TÉCNICO LISBOA

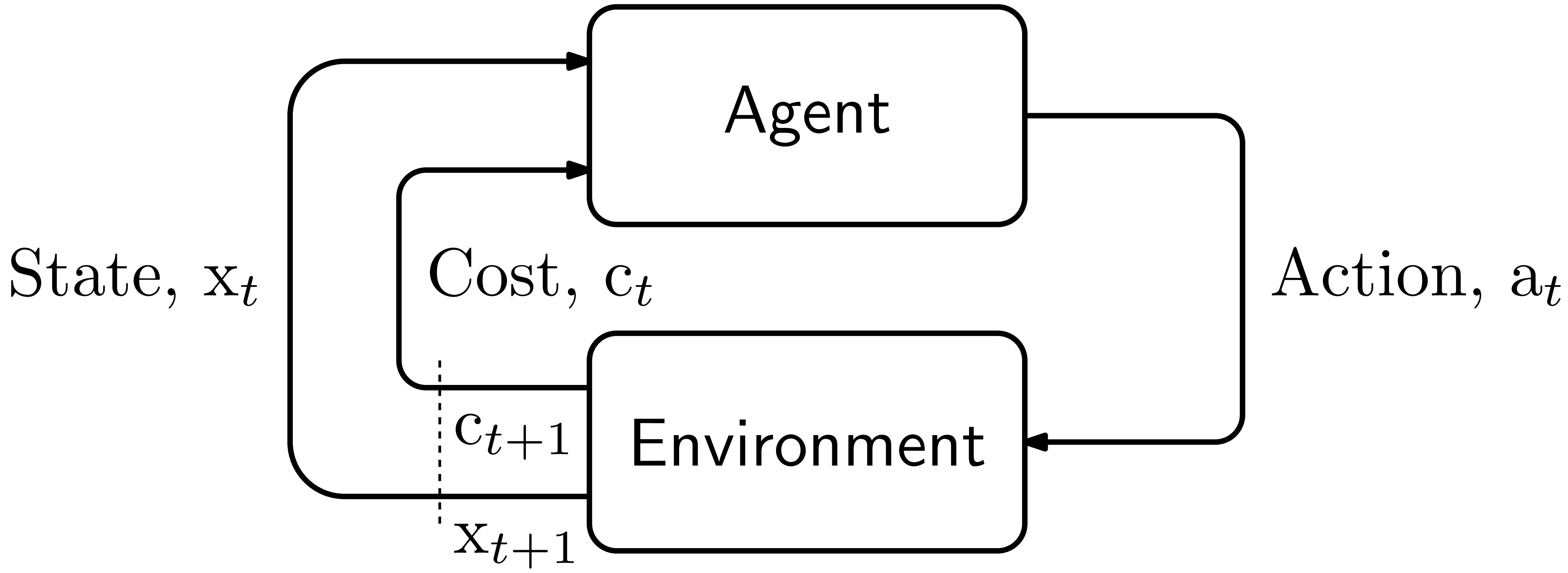
Planning, Learning and Decision Making

Lecture 7. Markov decision problems (cont.)

Markov decision process

- **Model** for sequential decision processes
- Described by:
  - State space, $\mathscr{X}$
  - Action space, $\mathscr{A}$
  - Transition probabilities, $\{\mathbf{P}_a, a \in \mathscr{A}\}$
  - Immediate cost function, $\mathbf{c}$

Markov decision process



Discounted cost-to-go

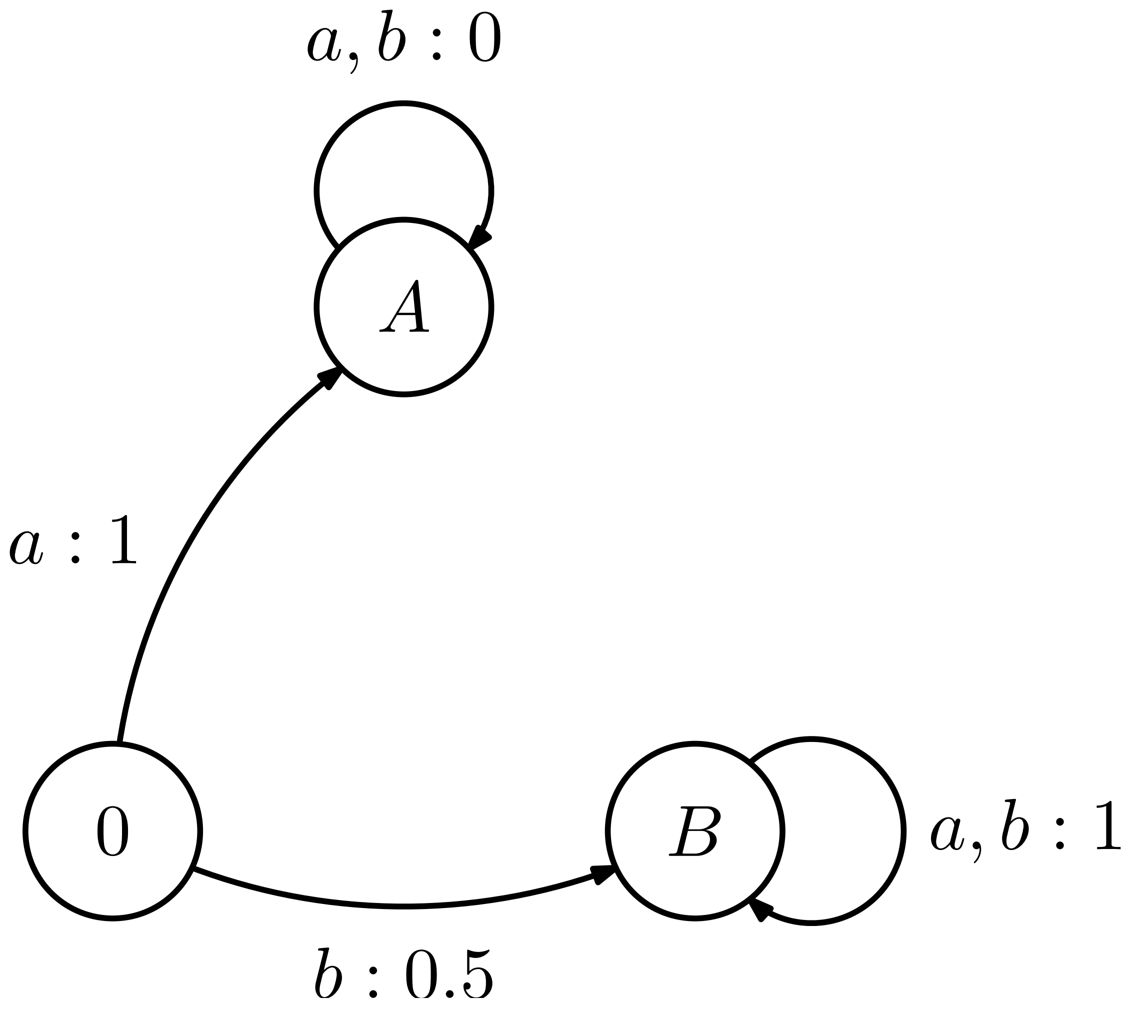
- Discounted cost-to-go:

$$DC \stackrel{\text{def}}{=} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{c}_t\right]$$

Example

- What is the discounted cost-to-go if we always select *b*?

$$\boldsymbol{J} = \begin{bmatrix} \frac{1}{2} \cdot \frac{1+\gamma}{1-\gamma} \\ 0 \\ \frac{1}{1-\gamma} \end{bmatrix}$$



Policies

(or “ways of selecting actions”)

We 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:*

$$h_t = \{x_0, a_0, x_1, a_1, \ldots, x_{t-1}, a_{t-1}, x_t\}$$

- Set of *t*-length histories (histories up to time *t*) is denoted as $\mathscr{H}_t$

Policies

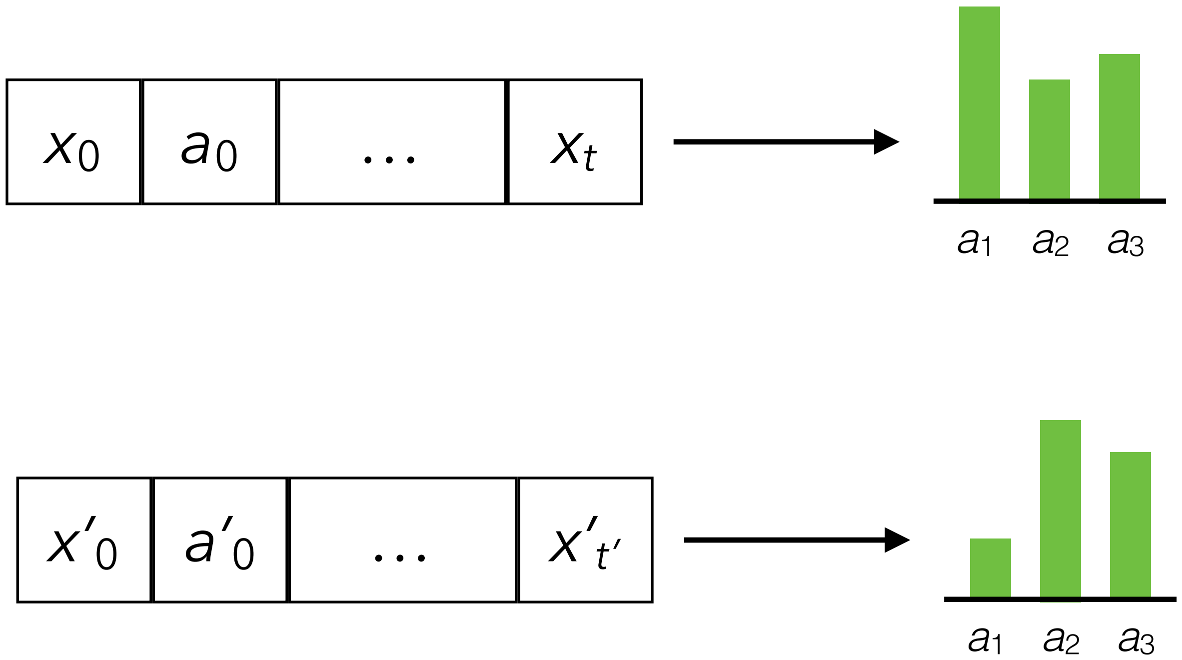
Distributions over actions

- A **policy** is a mapping $\pi : \mathscr{H} \to \Delta(\mathscr{A})$

- $\pi(a \mid h)$ is a probability of selecting action a after observing history h

History h

History h'

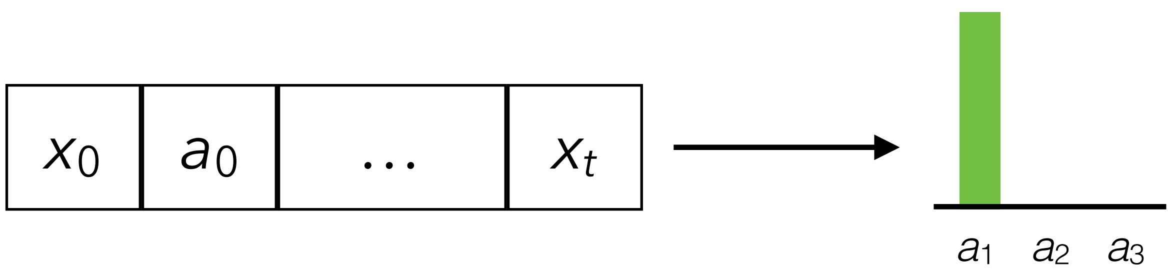


Distribution $\pi(\cdot \mid h)$

Distribution $\pi(\cdot \mid h')$

Types of policies

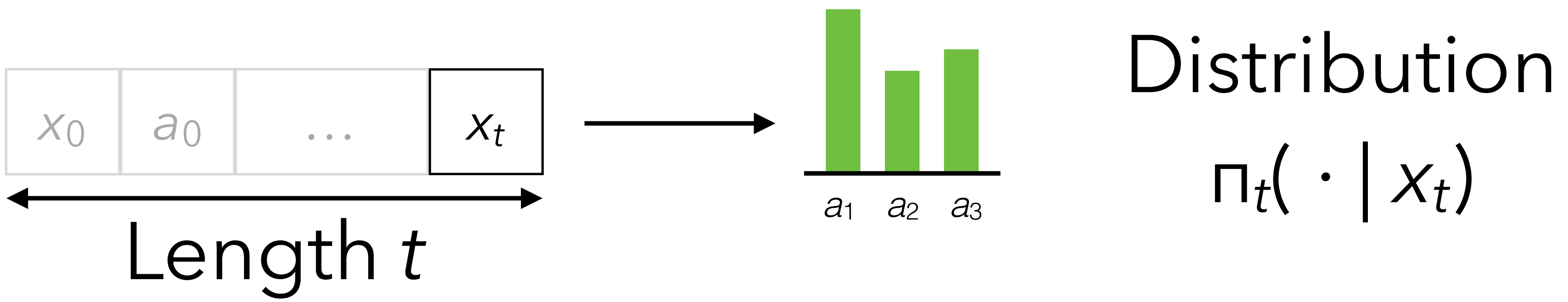
- **Deterministic**
  - … if there is one action that is selected with probability 1
  - We write $\pi(h)$ to denote such action



- **Stochastic**
  - … if it is not deterministic

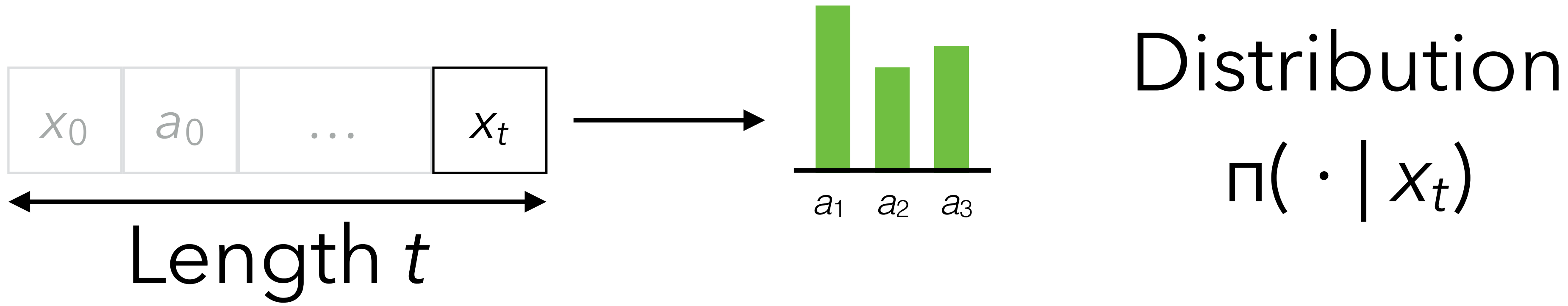
Types of policies

- **Markov**
  - … if the probability $\pi(a \mid h)$ depends only on the length of h and on its last state
  - If $h = \{x_0, a_0, \ldots, x_t\}$, we write $\pi_t(a \mid x_t)$ to denote the probability $\pi(a \mid h)$



Types of policies

- **Stationary**
  - … if it is Markov and does not depend on the length of h
  - If $h = \{x_0, a_0, \ldots, x_t\}$, we write $\pi(a \mid x_t)$ to denote the probability $\pi(a \mid h)$



Markov cost process

- What happens if the agent selects the actions…
  - … using only **current information** (does not depend on the past)
  - … always in the same way across time

Stationary
policy

TÉCNICO LISBOA

Markov cost process

- If an agent follows a fixed stationary policy π, a Markov decision process becomes a **Markov cost process**
- The state process $\{x_t\}$ is a **Markov chain** with transition probabilities

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

Markov cost process

- If an agent follows a fixed stationary policy π, a Markov decision process becomes a **Markov cost process**
- At each step, the expected cost is

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

Optimality

Cost-to-go function

- Cost-to-go function:
  - Fix a policy, π
  - Start the agent in state x
  - Let the agent go
  - Keep track of all costs to pay

Cost-to-go function

- How much will the agent pay?
  - Depends on the policy π
  - Depends on the initial state x

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

Discounted cost-to-go

Policy

Initial state

Cost-to-go function

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

Optimality

- A policy π^* is optimal if

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

  for all states

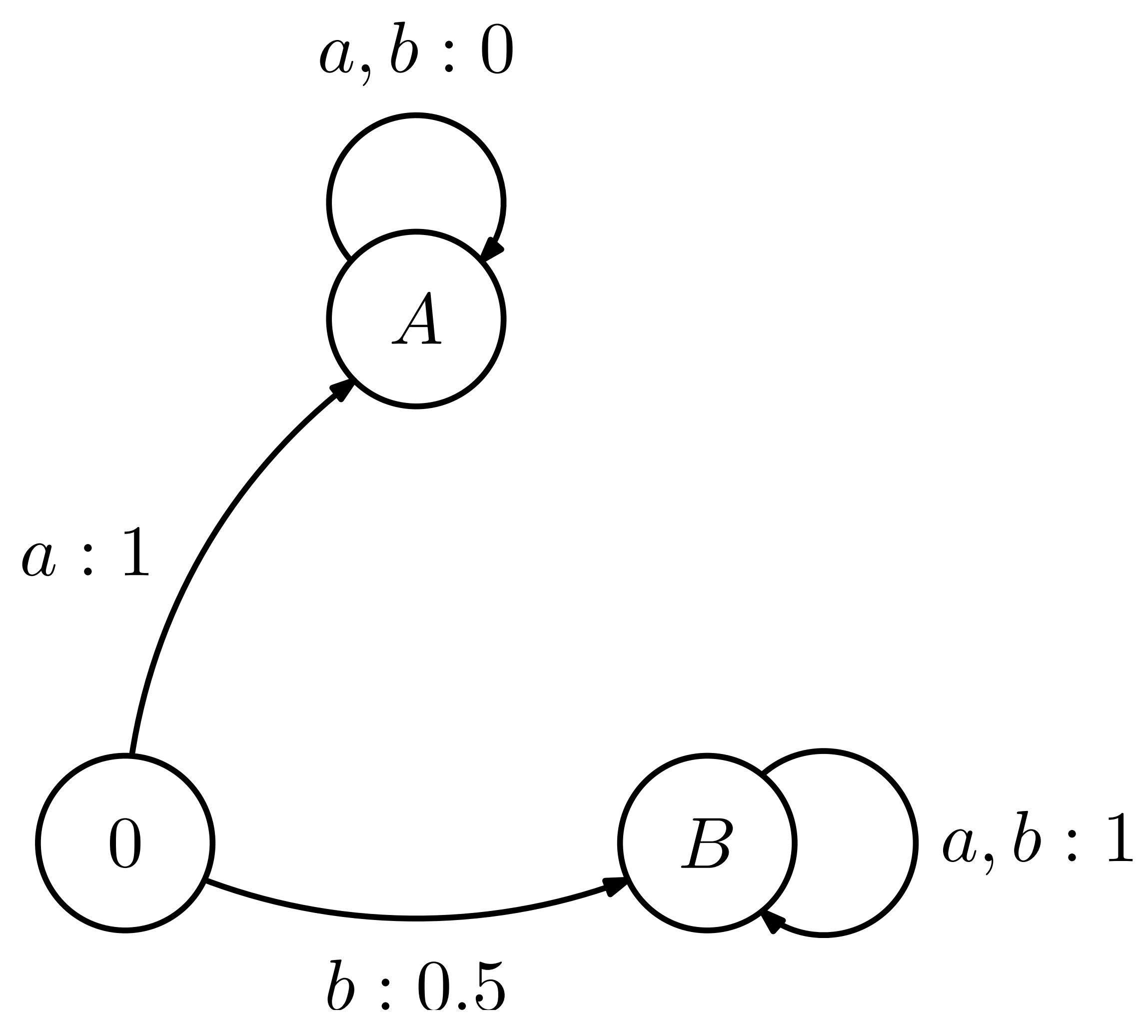
- In other words:

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

Does a policy like that even exist??

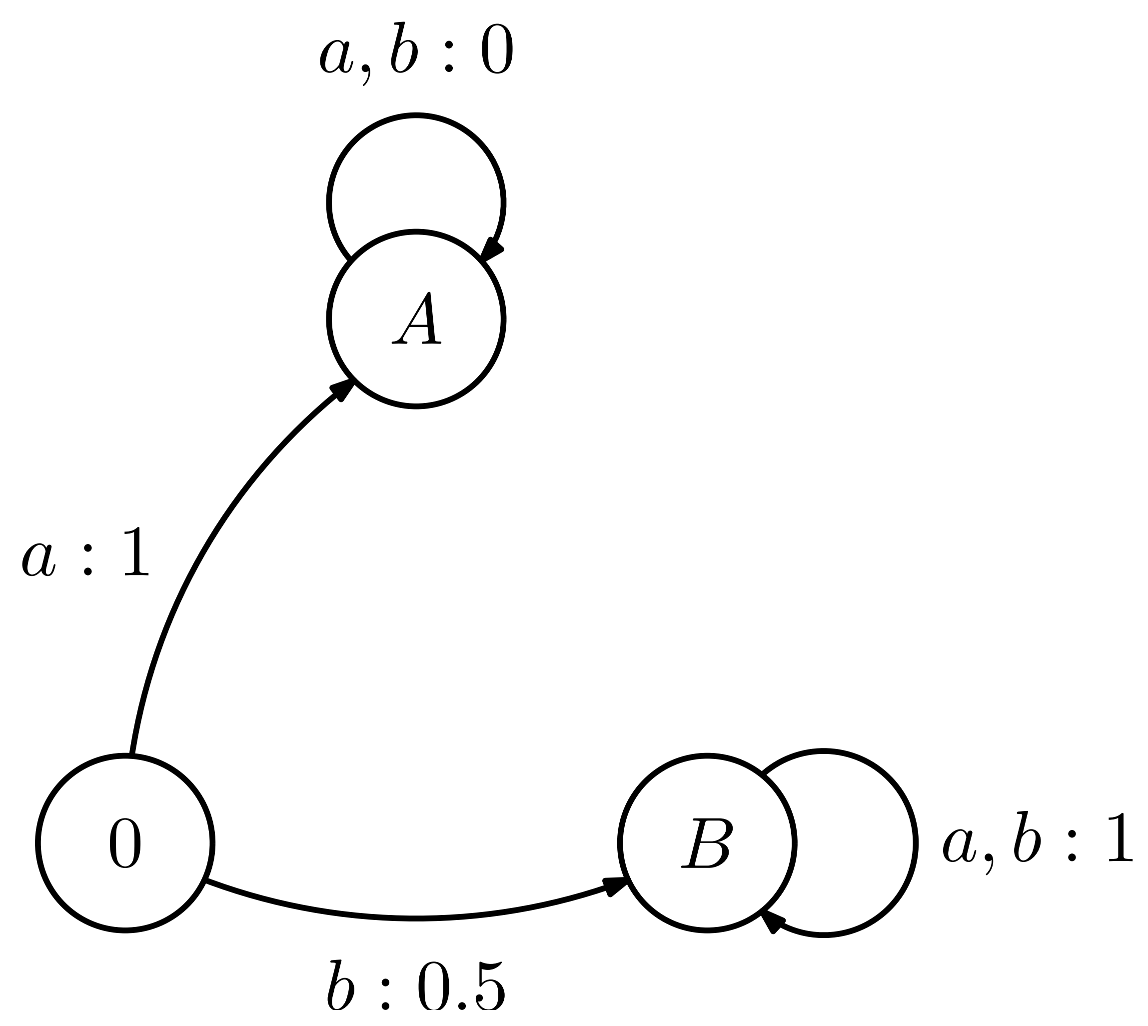
Example

- Consider the example:



Example

- We compare two policies:
  - A policy that always selects *a*
  - A policy that always selects *b*



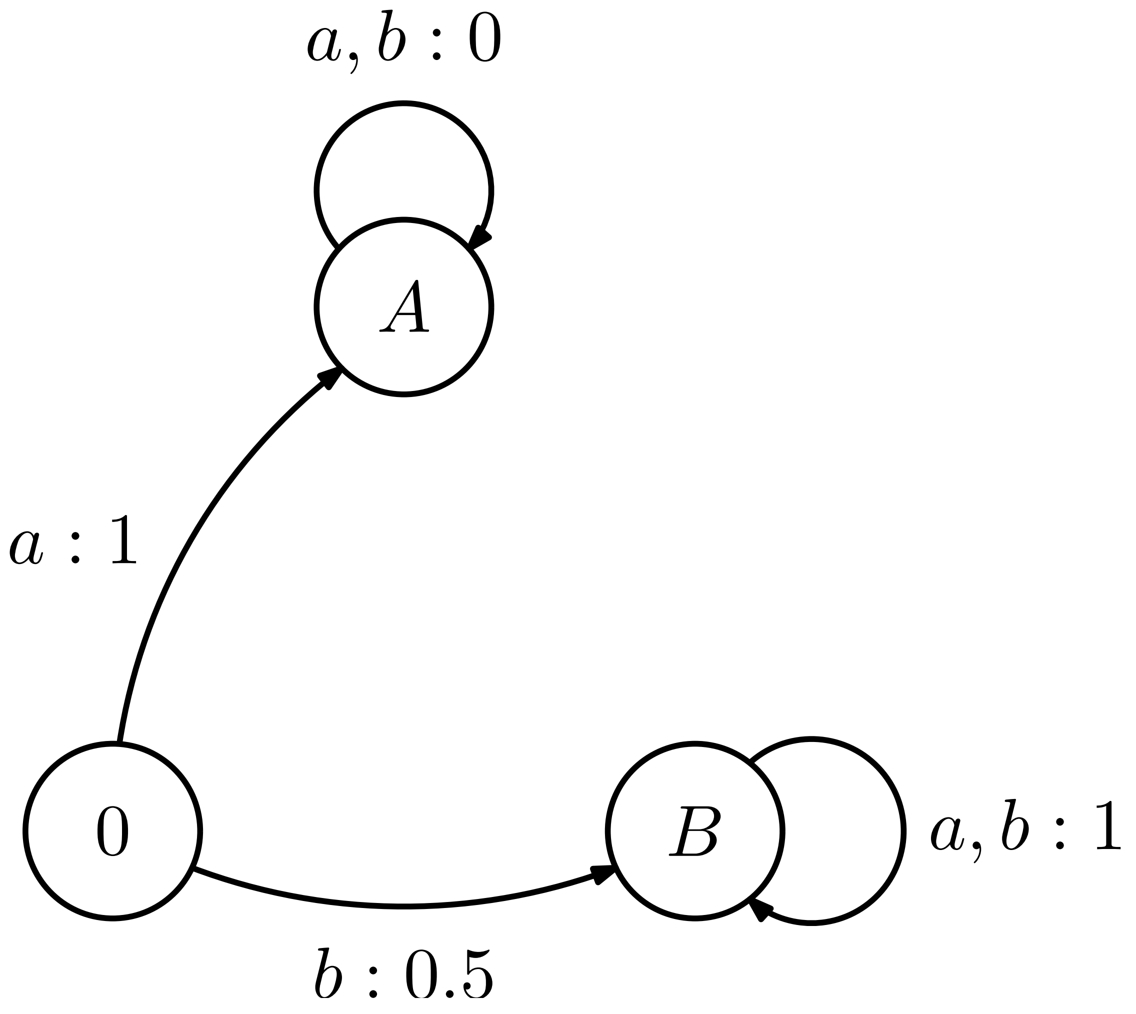
Example

- Policy that always selects *a*:

$$J^\pi(A) = 0 + \gamma 0 + \ldots = 0$$

$$J^\pi(B) = 1 + \gamma 1 + \ldots = \frac{1}{1-\gamma}$$

$$J^\pi(0) = 1 + \gamma 0 + \ldots = 1$$



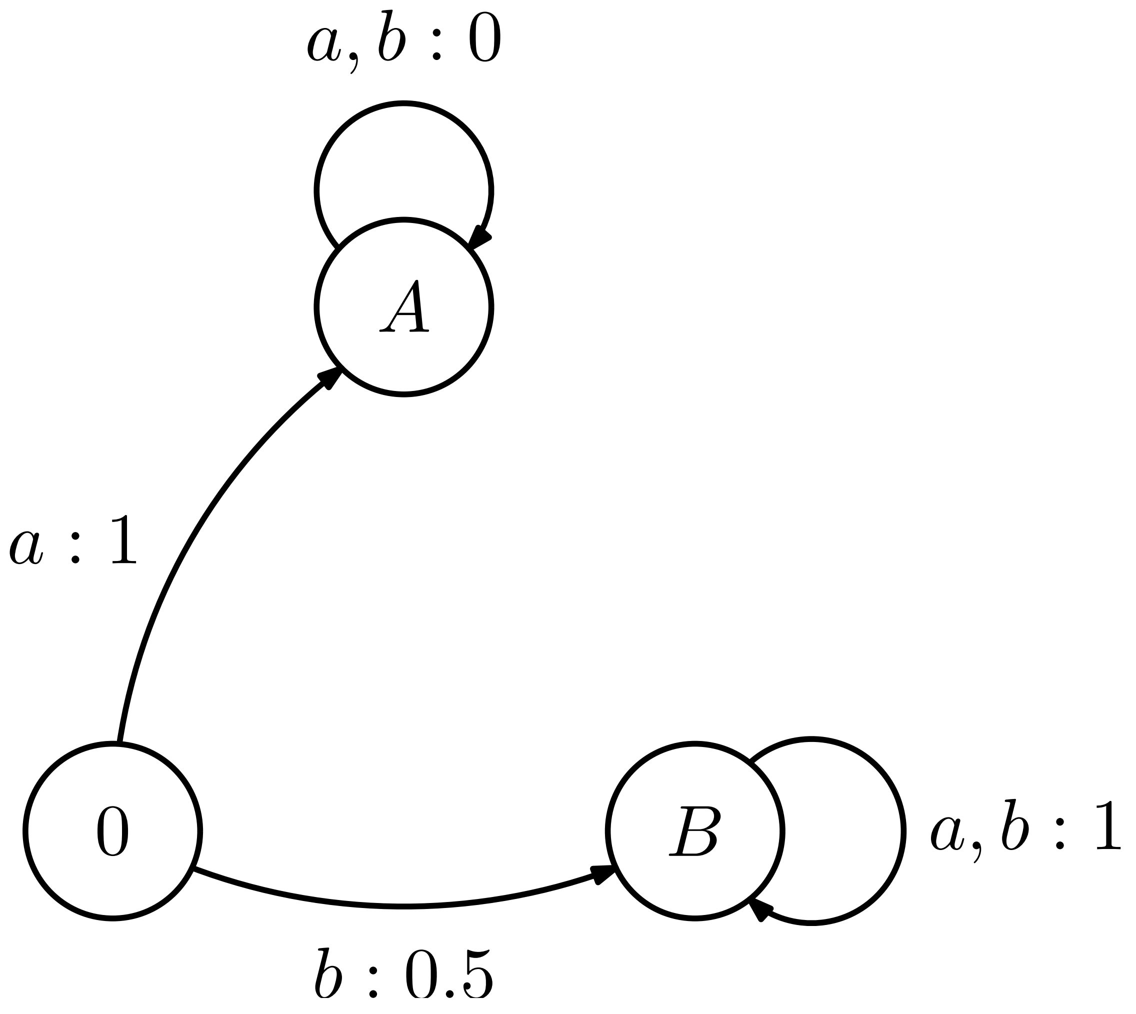
Example

- Policy that always selects *b*:

$$J^{\pi'}(A) = 0 + \gamma 0 + \ldots = 0$$

$$J^{\pi'}(B) = 1 + \gamma 1 + \ldots = \frac{1}{1-\gamma}$$

$$\begin{aligned} J^{\pi'}(0) &= 0.5 + \gamma 1 + \ldots \\ &= 0.5 + \gamma J(B) \\ &= \frac{1}{2} \cdot \frac{1+\gamma}{1-\gamma} \end{aligned}$$



Example

- Policy π is better if

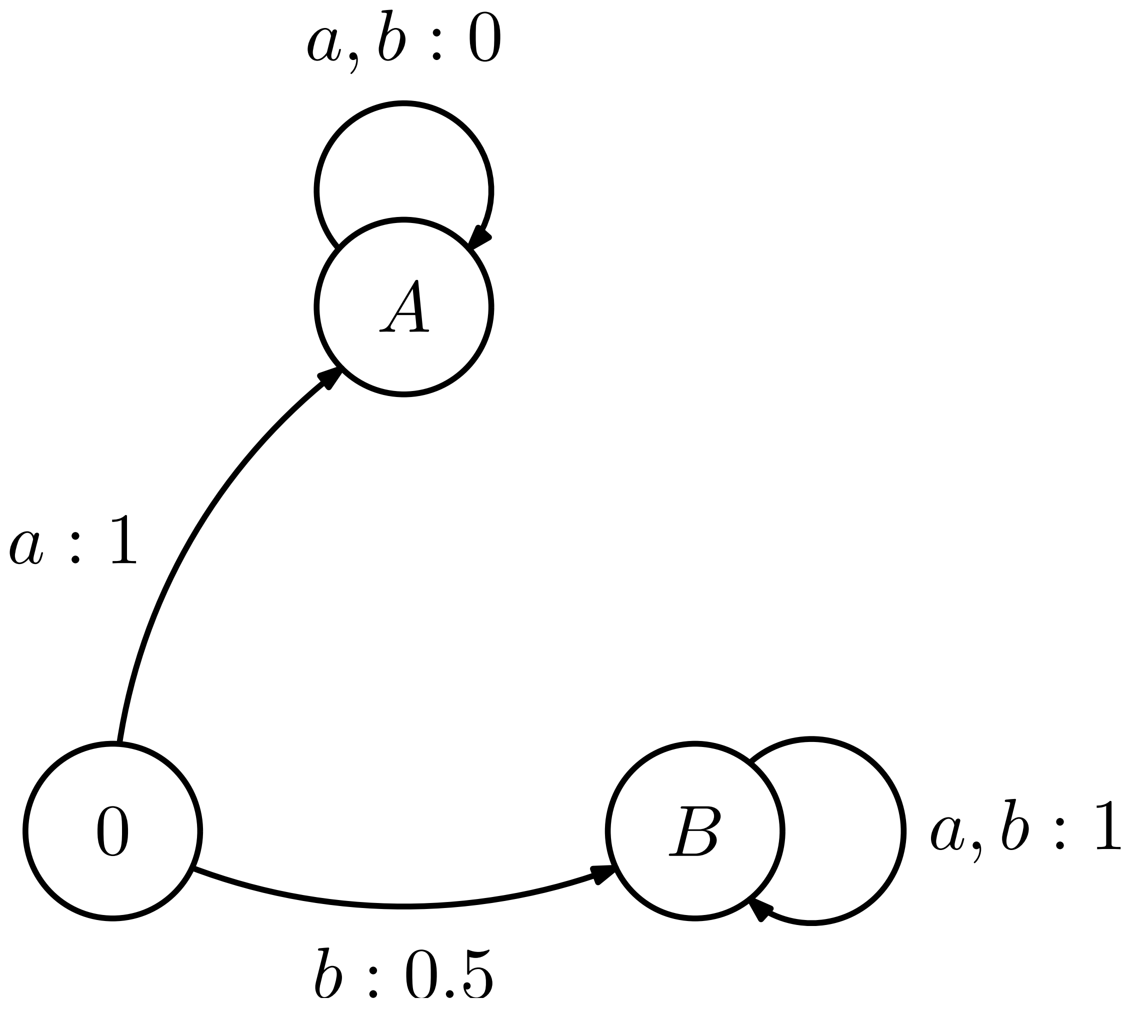
$$J^{\pi}(0) < J^{\pi'}(0)$$

- Equivalently, if

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

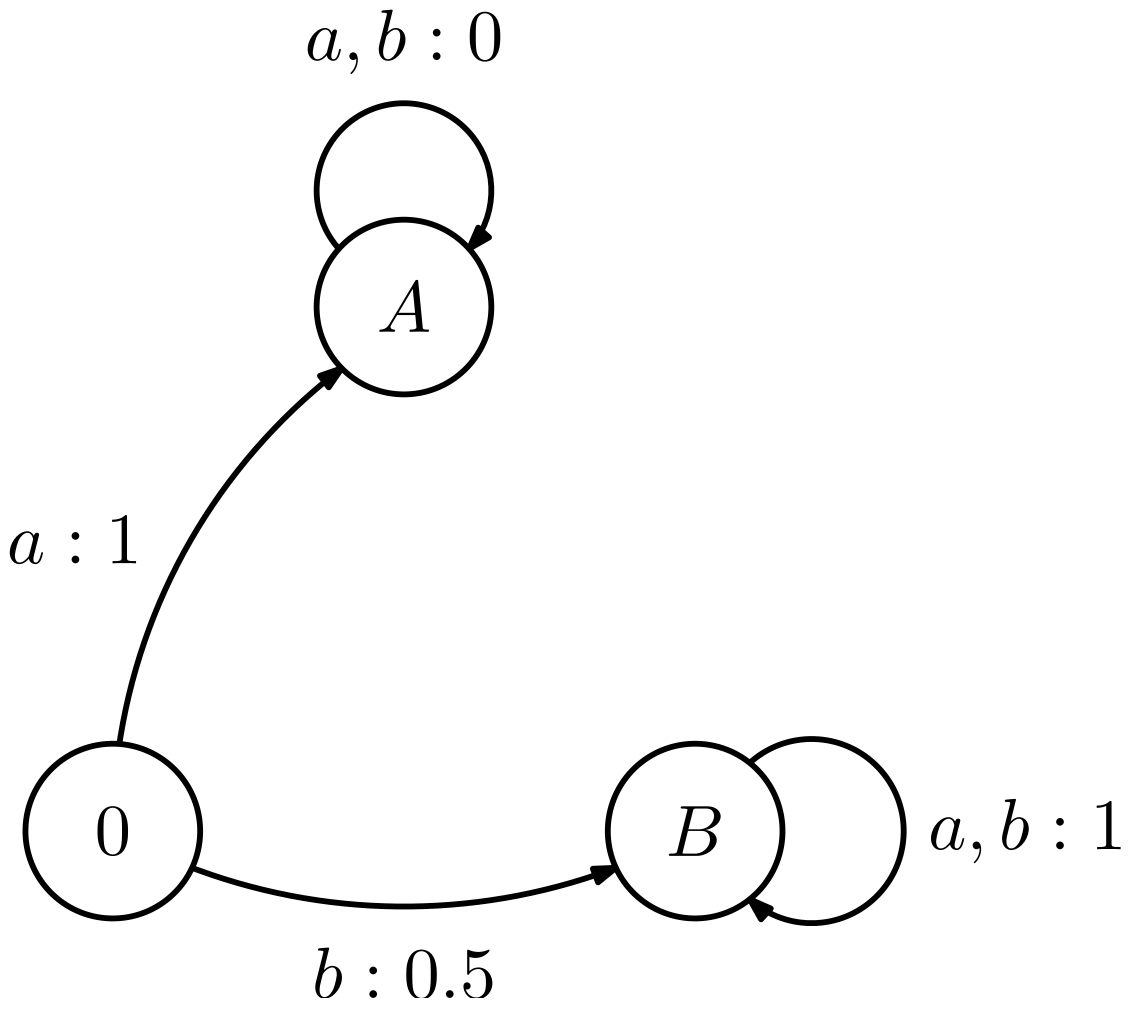
or

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



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??

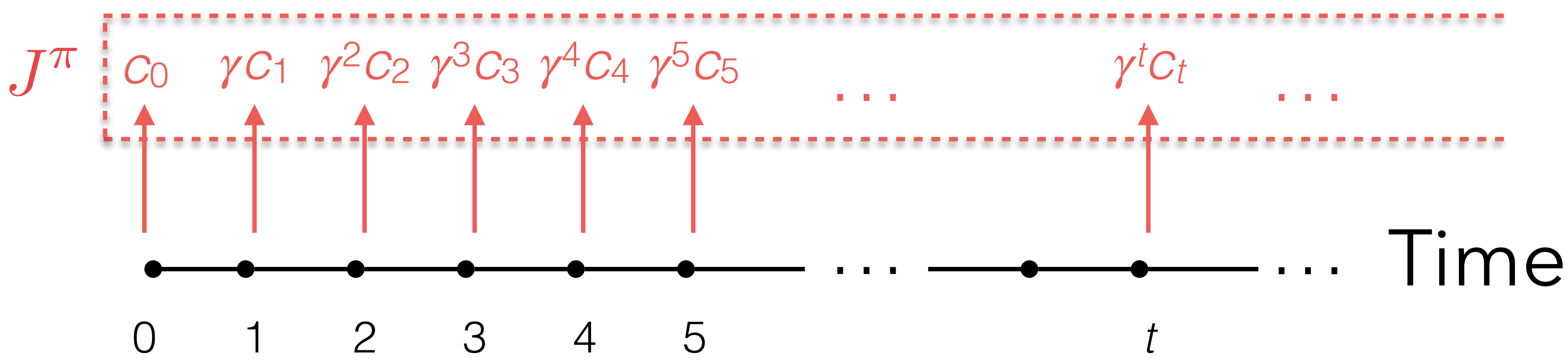
If so, how can we compute it?

A simpler problem

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

Prediction problem

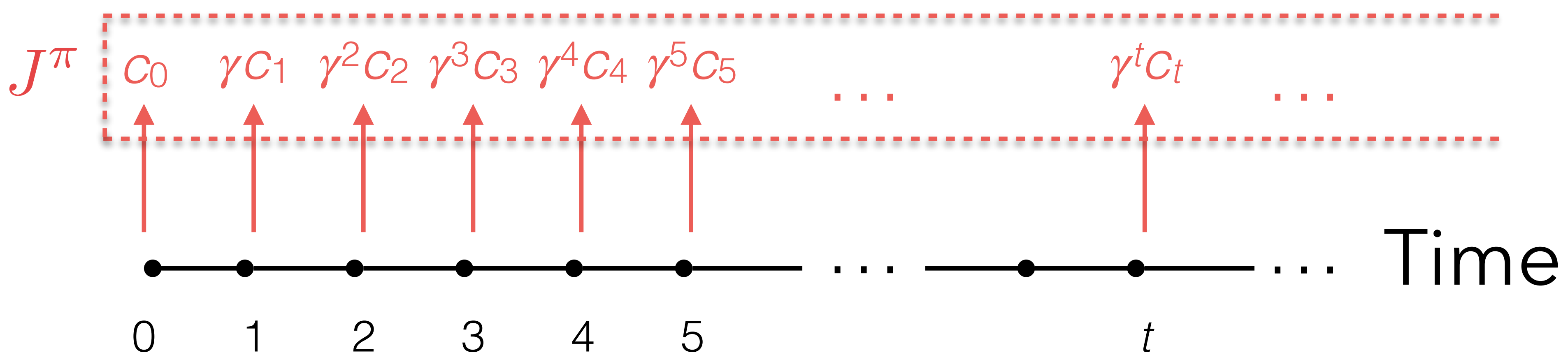
$$J^\pi(x) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_t \mid \mathrm{x}_0 = x\right]$$



A simpler problem

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

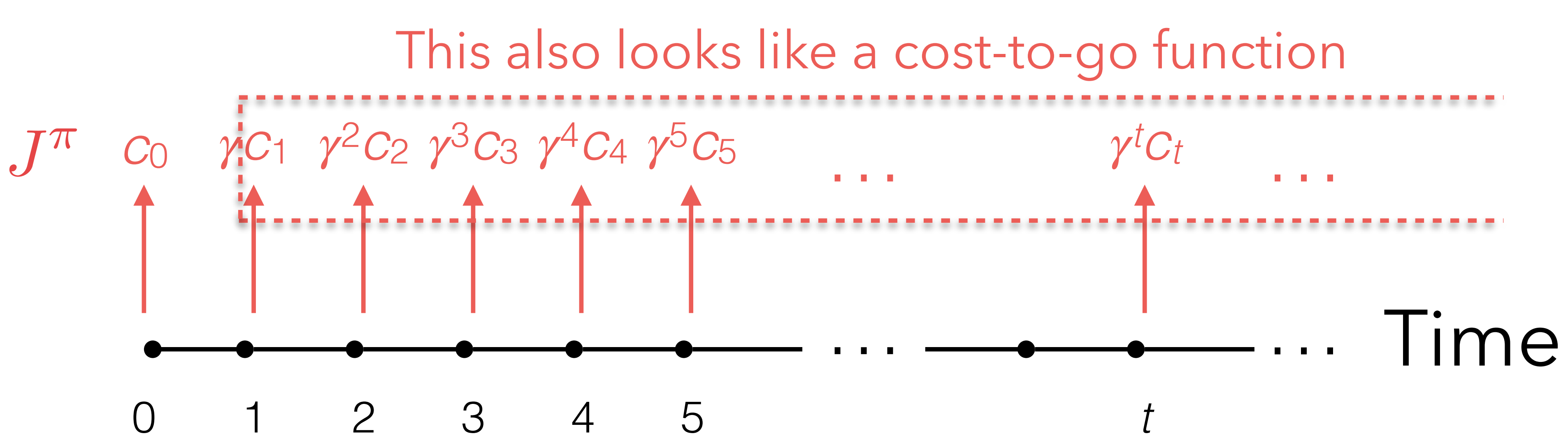
$$J^\pi(x) = \mathbb{E}_\pi\left[\mathrm{c}_0 + \gamma \mathrm{c}_1 + \gamma^2 \mathrm{c}_2 + \ldots \mid \mathrm{x}_0 = x\right]$$



A simpler problem

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

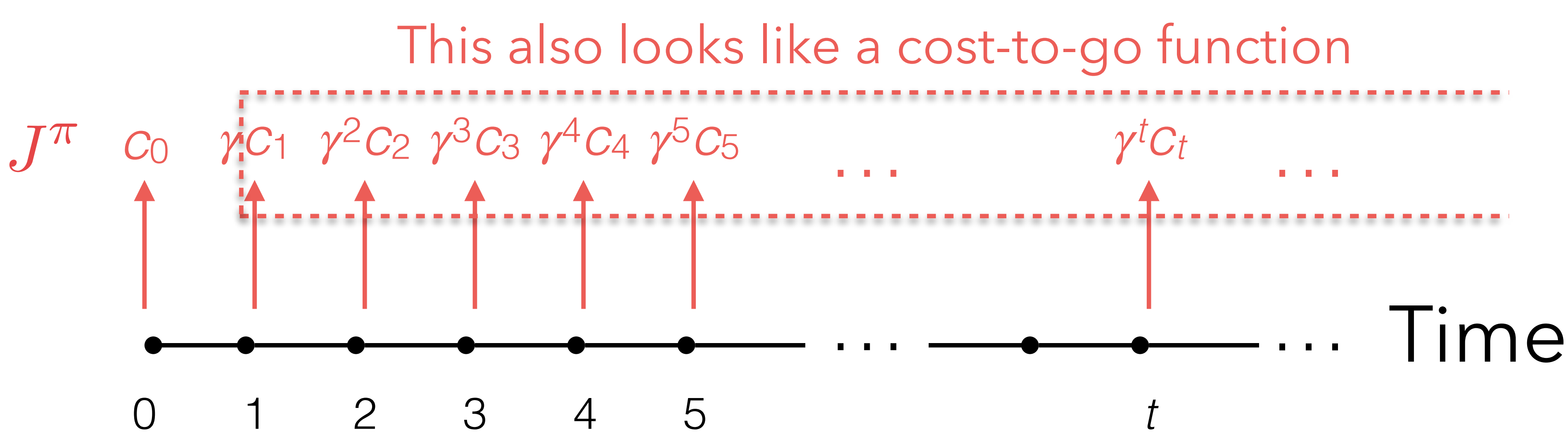
$$J^\pi(x) = \mathbb{E}_\pi\left[\mathrm{c}_0 + \gamma(\mathrm{c}_1 + \gamma \mathrm{c}_2 + \ldots) \mid \mathrm{x}_0 = x\right]$$



A simpler problem

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

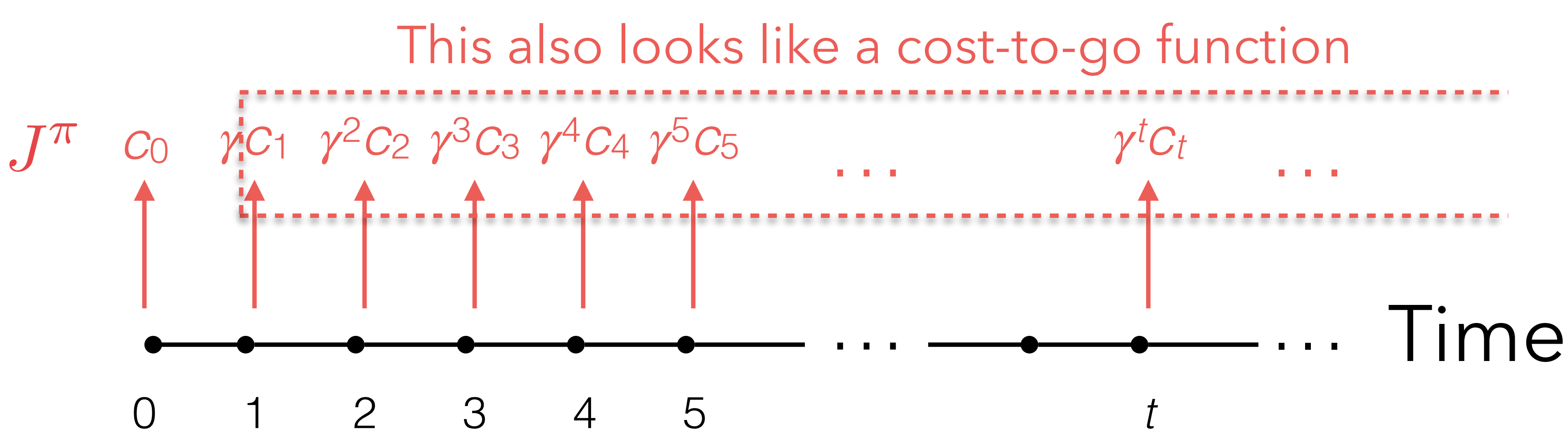
$$J^\pi(x) = \underbrace{\boxed{\mathbb{E}_\pi\left[\mathrm{c}_0 \mid \mathrm{x}_0 = x\right]}}_{c_\pi(x)} + \gamma\mathbb{E}_\pi\left[\mathrm{c}_1 + \gamma\mathrm{c}_2 + \ldots \mid \mathrm{x}_0 = x\right]$$



A simpler problem

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

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



A simpler problem

- Fix some stationary policy π
- What to do with the second term?

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

We would like to have x_1

A simpler problem

- Fix some stationary policy π
- What to do with the second term?

$$J^\pi(x) = c_\pi(x) + \gamma \sum_{y \in \mathcal{X}} \underbrace{\boxed{\mathbb{E}_\pi \left[\mathrm{c}_1 + \gamma \mathrm{c}_2 + \ldots \mid \mathrm{x}_1 = y \right]}}_{J^\pi(y)} \mathsf{P}_\pi(y \mid x)$$

We use the total probability law!

A simpler problem

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

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

- If we write these using vector notation:

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

Vector Vector Matrix Vector

Computing J$^{\pi}$

- We can easily solve the system

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

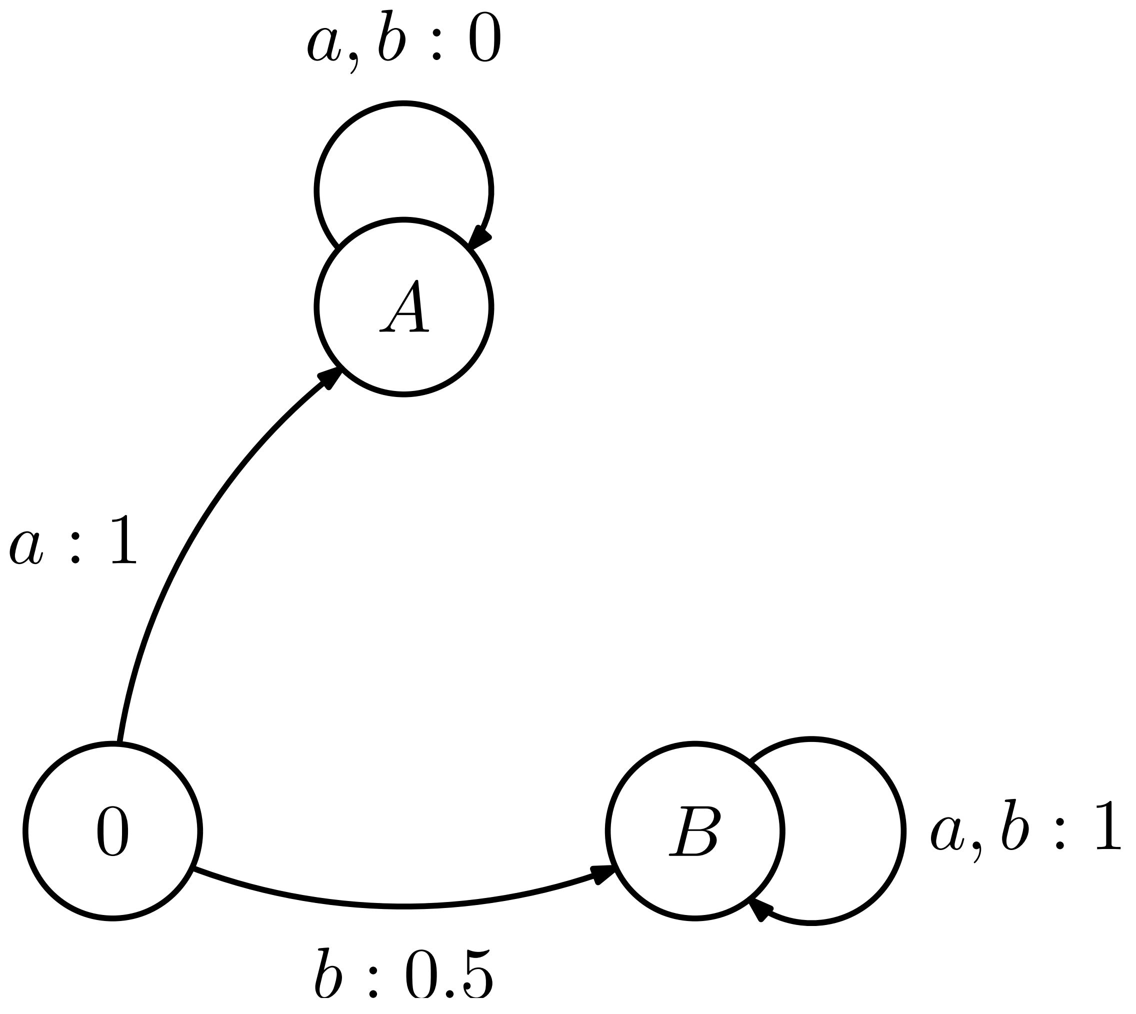
- We get

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

Example

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

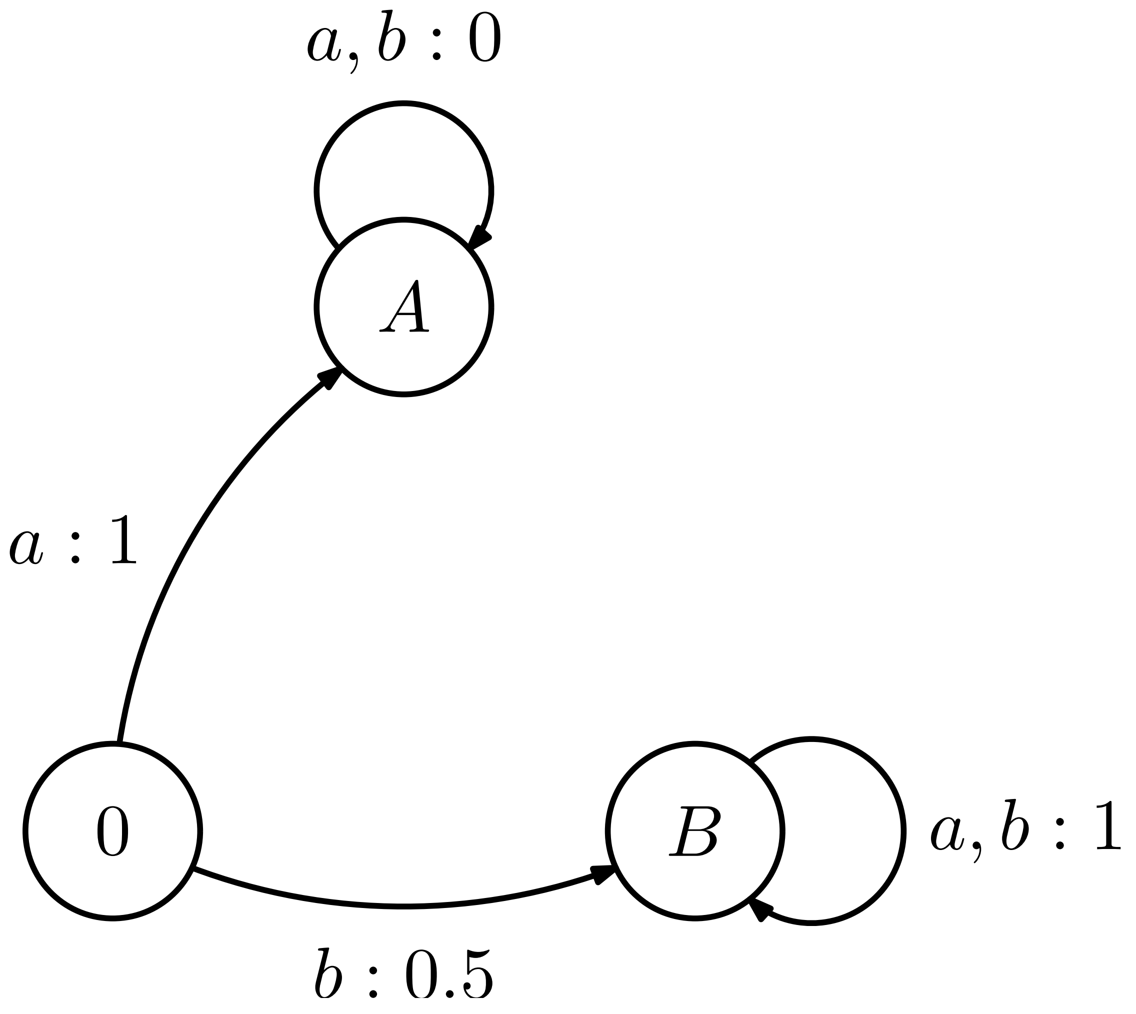
$$\boldsymbol{c}_\pi = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$



Example

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

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



Example

- We have

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

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

Example

- We have

$$
\begin{aligned}
\boldsymbol{J}^\pi &= (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \boldsymbol{c}_\pi \\
&= \begin{bmatrix} 1 & 99 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 \\ 0 \\ 100 \end{bmatrix}
\end{aligned}
$$

Values computed before!

Is this efficient?

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

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

$$\boldsymbol{J} \longrightarrow \boxed{\text{Operator } \mathbf{T}_\pi} \longrightarrow \boldsymbol{c}_\pi + \gamma \mathbf{P}_\pi \boldsymbol{J}$$

Is this efficient?

- We get an iterative approach:

$$\boldsymbol{J}_0 \to \boxed{\text{Operator } \mathbf{T}_\pi} \to \boldsymbol{J}_1 \to \boxed{\text{Operator } \mathbf{T}_\pi} \to \boldsymbol{J}_2 \to \boxed{\text{Operator } \mathbf{T}_\pi} \to \boldsymbol{J}_3 \to \ldots \to \boldsymbol{J}_\pi$$

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

Value Iteration, 1.0

Require: MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \{\boldsymbol{P}_a\}, c, \gamma)$; tolerance $\varepsilon > 0$; policy π;

1: Initialize $k = 0,\ J^{(0)} \equiv 0$
2: **repeat**
3: $\quad J^{(k+1)} \leftarrow \mathsf{T}_\pi J^{(k)}$
4: $\quad k \leftarrow k + 1$
5: **until** $\left\| J^{(k-1)} - J^{(k)} \right\|_2 < \varepsilon.$
6: **return** $J^{(k)}$

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

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

which is equivalent to

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

Another recursion

- … and replace π by min:

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

Another recursion

- … and replace π by min:

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

THE AMAZING RESULT

1. The recursion

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

has a single solution

THE AMAZING RESULT

2. The solution to the recursion

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

is J^*.

THE AMAZING RESULT

3. The action that minimizes the rhs of

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

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

$$\pi(x) = \operatorname*{argmin}_{a \in \mathcal{A}} \left[c(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathsf{P}_a(y \mid x) J(y) \right] \qquad (\dagger)$$

is optimal

Another value iteration

- We can use the recursive expression:

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

$$\boldsymbol{J} \longrightarrow \boxed{\text{Operator } \mathbf{T}} \longrightarrow \min\left[\boldsymbol{c}_a + \gamma \mathbf{P}_a \boldsymbol{J}\right]$$

TÉCNICO LISBOA

Another value iteration

- We get yet another iterative approach:

$$\boldsymbol{J}_0 \to \boxed{\text{Operator } \mathbf{T}} \to \boldsymbol{J}_1 \to \boxed{\text{Operator } \mathbf{T}} \to \boldsymbol{J}_2 \to \boxed{\text{Operator } \mathbf{T}} \to \boldsymbol{J}_3 \to \ldots \to \boldsymbol{J}^*$$

- This is also **value iteration**, this time to compute *J**

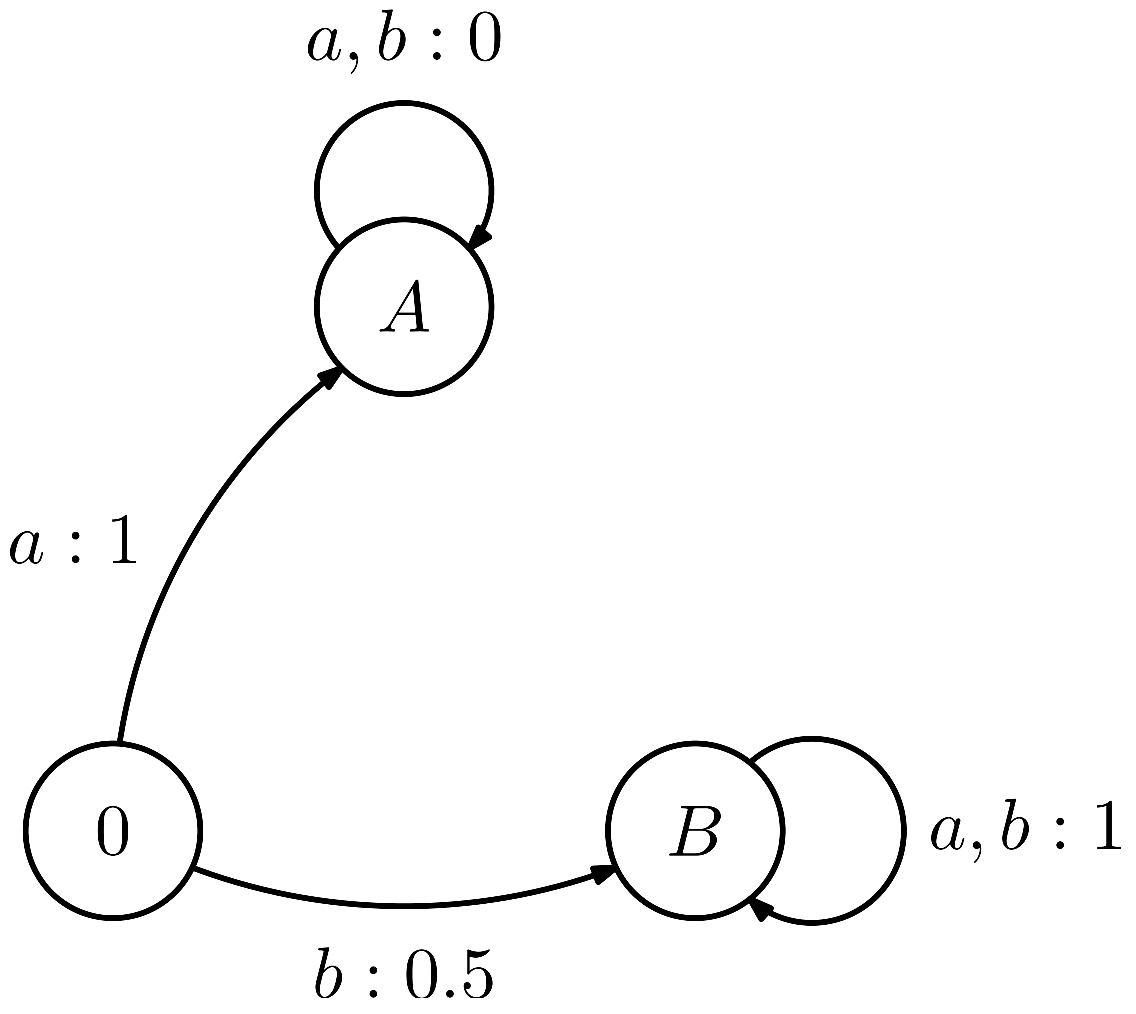
Value Iteration, 2.0

Require: MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \{\boldsymbol{P}_a\}, c, \gamma)$; tolerance $\varepsilon > 0$;

1: Initialize $k = 0$, $J^{(0)} \equiv 0$
2: **repeat**
3: $\quad J^{(k+1)} \leftarrow \mathsf{T} J^{(k)}$
4: $\quad k \leftarrow k + 1$
5: **until** $\left\| J^{(k-1)} - J^{(k)} \right\|_\infty < \varepsilon$.
6: Compute π^* using (†)
7: **return** π^*

Example

- Let us compute the optimal policy using VI
- Start with $\boldsymbol{J}^{(0)} = \mathbf{0}$



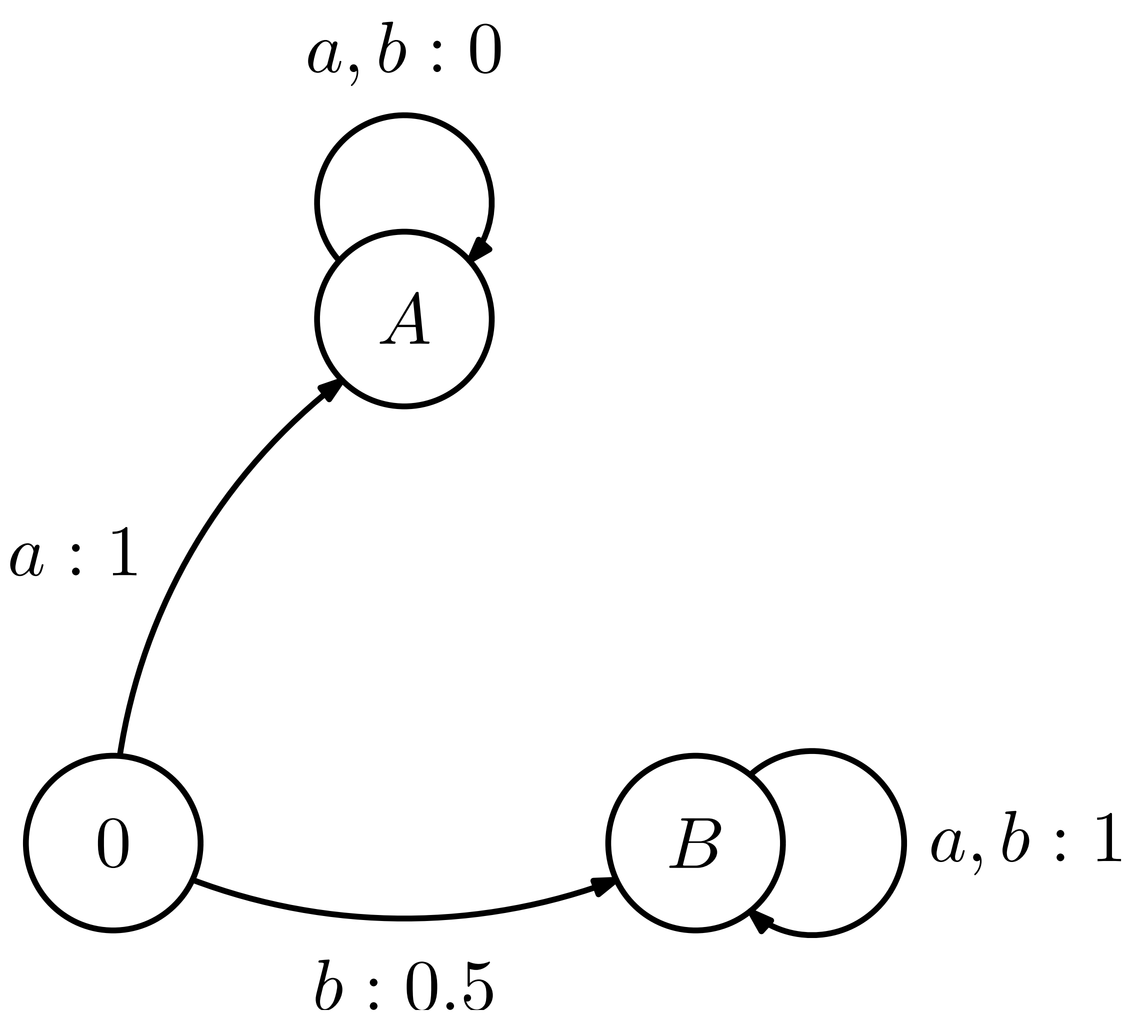
Example

- We have

$$\boldsymbol{J}^{(1)} = \min_a \left[\boldsymbol{c}_a + \gamma \mathbf{P}_a \boldsymbol{J}^{(0)} \right]$$

- For action *a*:

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



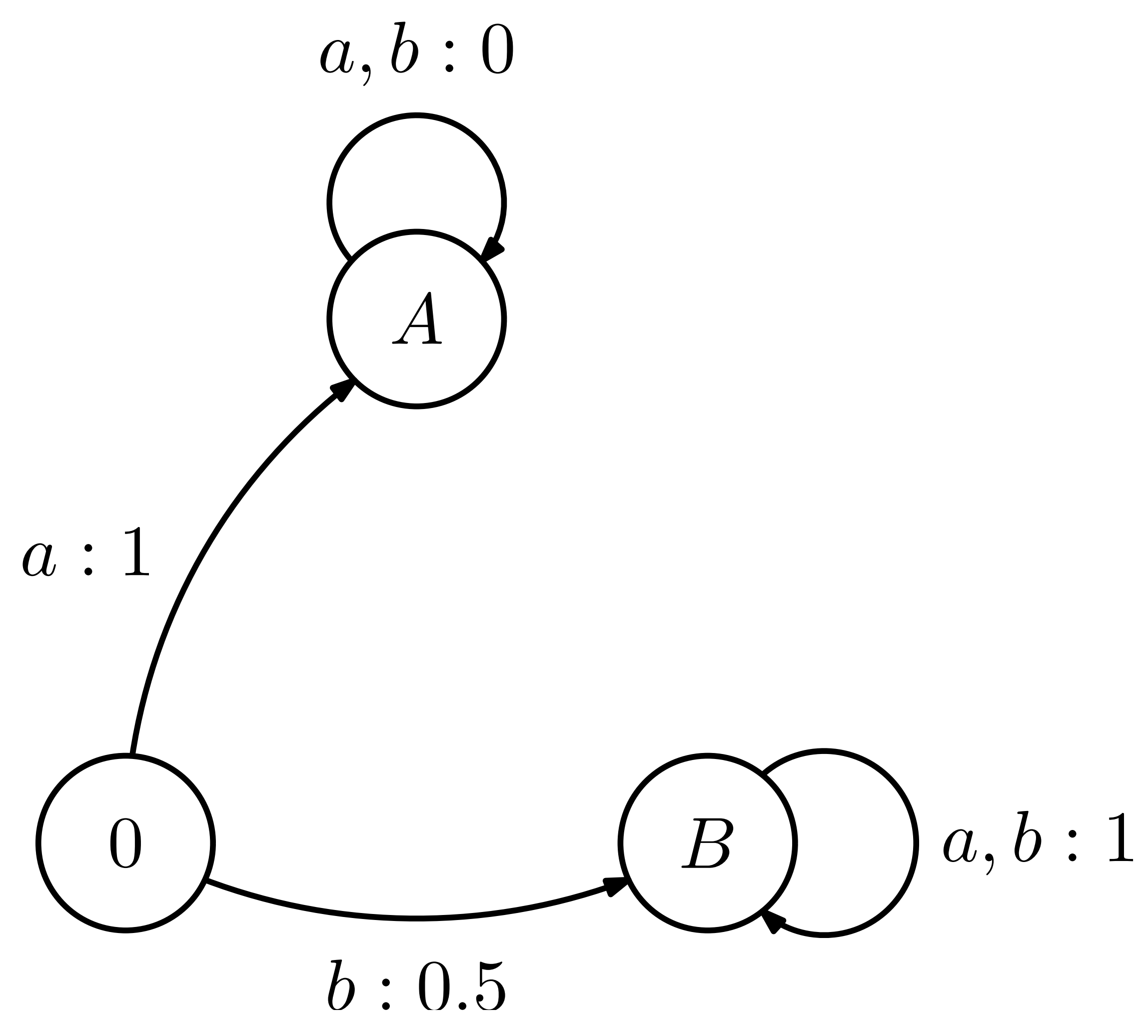
Example

- We have

$$\boldsymbol{J}^{(1)} = \min_a \left[\boldsymbol{c}_a + \gamma \mathbf{P}_a \boldsymbol{J}^{(0)}\right]$$

- For action *b*:

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



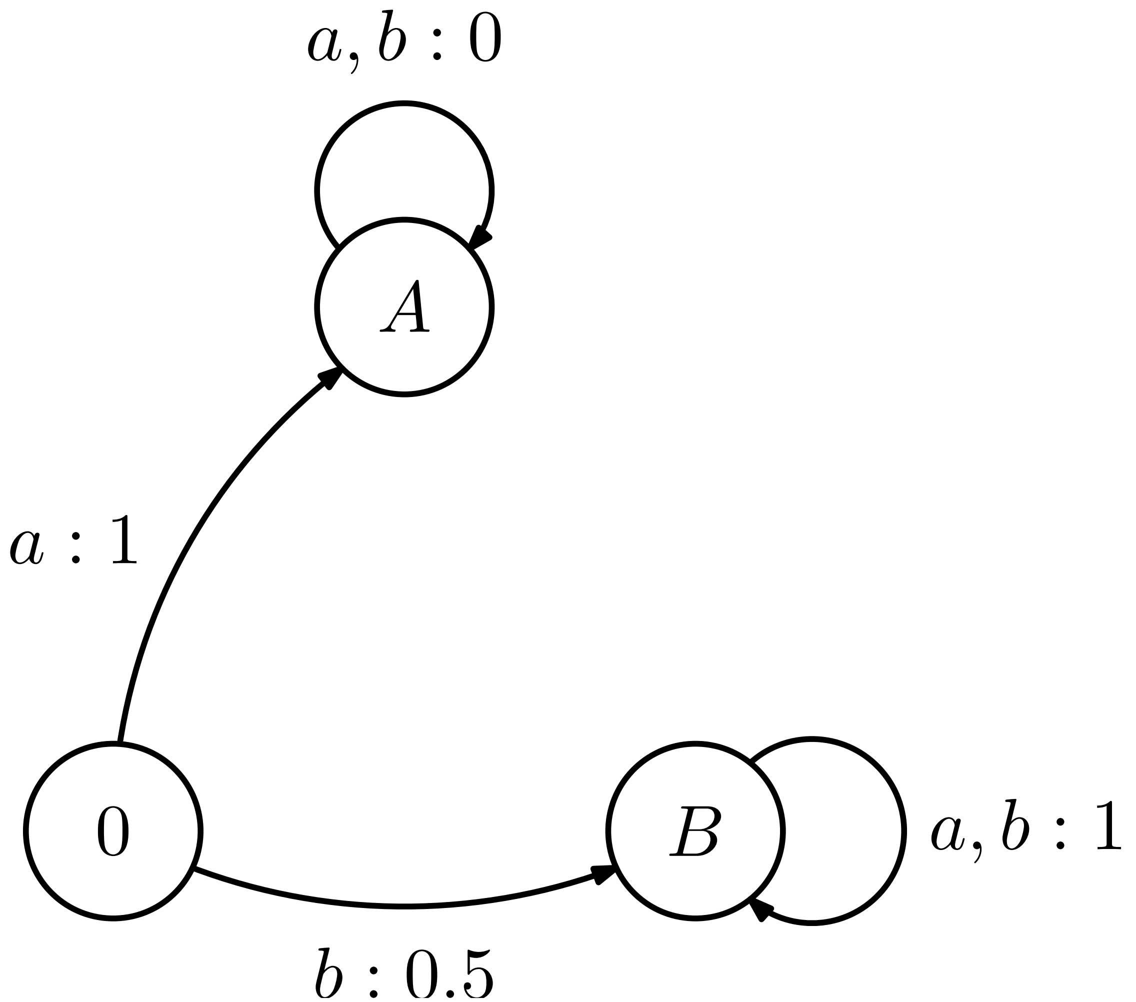
Example

- We have

$$\boldsymbol{J}^{(1)} = \min_a \left[\boldsymbol{c}_a + \gamma \mathbf{P}_a \boldsymbol{J}^{(0)}\right]$$

- Finally

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



Example

- Continuing this process…

fmelo — python — 80×30

```
Python 3.5.2 |Anaconda custom (x86_64)| (default, Jul  2 2016, 17:52:12)
[GCC 4.2.1 Compatible Apple LLVM 4.2 (clang-425.0.28)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>>
>>> ca = np.array([[1],[0], [1]])
>>> cb = np.array([[0.5],[0], [1]])
>>> Pa = np.array([[0, 1, 0], [0, 1, 0], [0, 0, 1]])
>>> Pb = np.array([[0, 0, 1], [0, 1, 0], [0, 0, 1]])
>>> gamma = 0.99
>>>
>>> J = np.zeros((3, 1))
>>> err = 1
>>> i = 0
>>>
>>> while err > 1e-8:
...     Qa = ca + gamma * Pa.dot(J)
...     Qb = cb + gamma * Pb.dot(J)
...     Jnew = np.min((Qa, Qb), axis=0)
...     err = np.linalg.norm(Jnew - J)
...     i += 1
...     J = Jnew
...
>>> print(J)
[[  1.        ]
 [  0.        ]
 [ 99.99999901]]
>>> print(i)
1834
>>>
```