Reinforcement Learning

Nicolas Maia
linkedin/NicolasSMaia

Presença

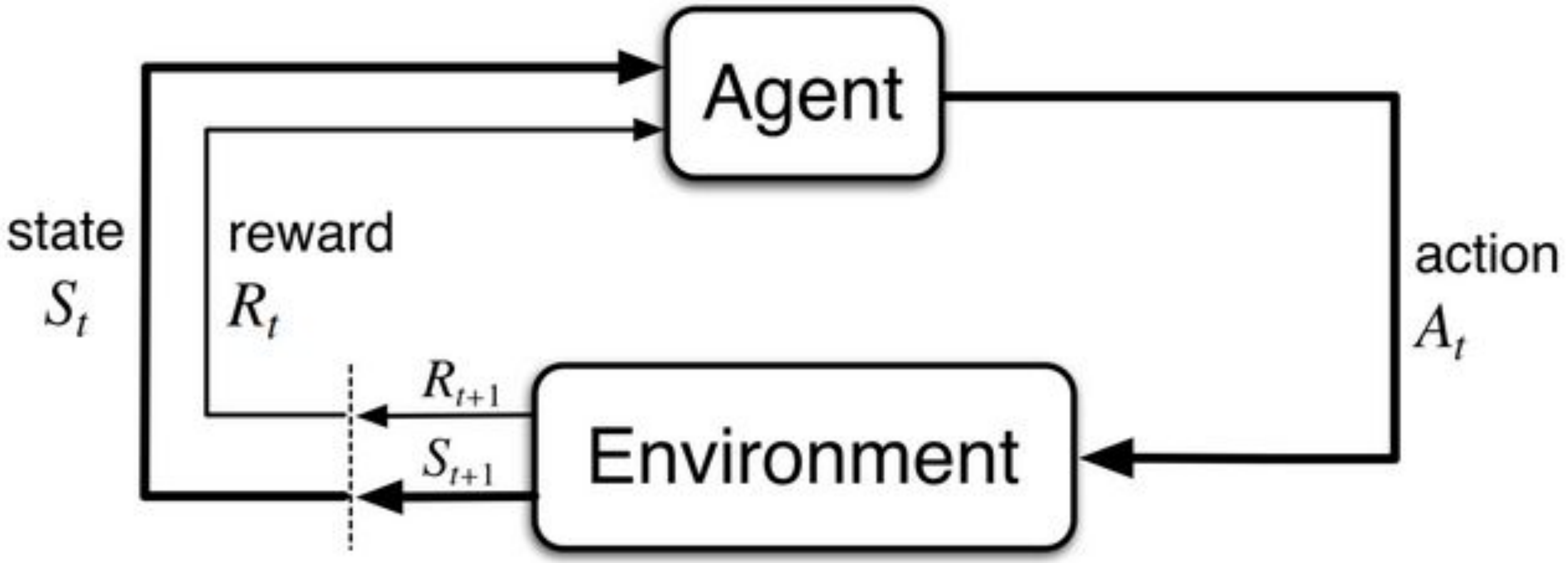
- Linktree: Presente na bio do nosso instagram
- Presença ficará disponível até 1 hora antes da próxima aula
- É necessário 70% de presença para obter o certificado

Presença

Revisão

Recapitulando

Diagrama básico do ciclo Agente-Ambiente:



- $t \in \{1, 2, 3, ..\}$
- $s \in S$
- $a \in A(s)$
- $r \in \mathbb{R}, r \neq \infty$

Recapitulando

Quatro principais elementos:

- Policy
- Recompensa
- Função valor
- Modelo

MDP

Definição

- Markov Decision Processes (ou Processos de Decisão de Markov)
- Descrevem o problema do Aprendizado por Reforço matematicamente
- Em particular, vamos discutir os MDPs finitos
- Além das recompensas imediatas, considera recompensas futuras
- A decisão é baseada também no estado atual, diferindo do Bandit Problem discutido até então

Definição

- No Bandit Problem, estimamos $q_*(a)$ para cada ação a
- Em MDPs, estimamos $q_*(s, a)$, que depende do estado atual s

O que isso significa no RL

- Agora, agente e ambiente interagem entre si seguindo:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

- Isso acontece em tempo discreto
- Estado So leva o agente a uma ação A0, que tem como retorno uma recompensa R1 e um novo estado S1, que por sua vez leva o agente a uma ação A1, e assim segue

O que isso significa no RL

- Os conjuntos de estados, ações e recompensas são finitos
- As variáveis aleatórias R_t e S_t possuem distribuições de probabilidade discretas bem definidas, dependentes apenas do estado e ação anteriores

- Para valores particulares dessas variáveis aleatórias, teremos

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

- E como p define a distribuição para cada estado e ação, temos

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \text{ para cada } s \in \mathcal{S}, a \in \mathcal{A}(s).$$

Formas de ver o problema

Hipótese da recompensa

- **Recompensa:** imediata, após mudança de estado
- **Retorno:** soma de várias recompensas tidas como esperadas no futuro
- **Objetivos do modelo:** maximização do valor esperado do Retorno

Eventos contínuos x episódicos

- **Episódicos:** o agente percorre o ambiente até atingir o estado terminal (fim de sua experiência)

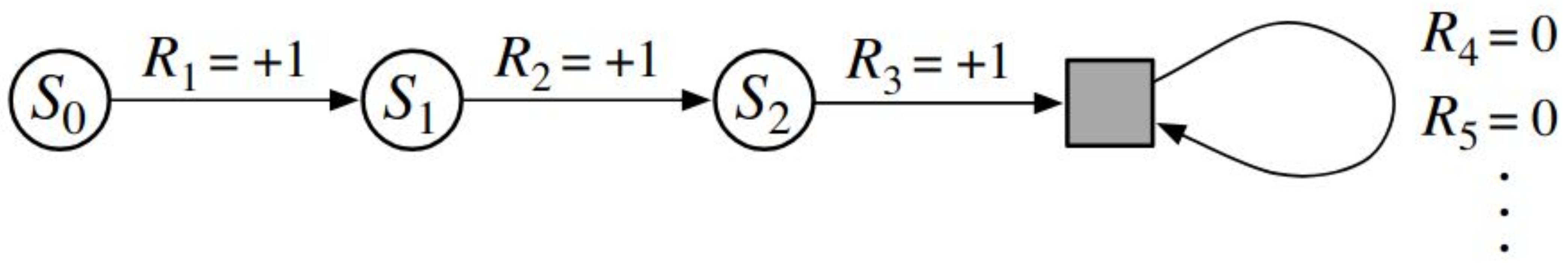
$$G_t = R_1 + R_2 + R_3 + ... + R_T = \sum_{i=1}^{T} R_i$$

- **Contínuos:** eventos sem estado terminal (agente interage infinitamente); retorno infinito e incalculável
- **Solução:** fator ‘gamma’ de desconto
- Tamanho de gamma determina pensamento a curto ou longo prazo

$$G_t = \gamma^0 R_1 + \gamma^1 R_2 + \gamma^2 R_3 + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, 0 < \gamma < 1$$

Noção Unificada

- Se considerarmos um evento episódico como um caso particular do contínuo, em que o estado terminal é um estado que “prende” o modelo para sempre com recompensa 0, é possível descrever uma fórmula generalizada de retorno



$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

Revisando as Funções

Função valor $V^\pi(s_t)$

- Cumulativa das recompensas de várias interações (longo prazo)
- Como o agente enxerga seu futuro em cada estado, seguindo uma política específica
- Deve ser estimado para problemas práticos (determiná-lo exatamente é custoso/impossível)

$$V^\pi(s) = \mathbb{E}_\pi[R_t \mid S_t = s]$$

Função ação-valor $q_\pi(s, a)$

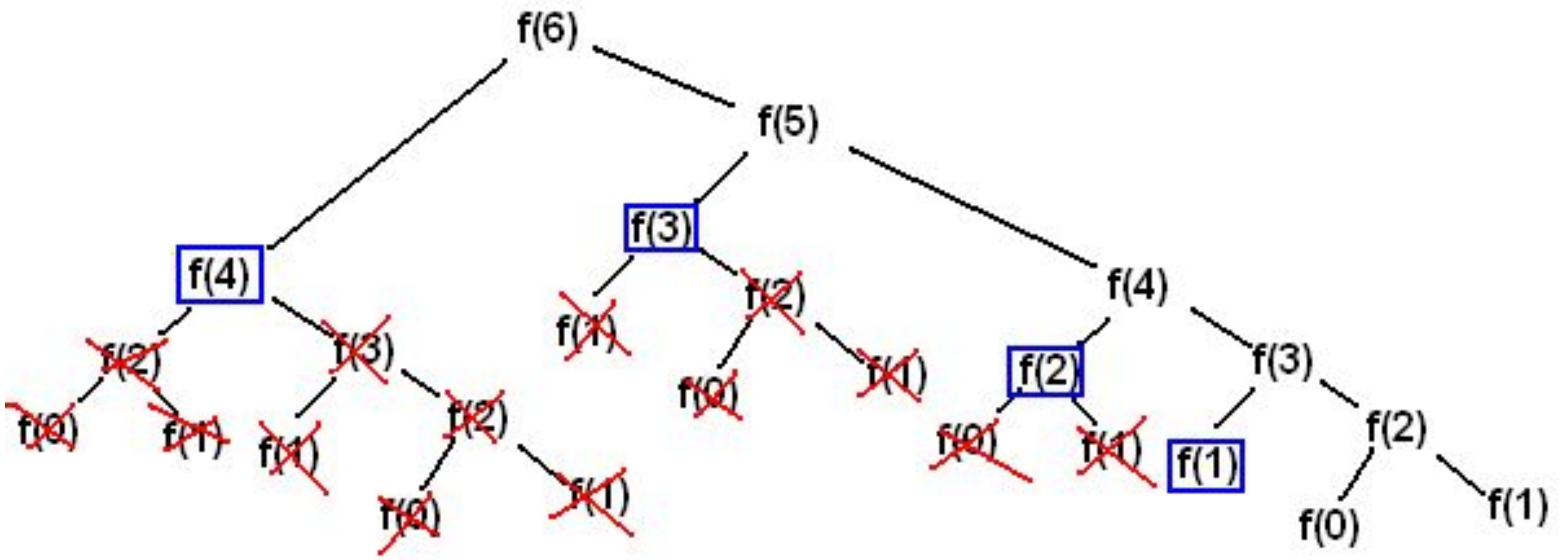
- **Ideia:** Extensão do conceito de função valor
- Além de um estado fixado, também é fixada uma ação
- "Quão bom é estar nesse estado e tomar esta ação, contida na minha política"
- Também é difícil de determinar precisamente

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

DP

Programação Dinâmica

- Geral: Quebra problemas complexos em subproblemas menores, resolve cada um e combina resultados, utilizando valores já calculados



Programação Dinâmica

Problema complexo: encontrar a função valor ótima $V*(s)$

Subproblemas mais fáceis:

1. Estimar a função valor de um único estado dado uma política
2. Encontrar uma política melhor dado uma função de valor

Melhorando a Policy

Policy Evaluation

- Processo de conseguir calcular a função $v_\pi(s)$ para uma política π

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

Policy Improvement

- Dado uma função $v_{\pi 1}(s)$ para uma política $\pi 1$, conseguimos achar uma política $\pi 2$, de forma *greedy* em relação à $v_{\pi 1}(s)$, melhor que $\pi 1$.

$policy\text{-}stable \leftarrow true$
For each $s \in \mathcal{S}$:
  $old\text{-}action \leftarrow \pi(s)$
  $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
  If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Policy Iteration

- 1º Passo: Policy Evaluation (A)
  - Calcular a função valor a partir de uma política;
- 2º Passo: Policy Improvement (M)
  - Derivar uma nova política melhor a partir da função valor;
- 3º Passo: Policy Iteration
  - Realizar iterativamente os 1º e 2º passos.

$$\pi_0 \xrightarrow{A} v_{\pi_0} \xrightarrow{M} \pi_1 \xrightarrow{A} v_{\pi_1} \xrightarrow{M} \pi_2 \xrightarrow{A} \cdots \xrightarrow{M} \pi_n \xrightarrow{A} v_{\pi_n}$$

Limitação do DP

- Aplicável em RL quando temos **modelo do ambiente**: $p(s', r \mid s, a)$

Asynchronous DP

Problema de varreduras completas

- Métodos de Programação Dinâmica (DP) tradicionais exigem varreduras completas (*sweeps*) de **todo o espaço** de estados a cada iteração

- Isso é **computacionalmente inviável** para problemas com espaços de estados gigantescos

- **Exemplo Prático:** O jogo de gamão, que possui mais de 10^{20} estados, levaria milhares de anos para completar uma única varredura

Solução: Asynchronous DP

- **Ideia Central:** Eliminar a necessidade de varreduras sistemáticas e completas

- **Como Funciona**:
  - Atualiza os valores dos estados **em qualquer ordem**, sem uma sequência fixa
  - Usa os valores mais recentes disponíveis de outros estados, mesmo que não estejam "sincronizados"
  - Alguns estados podem ser atualizados várias vezes antes de outros

- **Condição para Convergência:** Para garantir que a solução ótima seja encontrada, o algoritmo deve visitar e atualizar todos os estados eventualmente

Implicações para o RL:

- **Principal Vantagem:** Permite **intercalar o cálculo com a interação** em tempo real do agente no ambiente (por isso, assíncrono)

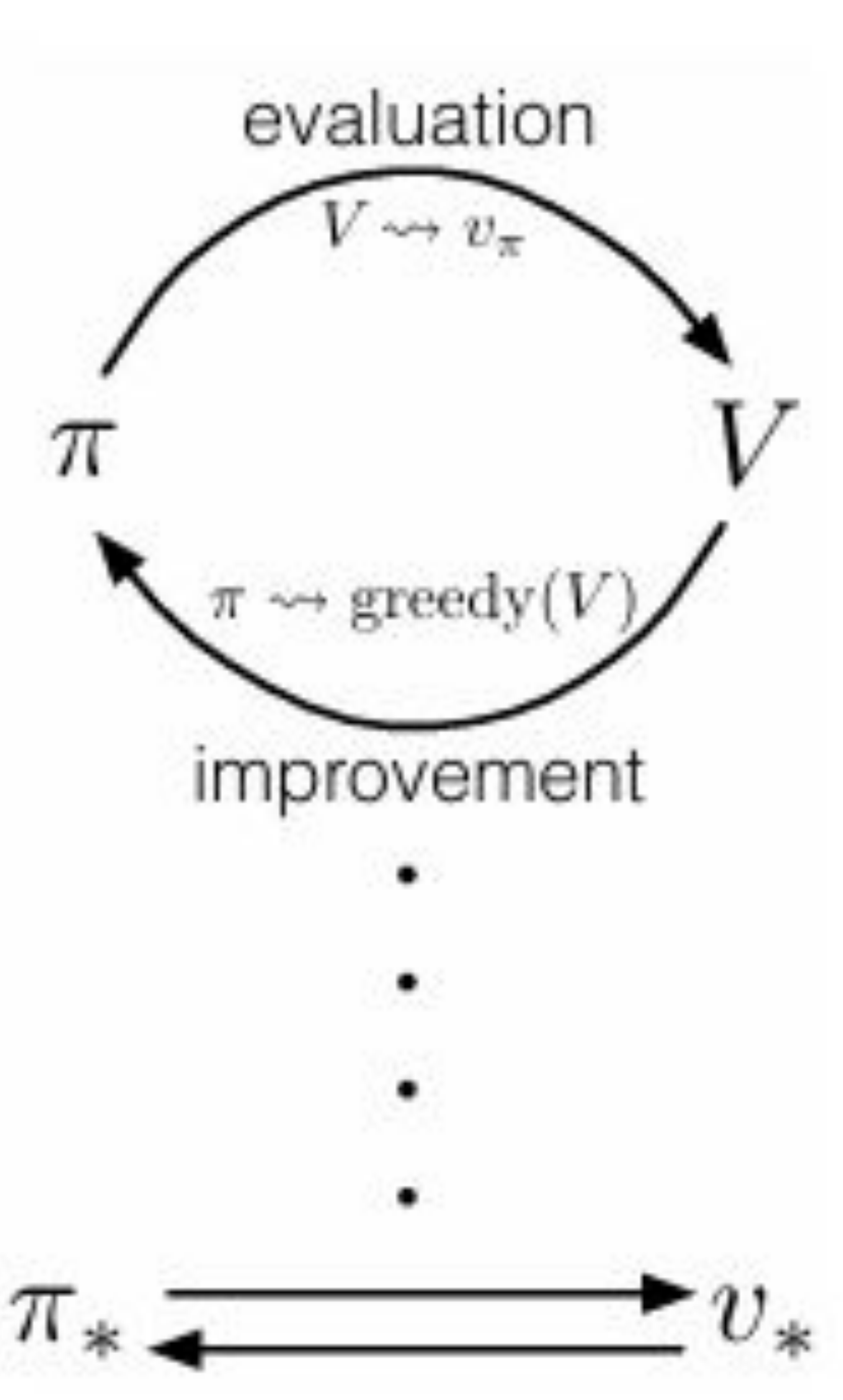
- O algoritmo pode focar as atualizações de valor apenas nos estados que o agente **realmente visita**

- Isso introduz um conceito fundamental em RL: Focar o **esforço computacional** nas **partes mais relevantes** do problema.

GPI

Iteração de Política Generalizada (GPI)

- **Conceito Central:** Framework que descreve a **estrutura da maioria dos algoritmos** de Aprendizado por Reforço
- **Ideia Principal:** Uma **interação** contínua **entre dois processos** que buscam, juntos, uma solução ótima
- Os dois processos são a **Avaliação** e a **Melhoria** da política:



Dinâmica da Convergência

No curto prazo, os processos **competem**:

- **Melhorar a política** torna a função de valor "incorreta"
- **Atualizar a função** de valor torna a política "sub-ótima"

No longo prazo, eles **cooperam**:

- Essa **"dança" entre os dois processos** move o sistema em direção à solução ótima (π_* e v_*)

Convergência: Ocorre quando a política se torna **estável**, ou seja, já **é gananciosa em relação à sua própria função** de valor

Análise

Eficiência e Memória

Se temos um problema com **n** estados e **k** possíveis ações

Eficiência e Memória

Se temos um problema com **n** estados e **k** possíveis ações

Busca **completa** do espaço de soluções é exponencial **kˆn**

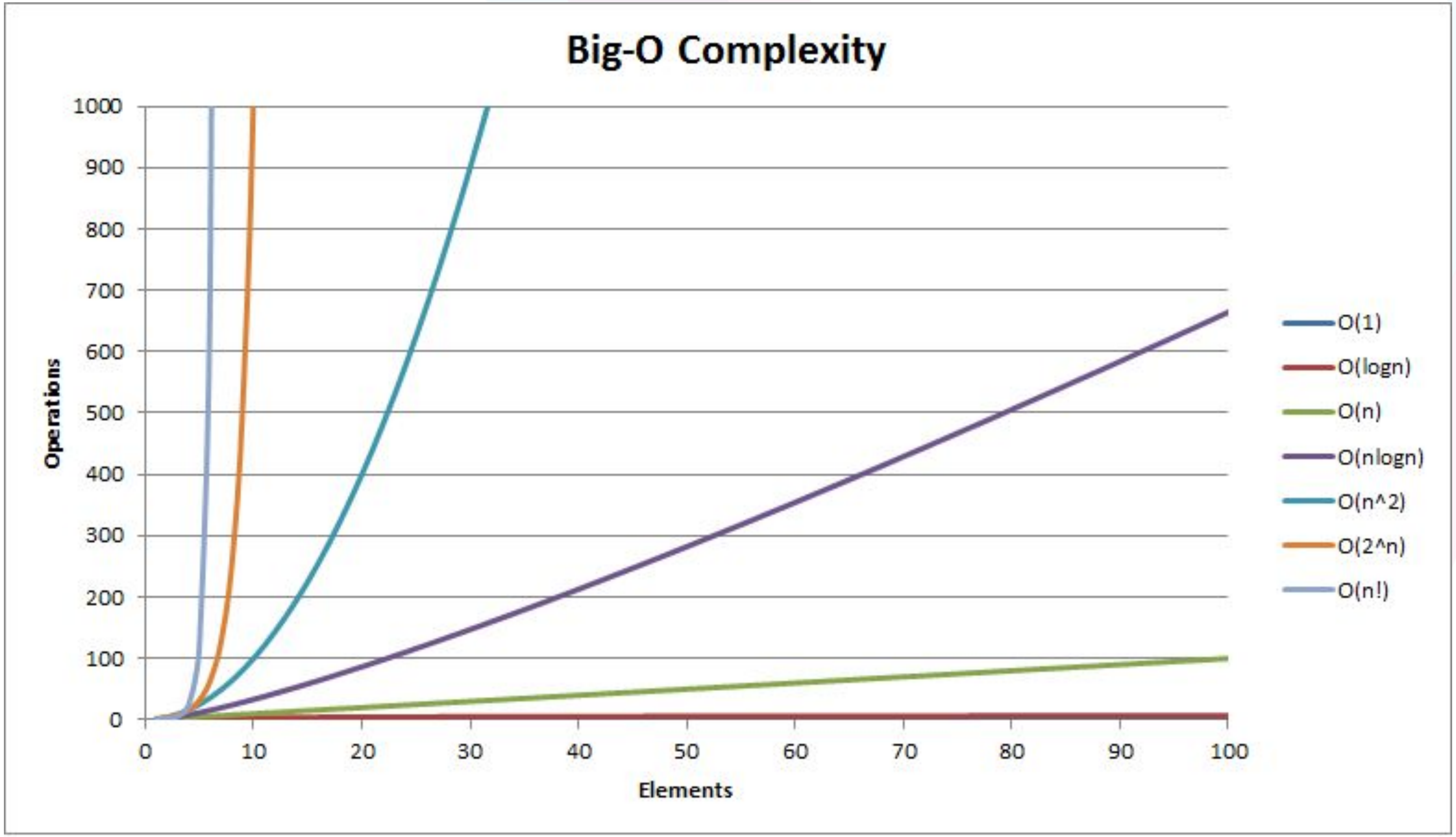
Eficiência e Memória

Se temos um problema com **n** estados e **k** possíveis ações

Busca **completa** do espaço de soluções é exponencial **k^n**

Com **DP** é garantido achar solução em tempo polinomial de **n** e **k**

Eficiência e Memória



Prática

Estados

Temperatura:

- 0 a 60

Nível da Água

- 0 a 60

Ações

- Mudar o volume da água no intervalo [-6, 6]

Ambiente

- Agente tem uma chance de temperatura/60 de ganhar +1
- Se temperatura=60 o agente perde
- A temperatura muda seguindo:

$$T_{t+1} = T_t + 0.1 * T_t - 0.2 * agua$$

- Gamma = 0,99

Analisando o Algoritmo

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

“Reinforcement Learning: An Introduction” Barto & Sutton, pag. 80

data@icmc.usp.br

@data.icmc

/c/DataICMC

/icmc-data

data.icmc.usp.br

obrigado!