J^\pi(x) = \underbrace{c_\pi(x) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_\pi(y \mid x) J^\pi(y)}_{\mathbf{T}_\pi}$$

- Operator $\mathbf{T}_\pi$ transforms Js into Js:

$$\mathbf{T}_\pi \mathbf{J} = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{J}$$

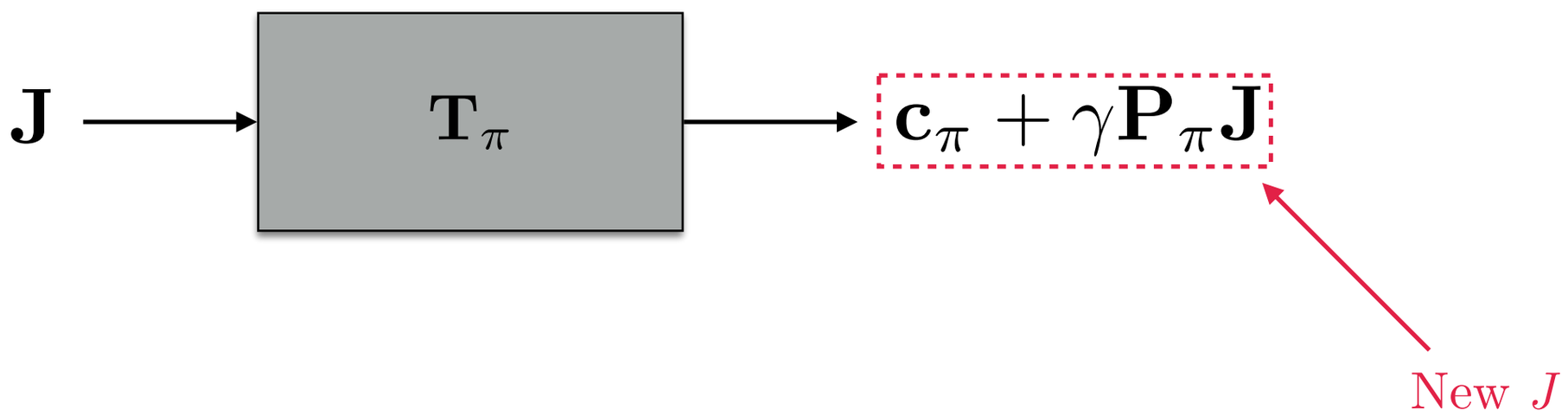
Arbitrary J

Is this efficient?

- For large problems, matrix inversion is inefficient
- Alternatively, we can use the recursive expression:

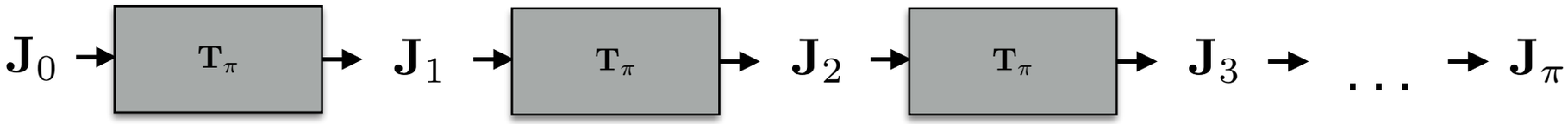
$$J^\pi(x) = \underbrace{c_\pi(x) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_\pi(y \mid x) J^\pi(y)}_{\mathbf{T}_\pi}$$

- Operator $\mathbf{T}_\pi$ transforms Js into Js:



Is this efficient?

- We get an iterative approach:



- This is a **dynamic programming approach** called **value iteration**

Value Iteration, 1.0

- We get an iterative approach:

Input: MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \{\mathbf{P}_a\}, c, \gamma)$

Input: Tolerance $\epsilon > 0$

Input: Stationary policy π

Initialize $k = 0$

Initialize $\mathbf{J}_0 = \mathbf{0}$

repeat

    $\mathbf{J}_{k+1} = \mathbf{T}_\pi \mathbf{J}_k$

    $k = k + 1$

until $\|\mathbf{J}_k - \mathbf{J}_{k-1}\| < \epsilon$

return $\mathbf{J}_k$

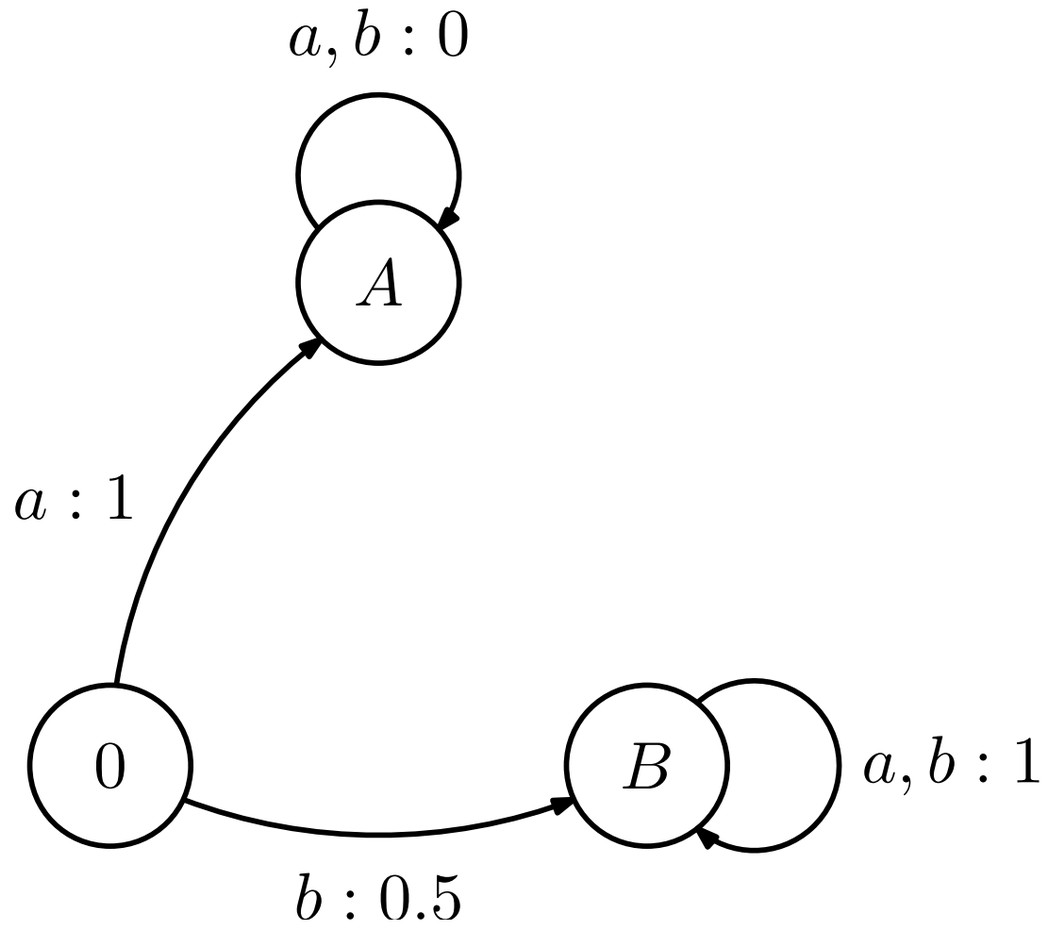
- This is a **dynamic programming approach** called **value iteration**

Example

- We again consider the policy that always selects a

$$\boldsymbol{c}_\pi = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathbf{P}_\pi = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Example

- We have

$$\mathbf{J}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Example

- We have

$$\mathbf{J}_1 = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{J}_0$$

$$= \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + 0.99 \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Example

- We have

$$\mathbf{J}_2 = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{J}_1$$

$$= \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + 0.99 \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 0 \\ 1.99 \end{bmatrix}$$

... and so on.

But what about optimality??

Another recursion

- Let

$$J^*(x) = \inf_{\pi} J^{\pi}(x)$$

- The optimal policy π^*, if it exists, must be such that

$$J^{\pi^*}(x) = J^*(x)$$

Another recursion

- We now take the recursion

$$J^\pi(x) = c_\pi(x) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_\pi(y \mid x) J^\pi(y)$$

which is equivalent to

$$J^\pi(x) = \mathbb{E}_{\mathrm{a} \sim \pi(x)} \left[c(x, \mathrm{a}) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_{\mathrm{a}}(y \mid x) J^\pi(y) \right]$$

Average value when
a is selected using π

Another recursion

- ... and replace π by min:

$$J^{\pi}(x) = \mathbb{E}_{\mathrm{a}\sim\pi(x)}\left[c(x,\mathrm{a}) + \gamma \sum_{y\in\mathcal{X}} \mathbf{P}_{\mathrm{a}}(y \mid x) J^{\pi}(y)\right]$$

Another recursion

- ... and replace π by min:

$$J(x) = \min_a \left[c(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_a(y \mid x) J(y) \right]$$

Solution is no longer J^π

Does this new equation have a solution?

If so, what is it? And what is it useful for?

AMAZING RESULT - Part 1

1. The recursion

$$J(x) = \min_a \left[c(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_a(y \mid x) J(y) \right]$$

has a single solution.

AMAZING RESULT - Part 2

2. The solution to the recursion

$$J(x) = \min_a \left[c(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_a(y \mid x) J(y) \right]$$

is J^*.

AMAZING RESULT - Part 3

3. The action that minimizes the rhs of

$$J^*(x) = \min_a \left[c(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_a(y \mid x) J^*(y) \right]$$

is **optimal** in state x. In other words, the policy

$$\pi(x) = \arg\min_a \left[c(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_a(y \mid x) J^*(y) \right]$$

is optimal.

There is a **deterministic** and **stationary** optimal policy!

Value iteration 2.0

- We can use the recursive expression:

$$J^*(x) = \min_a \Big[c(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_a(y \mid x) J^*(y) \Big] \quad \mathbf{T}$$

- Operator $\mathbf{T}$ transforms arbitrary Js into new Js:

$$\mathbf{TJ} = \min_a \left[\mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{J} \right]$$

Arbitrary J

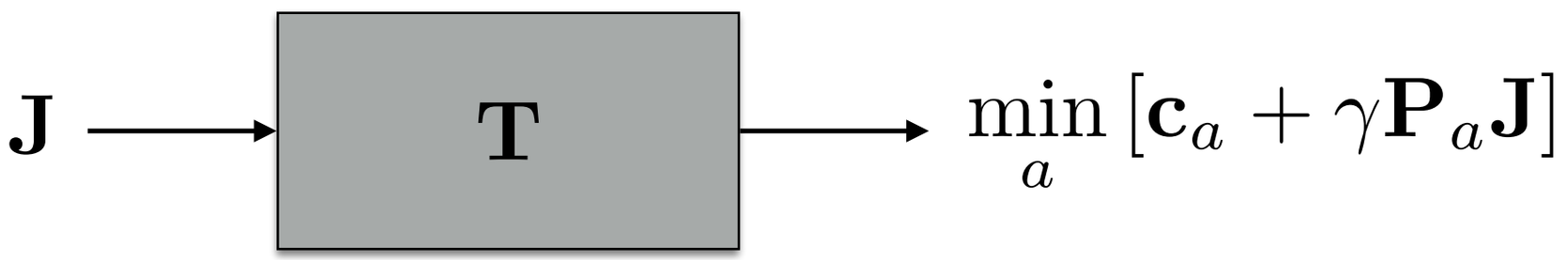
Column vector

Value iteration 2.0

- We can use the recursive expression:

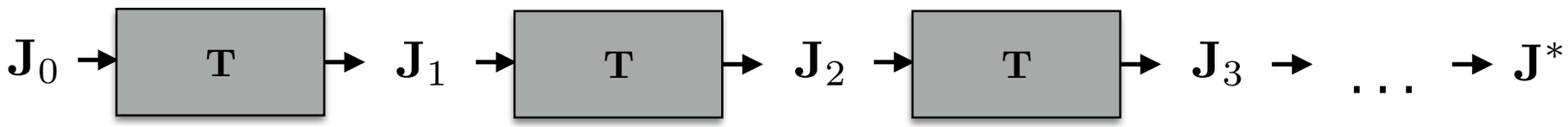
$$J^*(x) = \min_a \Big[c(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_a(y \mid x) J^*(y) \Big] \quad \mathbf{T}$$

- Operator $\mathbf{T}$ transforms arbitrary Js into new Js:



Value iteration 2.0

- We get an iterative approach:



- This is also a **dynamic programming approach**, and also called **value iteration** (this time to compute J^*)

Value iteration 2.0

- We get an iterative approach:

Input: MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \{\mathbf{P}_a\}, c, \gamma)$

Input: Tolerance $\epsilon > 0$

Initialize $k = 0$

Initialize $\mathbf{J}_0 = \mathbf{0}$

repeat

  $\mathbf{J}_{k+1} = \mathbf{T}\mathbf{J}_k$

  $k = k + 1$

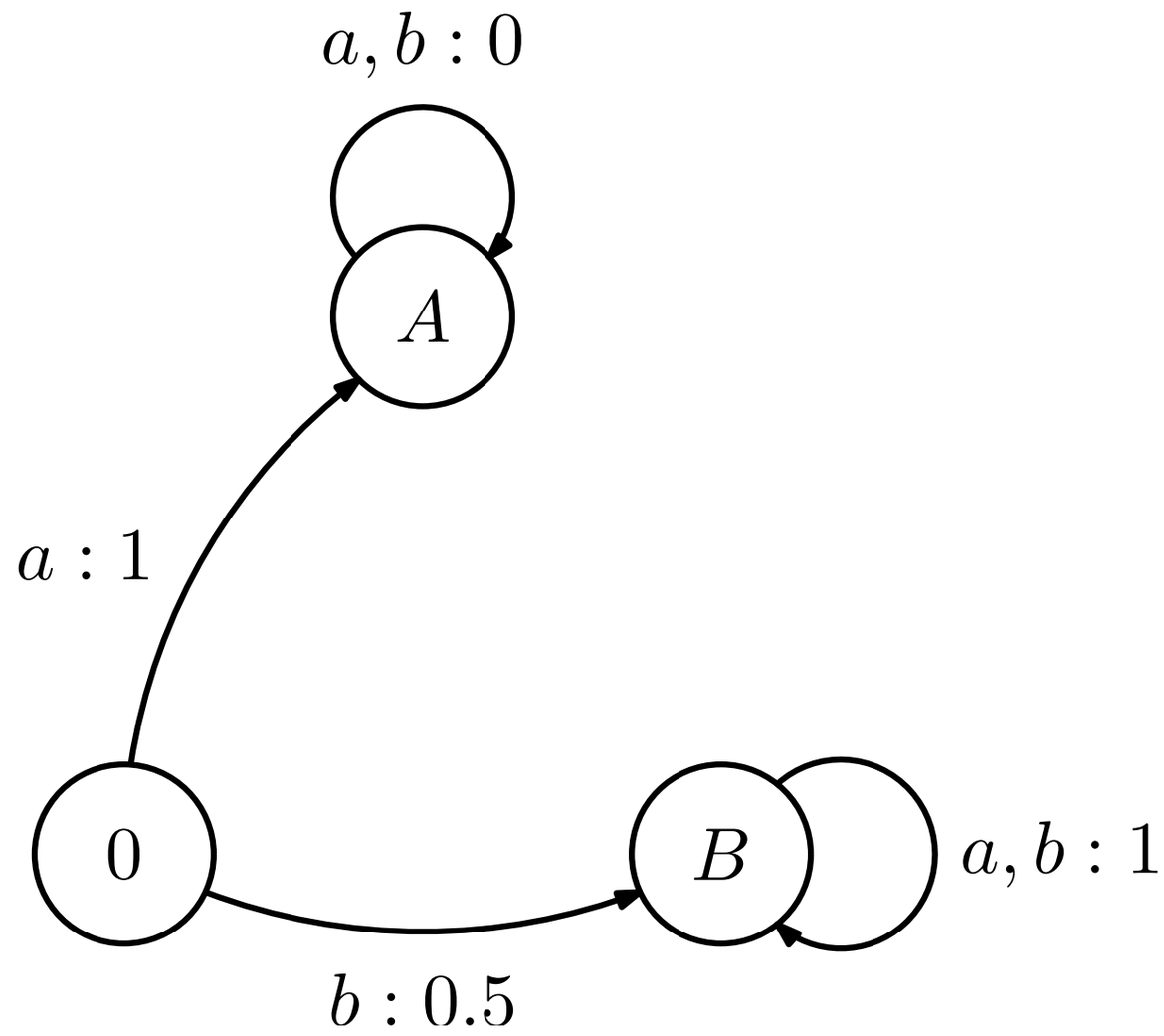
until $\|\mathbf{J}_k - \mathbf{J}_{k-1}\| < \epsilon$

return $\mathbf{J}_k$

- This is also **value iteration** (this time to compute J^*)

Example

- Let us compute the optimal policy using VI
- We start with $\mathbf{J}_0 = \mathbf{0}$

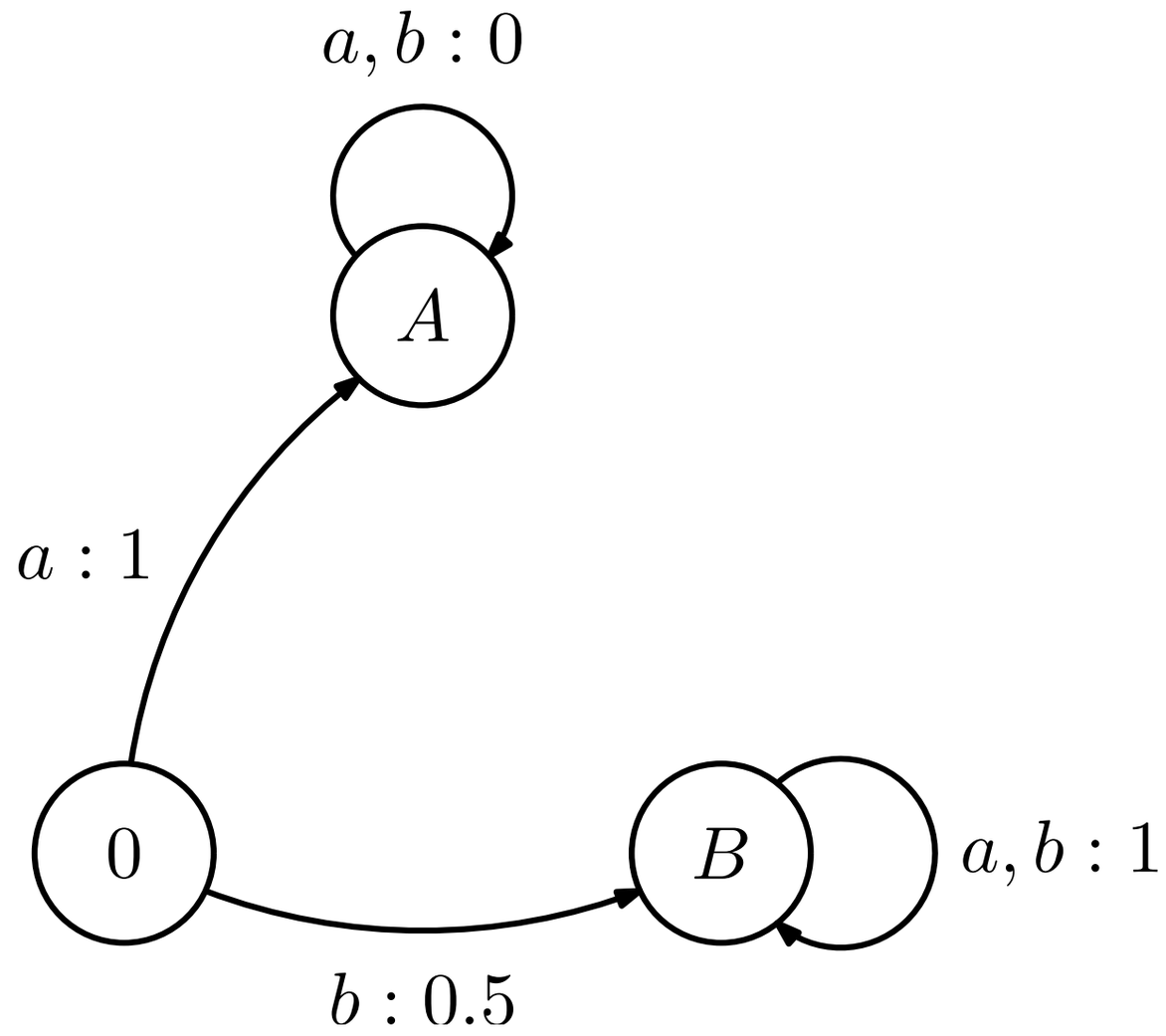


Example

- We have

$$\mathbf{J}_1 = \min_a [\mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{J}_0]$$

- We repeat the computation in square brackets for both actions



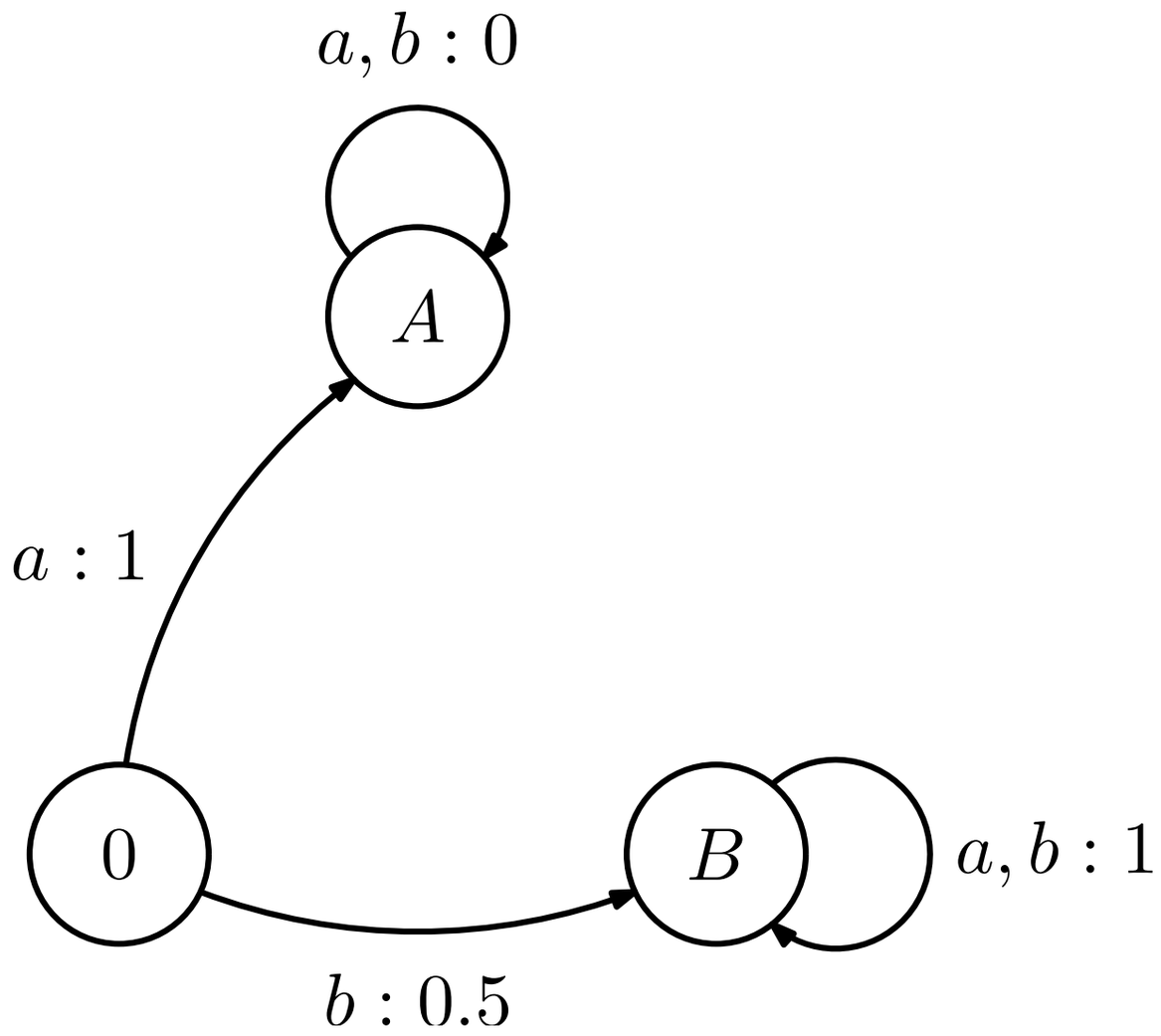
Example

- We have

$$\mathbf{J}_1 = \min_a [\mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{J}_0]$$

- For action a:

$$\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + 0.99 \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$



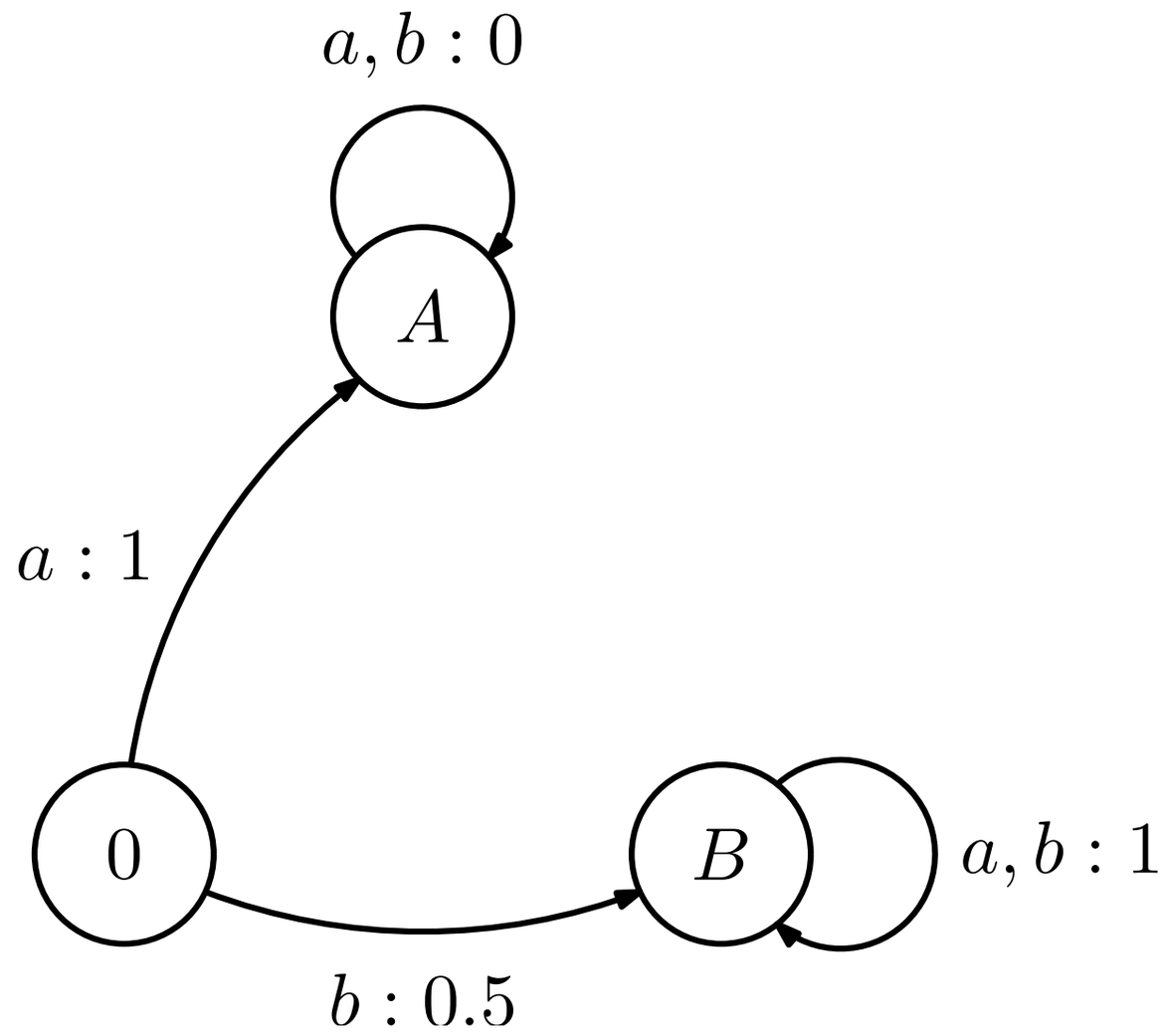
Example

- We have

$$\mathbf{J}_1 = \min_a [\mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{J}_0]$$

- ... and for action b:

$$\begin{bmatrix} 0.5 \\ 0 \\ 1 \end{bmatrix} + 0.99 \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \\ 1 \end{bmatrix}$$



Example

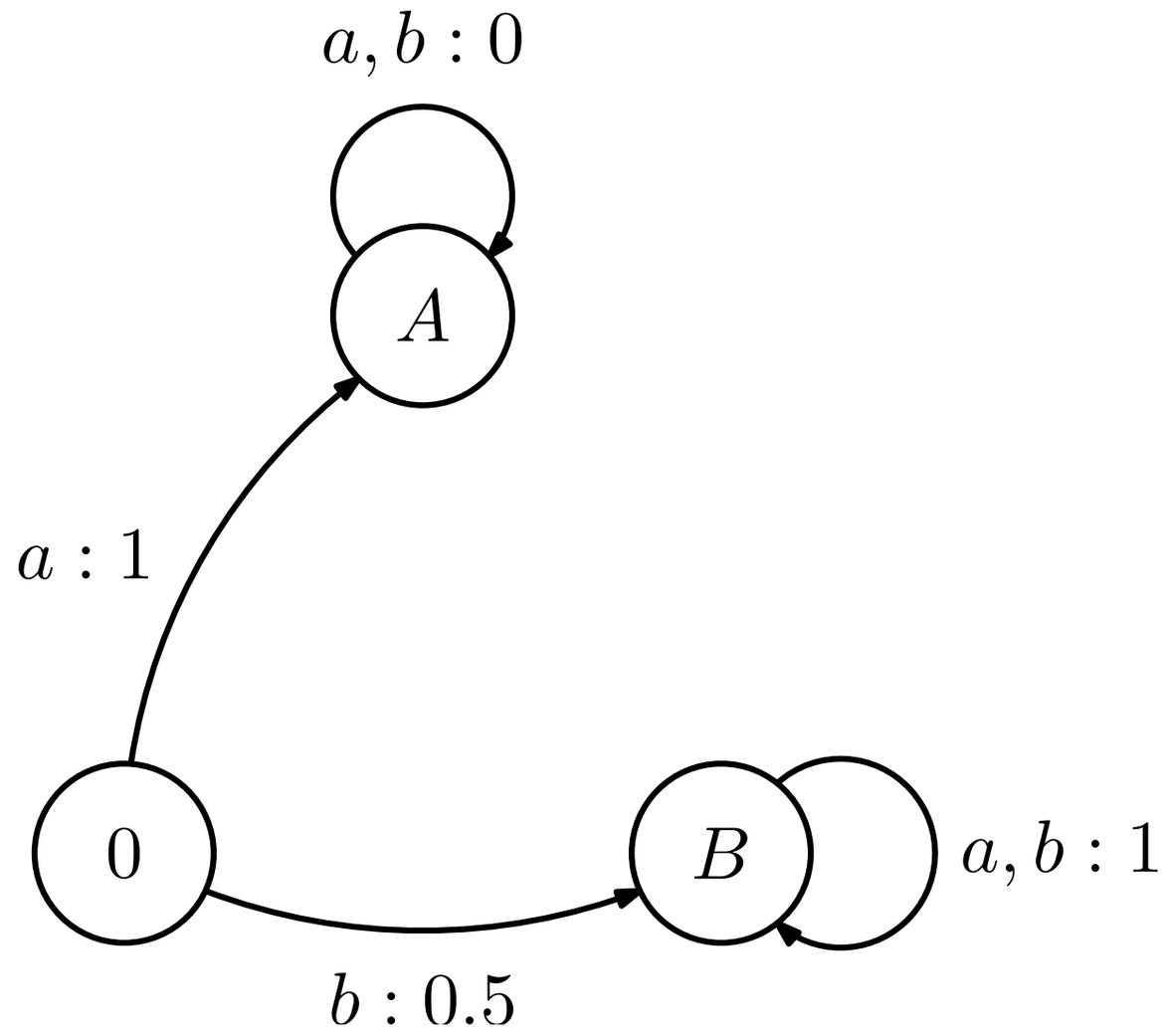
- We have

$$\mathbf{J}_1 = \min_a [\mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{J}_0]$$

- We take the minimum row-wise:

$$\min\left(\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.5 \\ 0 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 0.5 \\ 0 \\ 1 \end{bmatrix}$$

... and repeat the process.



Example

```python
import numpy as np

# Define the MDP:
X = ('0', 'A', 'B') # States
A = ('a', 'b') # Actions
Pa = np.array([[0, 1, 0], [0, 1, 0], [0, 0, 1]])
Pb = np.array([[0, 0, 1], [0, 1, 0], [0, 0, 1]])
P = (Pa, Pb) # Transition probabilities
c = np.array([[1, 0.5], [0, 0], [1, 1]])

M = (X, A, P, c, 0.99) # MDP definition

def value_iteration(M, eps):

    # These variables are just to help readibility
    X = M[0]
    A = M[1]
    P = M[2]
    c = M[3]
    gamma = M[4]

    # Initialize J with the size of state-space
    J = np.zeros((len(X), 1))
    err = 1.0
    niter = 0

    while err > eps:
        # Auxiliary array to store intermediate values
        Q = np.zeros((len(X), len(A)))

        for a in range(len(A)):
            Q[:, a, None] = c[:, a, None] + gamma * P[a].dot(J)

        # Compute minimum row-wise
        Jnew = np.min(Q, axis=1, keepdims=True)

        # Compute error
        err = np.linalg.norm(J - Jnew)

        # Update
        J = Jnew
        niter += 1

    print(f'Done after {niter} iterations.')
    return np.round(J, 3)

J = value_iteration(M, 1e-8)
print(J)
```

```
Done after 1834 iterations.
[[  1.]
 [  0.]
 [100.]]
```

We've encountered this J before...

Cost-to-go functions

- Cost-to-go functions map states to values

$$J^{\pi}(x) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_t \mid \mathrm{x}_0 = x\right]$$

$$J^{*}(x) = \min_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_t \mid \mathrm{x}_0 = x\right]$$

Cost-to-go functions

- Cost-to-go functions map states to values
- They can be used to...
  - ... evaluate policies
  - ... compute the optimal policy

... but there's more...

Other useful functions

- The cost-to-go function for a policy π verifies

$$J^\pi(x) = \mathbb{E}_{\mathrm{a}\sim\pi(x)} \underbrace{\left[c(x,\mathrm{a}) + \gamma \sum_{y\in\mathcal{X}} \mathbf{P}_{\mathrm{a}}(y \mid x) J^\pi(y) \right]}_{Q^\pi(x,a)}$$

Other useful functions

- The cost-to-go function for the optimal policy verifies

$$J^*(x) = \min_a \underbrace{\left[c(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_a(y \mid x) J^*(y) \right]}_{Q^*(x,a)}$$

Q-functions

- Q-functions map state-action pairs to values:

$$Q^{\pi}(x,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}\mathrm{c}_{t} \mid \mathrm{x}_0 = x, \mathrm{a}_0 = a\right]$$

$$Q^{*}(x,a) = \min_{\pi}\mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}\mathrm{c}_{t} \mid \mathrm{x}_0 = x, \mathrm{a}_0 = a\right]$$

Discounted cost-to-go if initial **state and action** are fixed

Q-functions

- Represent the discounted cost-to-go for a fixed policy, given the **initial state and action**

Why should we care?

Why we should care

- We can compute cost-to-go functions from Q-functions:

$$J^{\pi}(x) = \mathbb{E}_{\mathrm{a}\sim\pi(x)}\left[Q^{\pi}(x,\mathrm{a})\right]$$

$$J^{*}(x) = \min_{a} Q^{*}(x,a)$$

- We can compute the optimal policy directly from Q^*:

$$\pi^{*}(x) = \arg\min_{a} Q^{*}(x,a)$$

They act as an "informed" cost function

Bottom line: They make our life easier

How can we compute them?

More recursions...

- Since

$$Q^{\pi}(x,a) = c(x,a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_a(x' \mid x) J^{\pi}(x')$$

Replace

and

$$J^{\pi}(x) = \mathbb{E}_{\mathrm{a} \sim \pi(x)} \left[Q^{\pi}(x, \mathrm{a}) \right]$$

then

$$Q^{\pi}(x,a) = c(x,a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_a(x' \mid x) \mathbb{E}_{\mathrm{a}' \sim \pi(x')} \left[Q^{\pi}(x', \mathrm{a}') \right]$$

More recursions...

- Since

$$Q^*(x,a) = c(x,a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_a(x' \mid x) J^*(x')$$

Replace

and

$$J^*(x) = \min_a Q^*(x,a)$$

then

$$Q^*(x,a) = c(x,a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_a(x' \mid x) \min_{a'} Q^*(x',a')$$

More recursive algorithms

- We have recursive expressions for Q^π and for Q^*

$$Q^\pi(x,a) = c(x,a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_a(x' \mid x) \mathbb{E}_{\mathrm{a}' \sim \pi(x')} \left[Q^\pi(x', \mathrm{a}') \right]$$

$$Q^*(x,a) = c(x,a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_a(x' \mid x) \min_{a'} Q^*(x', a')$$

- Just as before, we turn these expressions into algorithms

Computing Q^π

- We use the recursive expression:

$$Q^\pi(x,a) = \underbrace{c(x,a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_a(x' \mid x) \mathbb{E}_{\mathrm{a}' \sim \pi(x')} \left[Q^\pi(x', \mathrm{a}') \right]}_{\mathbf{H}_\pi}$$

- Operator $\mathbf{H}_\pi$ transforms arbitrary Qs into new Qs:

$$\mathbf{H}_\pi \mathbf{Q} = \mathbf{C} + \gamma \mathbf{P} \mathbb{E}_\pi [\mathbf{Q}]$$

Arbitrary Q

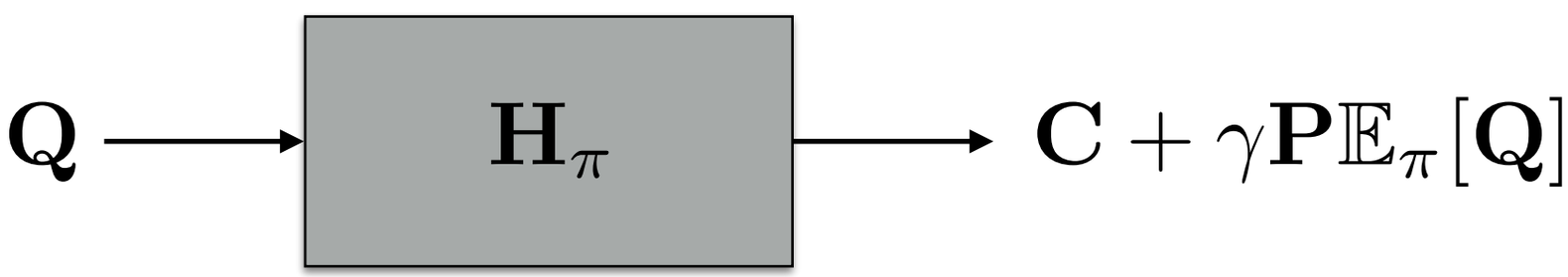
For all actions

Computing Q^π

- We use the recursive expression:

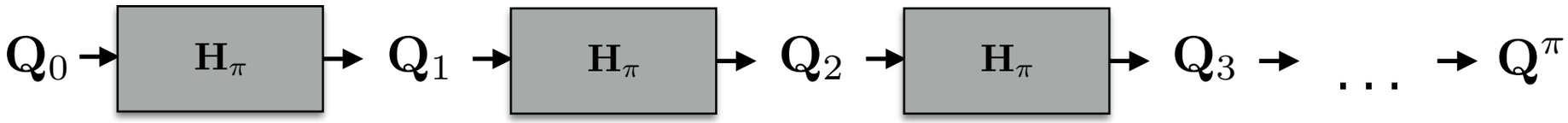
$$Q^\pi(x,a) = \underbrace{c(x,a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_a(x' \mid x) \mathbb{E}_{\mathrm{a}' \sim \pi(x')} \left[Q^\pi(x', \mathrm{a}') \right]}_{\mathbf{H}_\pi}$$

- Operator $\mathbf{H}_\pi$ transforms arbitrary Qs into new Qs:



Computing Q^π

- We get an iterative approach:



- This is also a **dynamic programming approach**, and also called **value iteration**

Value iteration 3.0

- We get an iterative approach:

Input: MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \{\mathbf{P}_a\}, c, \gamma)$

Input: Tolerance $\epsilon > 0$

Input: Stationary policy π

Initialize $k = 0$

Initialize $\mathbf{Q}_0 = \mathbf{0}$

repeat

  $\mathbf{Q}_{k+1} = \mathbf{H}_\pi \mathbf{Q}_k$

  $k = k + 1$

until $\|\mathbf{Q}_k - \mathbf{Q}_{k-1}\| < \epsilon$

return $\mathbf{Q}_k$

- This is also **value iteration**

Computing Q^*

- We use the recursive expression:

$$Q^*(x,a) = \underbrace{c(x,a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_a(x' \mid x) \min_{a'} Q^*(x',a')}_{\mathbf{H}}$$

- Operator $\mathbf{H}$ transforms arbitrary Qs into new Qs:

$$\mathbf{HQ} = \mathbf{C} + \gamma \mathbf{P} \min \mathbf{Q}$$

Arbitrary Q

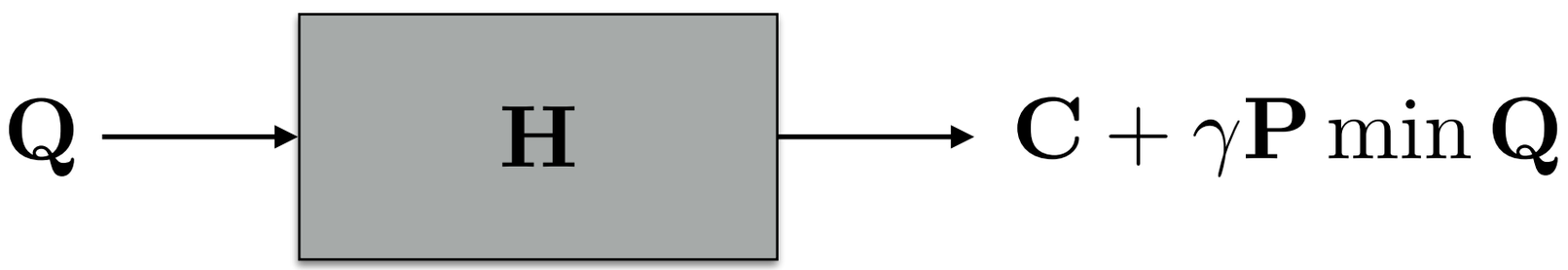
For all actions

Computing Q^*

- We use the recursive expression:

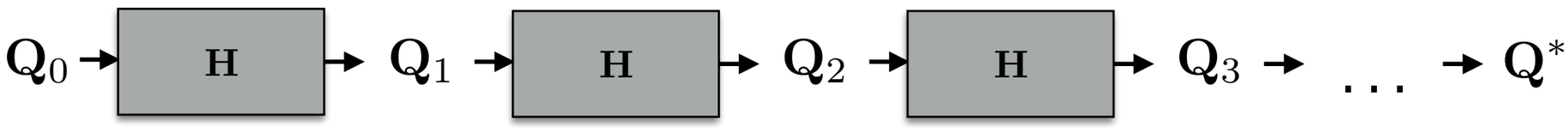
$$Q^*(x,a) = \underbrace{c(x,a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_a(x' \mid x) \min_{a'} Q^*(x',a')}_{\mathbf{H}}$$

- Operator $\mathbf{H}$ transforms arbitrary Qs into new Qs:



Computing Q^*

- We get an iterative approach:



- This is also a **dynamic programming approach**, and also called **value iteration**

Value iteration 4.0

- We get an iterative approach:

Input: MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \{\mathbf{P}_a\}, c, \gamma)$

Input: Tolerance $\epsilon > 0$

Initialize $k = 0$

Initialize $\mathbf{Q}_0 = \mathbf{0}$

repeat

  $\mathbf{Q}_{k+1} = \mathbf{H}\mathbf{Q}_k$

  $k = k + 1$

until $\|\mathbf{Q}_k - \mathbf{Q}_{k-1}\| < \epsilon$

return $\mathbf{Q}_k$

- This is also **value iteration**

Policy iteration

Value iteration

- Value iteration methods are guaranteed to converge asymptotically

Can we do better?

Our goal

- Our goal is to compute the optimal policy
- How many policies are there?

$$|\mathcal{A}|^{|\mathcal{X}|}$$

Finite number!

- If we search all, we'll finish in a finite number of iterations!

Policy iteration

- Build a sequence of policies,

$$\pi^{(0)}, \pi^{(1)}, \ldots, \pi^{(k)}, \pi^{(k+1)}, \ldots$$

  where each one is better than the previous

Greedy policy

- Given a cost-to-go function J, the greedy policy w.r.t. J is

$$\pi_g(x) = \arg\min_a \left[c(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_a(y \mid x) J(y) \right]$$

- ... for example, the optimal policy is greedy w.r.t. J^*

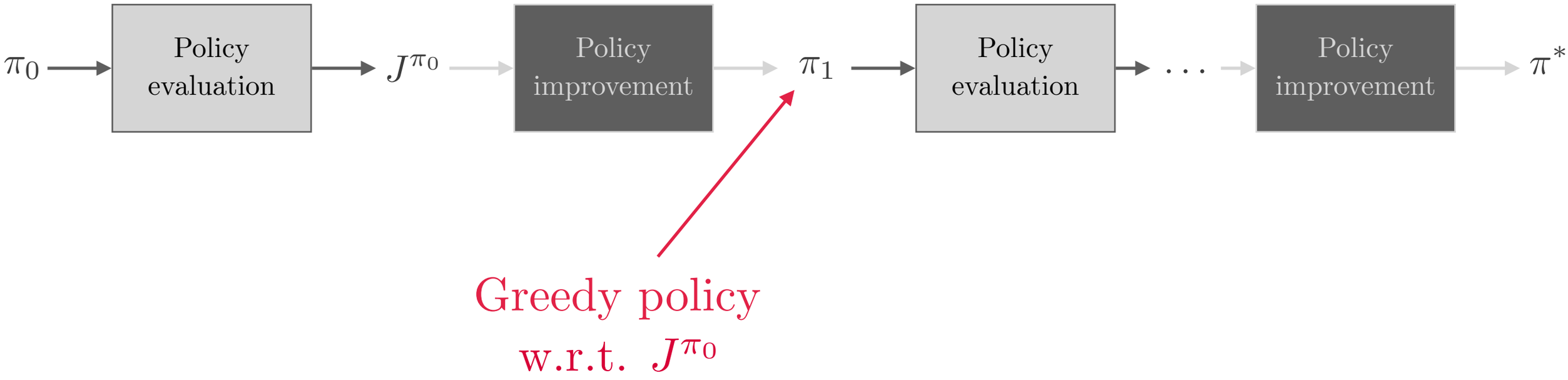
Improvement result

- Given a policy π...
- ... given the cost-to-go function J^{π},
- ... then

$$J^{\pi_g}(x) \leq J^{\pi}(x)$$

Policy π_g is better than π

Policy iteration



Policy iteration

- We get an iterative approach:

Input: MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \{\mathbf{P}_a\}, c, \gamma)$

1: Initialize $k = 0$

2: Initialize π_0 randomly

3: **repeat**

4: $\quad \mathbf{J} = (\mathbf{I} - \gamma \mathbf{P}_{\pi_\mathbf{k}})^{-1} \mathbf{c}_{\pi_k}$

5: $\quad \pi_{k+1} = \pi_g^J$

6: $\quad k = k + 1$

6: **until** $\pi_k = \pi_{k-1}$

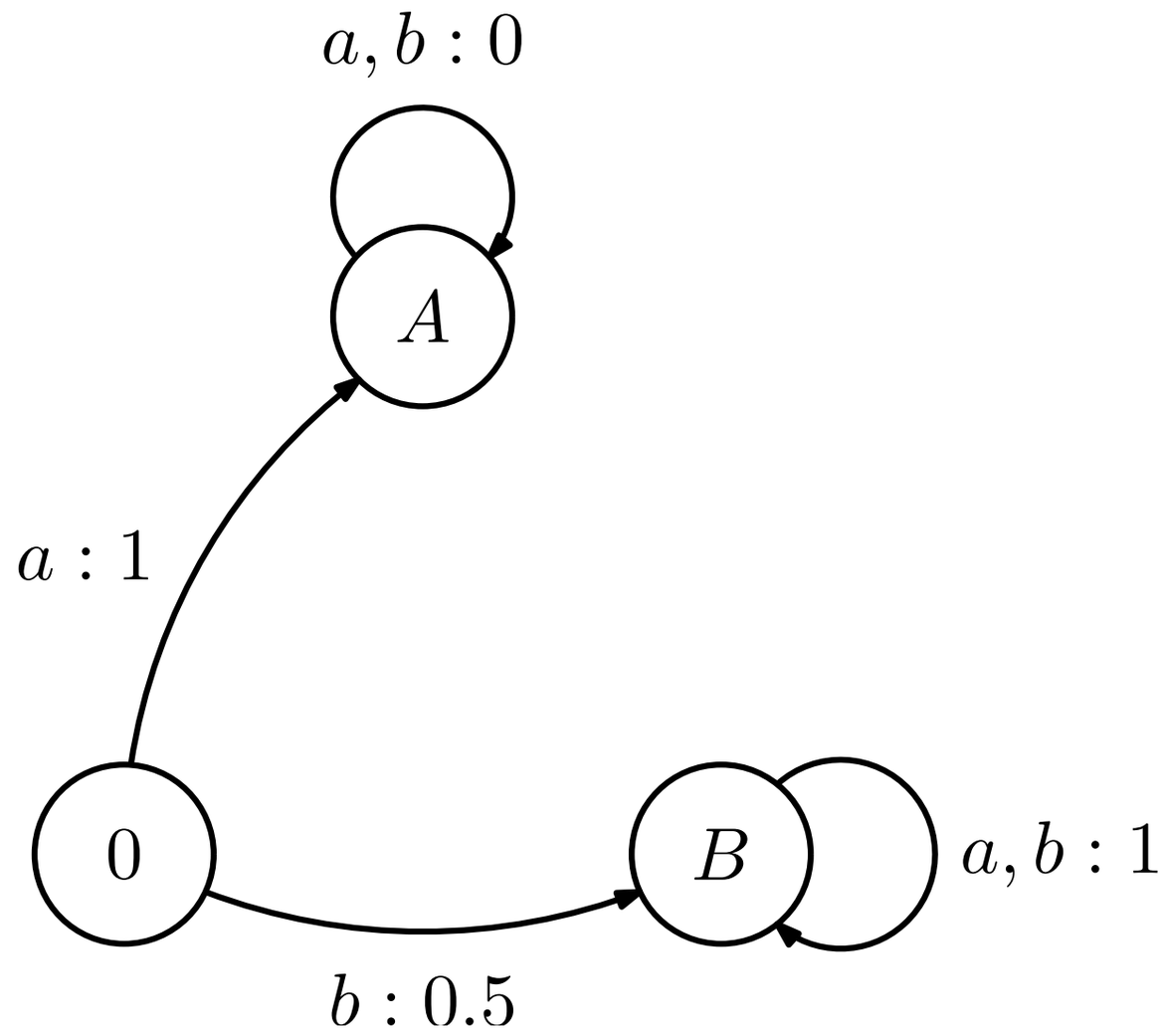
return π_k

- This algorithm is called **policy iteration**

Example

- Let us compute the optimal policy using PI
- Start with the uniform policy

$$\pi_0 = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$



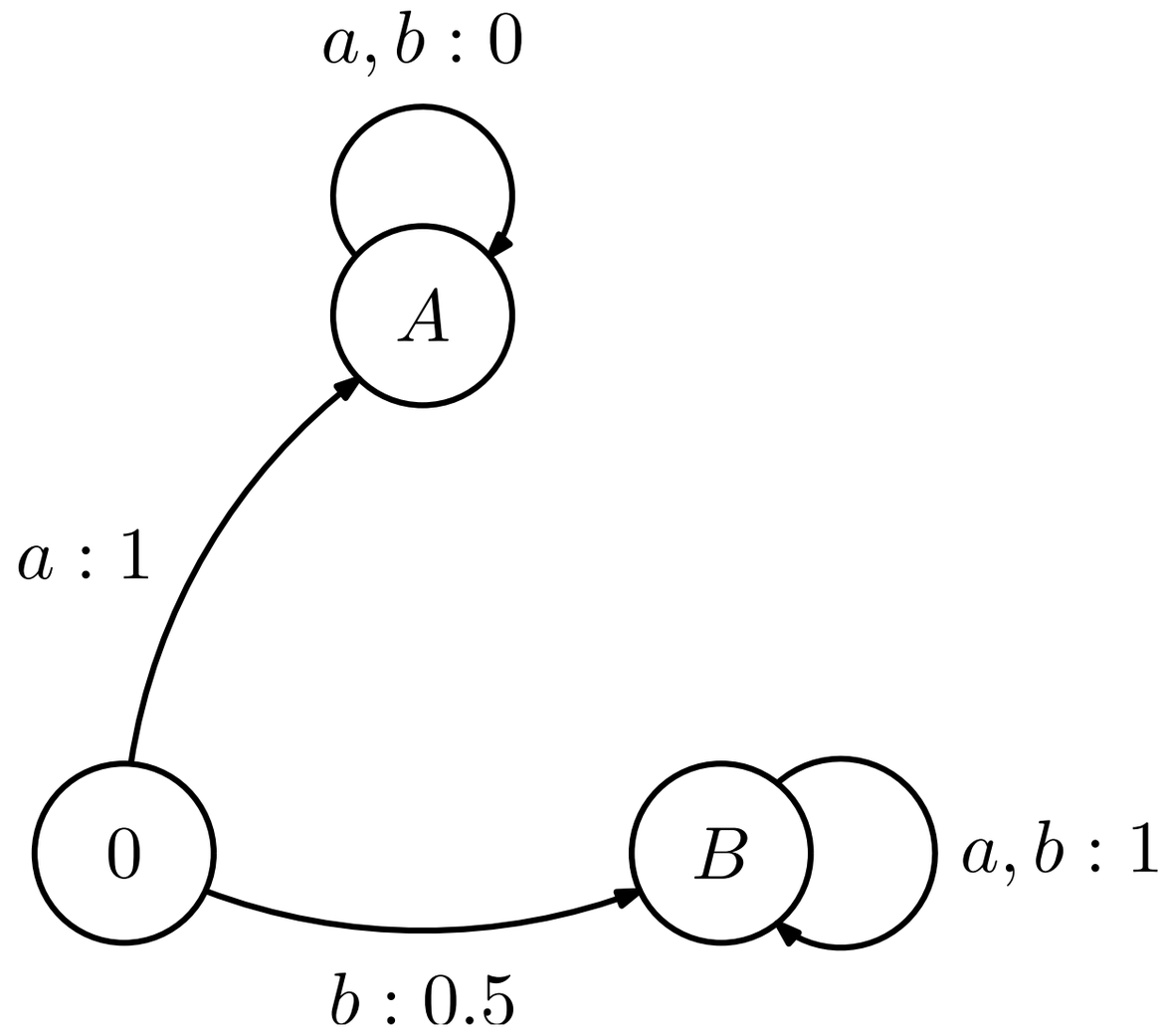
Example

- Step 1: Evaluate π

$$\mathbf{c}_\pi = \begin{bmatrix} 0.75 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathbf{P}_\pi = \begin{bmatrix} 0 & 0.5 & 0.5 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{J}^\pi = \begin{bmatrix} 50.25 \\ 0 \\ 100 \end{bmatrix}$$

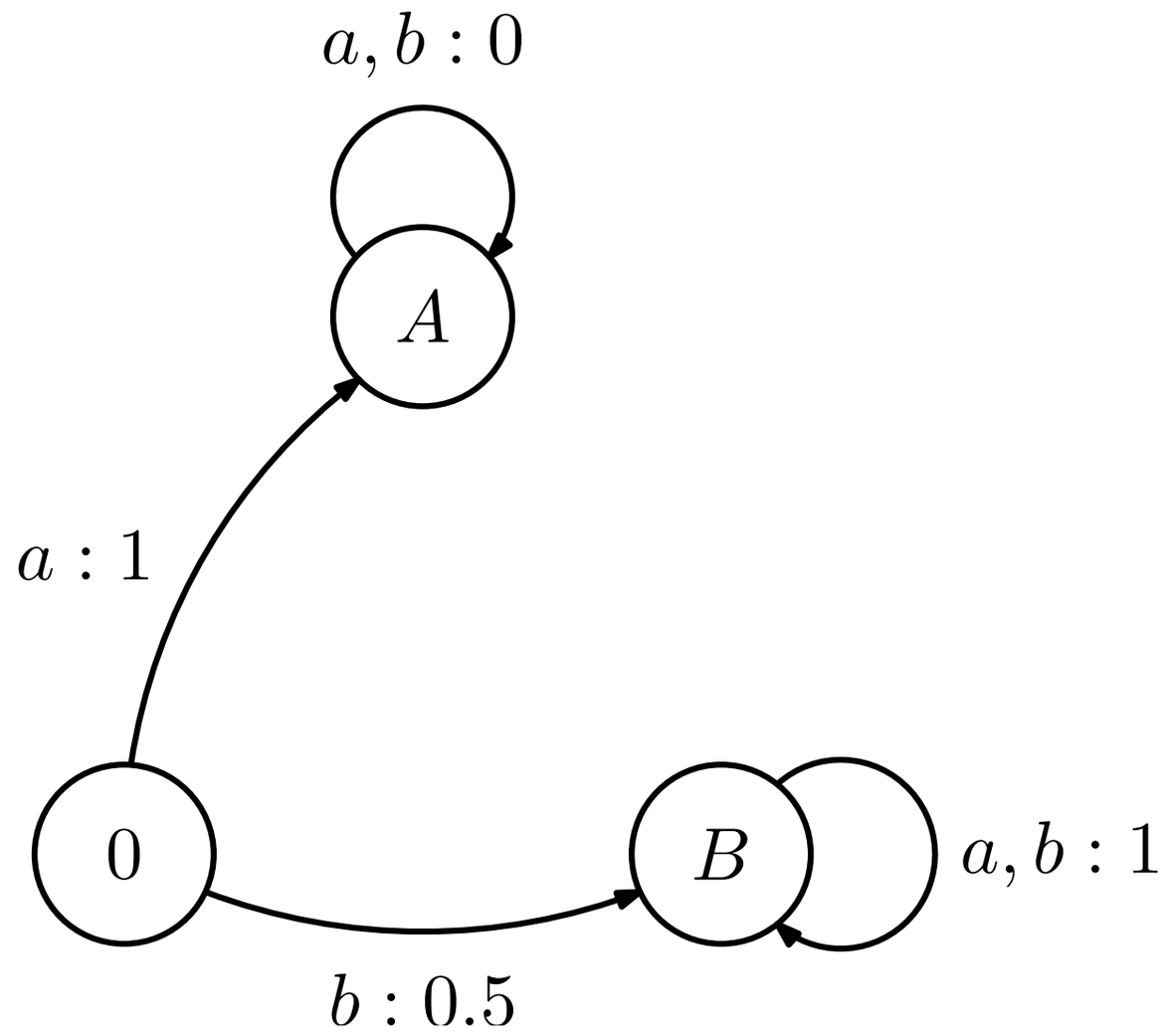


Example

- Step 2: Improve π

$$\mathbf{Q}^{\pi} = \begin{bmatrix} 1 & 99.5 \\ 0 & 0 \\ 100 & 100 \end{bmatrix}$$

$$\pi_1 = \begin{bmatrix} 1 & 0 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$



... and we're done!

Example

```python
def policy_iteration(M):

    # These variables are just to help readibility
    X = M[0]
    A = M[1]
    P = M[2]
    c = M[3]
    gamma = M[4]

    # Initialize pi with the uniform policy
    pol = np.ones((len(X), len(A))) / len(A)
    quit = False
    niter = 0

    while not quit:
        # Auxiliary array to store intermediate values
        Q = np.zeros((len(X), len(A)))

        # Policy evaluation
        cpi = np.sum(c * pol, axis=1, keepdims=True)
        Ppi = pol[:, 0, None] * P[0]

        for a in range(1, len(A)):
            Ppi += pol[:, a, None] * P[a]

        J = np.linalg.inv(np.eye(len(X)) - gamma * Ppi).dot(cpi)

        # Compute Q-values
        for a in range(len(A)):
            Q[:, a, None] = c[:, a, None] + gamma * P[a].dot(J)

        # Compute greedy policy
        Qmin = np.min(Q, axis=1, keepdims=True)

        pnew = np.isclose(Q, Qmin, atol=1e-8, rtol=1e-8).astype(int)
        pnew = pnew / pnew.sum(axis = 1, keepdims = True)

        # Compute stopping condition
        quit = (pol == pnew).all()

        # Update
        pol = pnew
        niter += 1

    print(f'Done after {niter} iterations.')
    return np.round(pol, 3)

pol = policy_iteration(M)
print(pol)
```

```
Done after 2 iterations.
[[1.  0. ]
 [0.5 0.5]
 [0.5 0.5]]
```

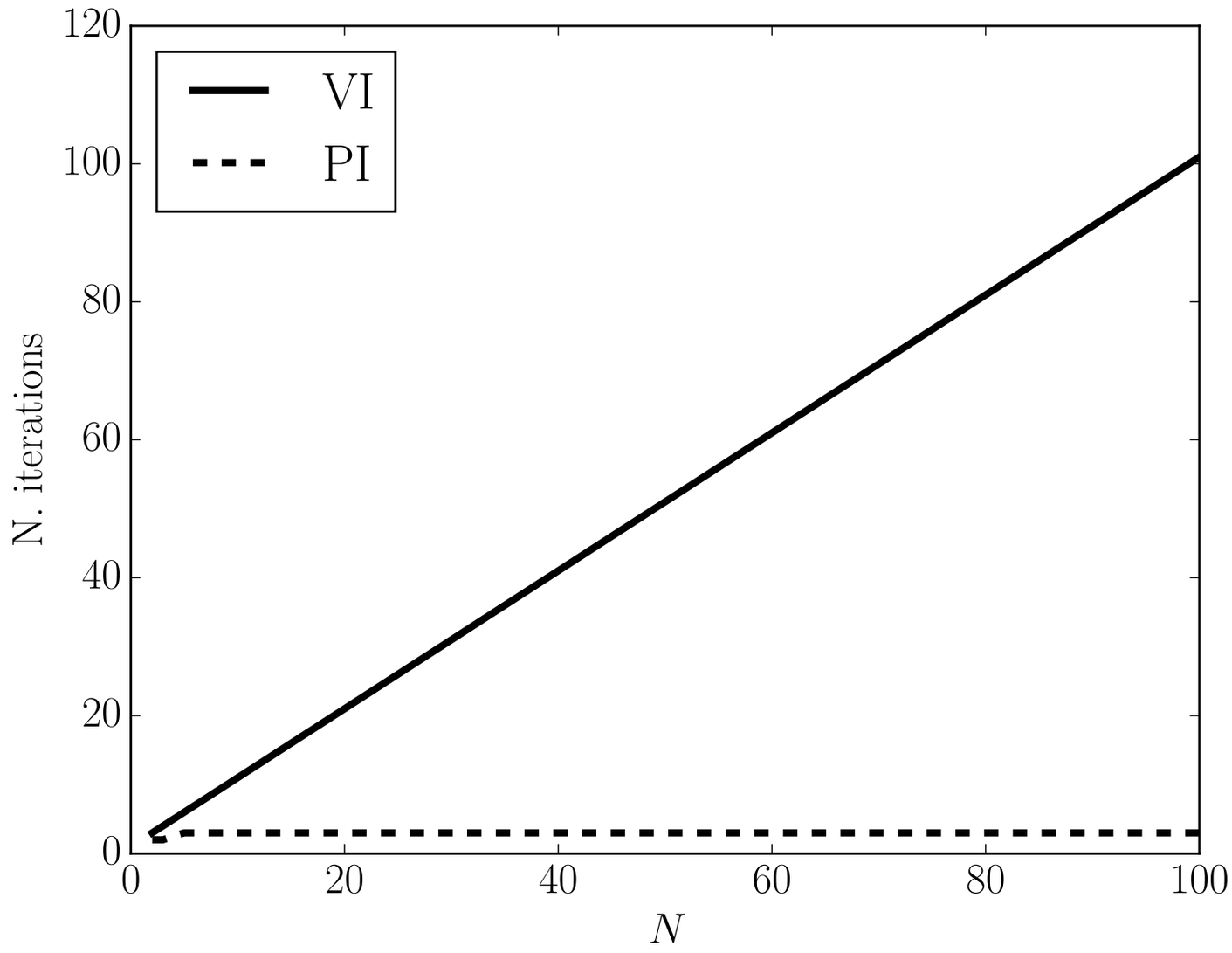
Value iteration took 1834 iterations...

Which should we use?

VI or PI?

Computational efficiency

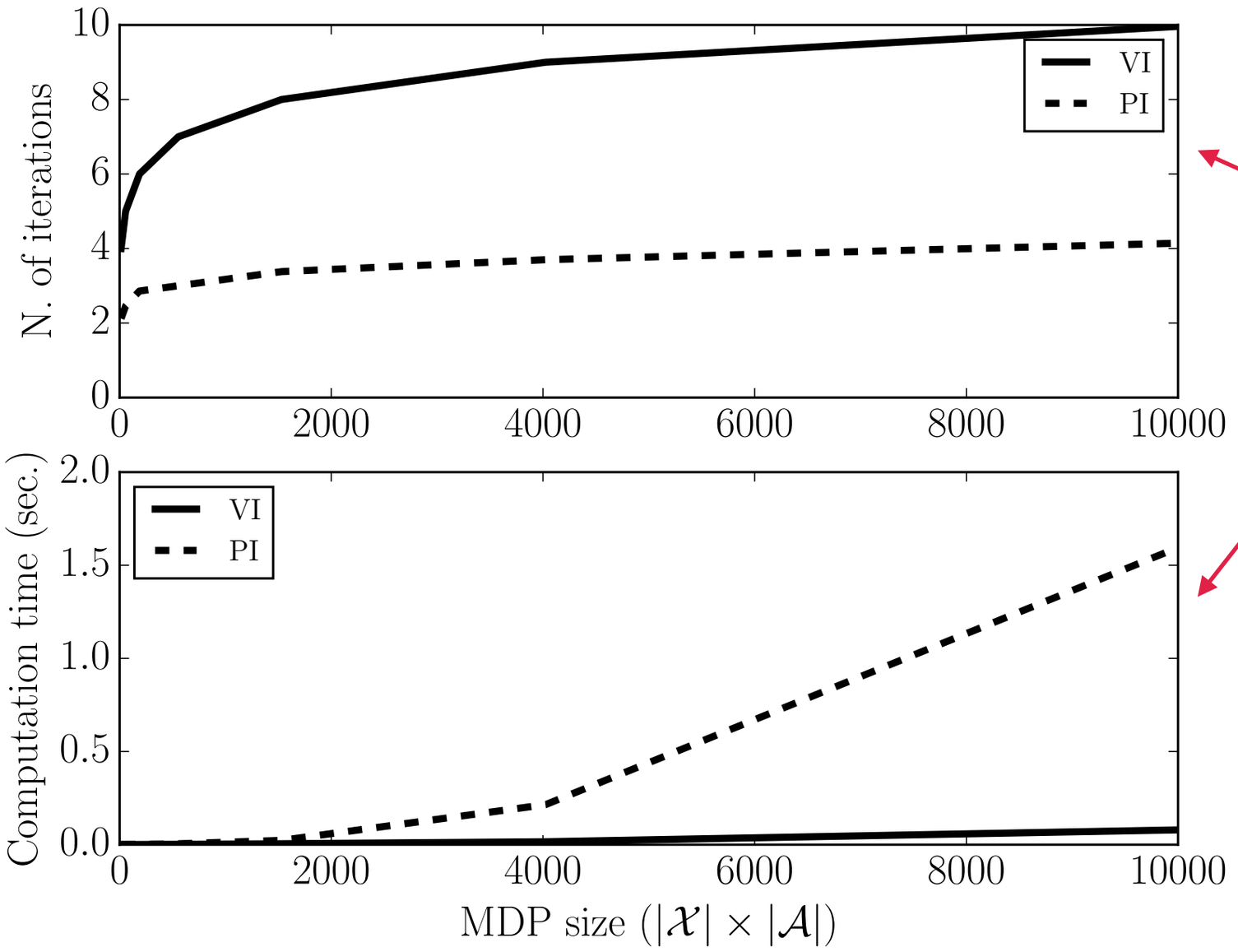
- Results in an MDP with N states:



Careful with this plot!

Computational efficiency

- Another example...



Although VI takes more iterations, it takes less time