UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Intelligenza Artificiale

Pianificazione mediante programmazione dinamica

a.a. 2022/2023

Outline

- Introduzione
- Policy Evaluation
- Policy Iteration
- Value Iteration
- Estensioni della Programmazione Dinamica

Cos'è la Programmazione Dinamica

- Introdotta nel 1953 da Richard Bellman
- Dinamica ↦ problema con componente sequenziale o temporale
- Programmare ↦ ottimizzare un «programma», cioè una policy
- Un metodo per risolvere problemi complessi suddividendoli in sottoproblemi (in modo ricorsivo)
  - Risolvere i sottoproblemi
  - Combinare le soluzioni dei sottoproblemi
- Non è divide et impera
  - Si differenzia per la decomposizione con overlap

Requisiti della Programmazione Dinamica

- La programmazione dinamica (DP) è un metodo generale per la risoluzione di problemi caratterizzati da due proprietà:
  - Sottostruttura ottimale
    - Si applica il *principio di ottimalità*
    - La soluzione ottima può essere suddivisa in sottoproblemi
  - Overlapping dei sottoproblemi
    - I sottoproblemi ricorrono più volte
    - Le soluzioni possono essere memorizzate nella cache e riutilizzate
- I processi decisionali di Markov soddisfano entrambe le proprietà
  - La Bellman Equation consente una suddivisione ricorsiva
  - La value function memorizza e riutilizza le soluzioni

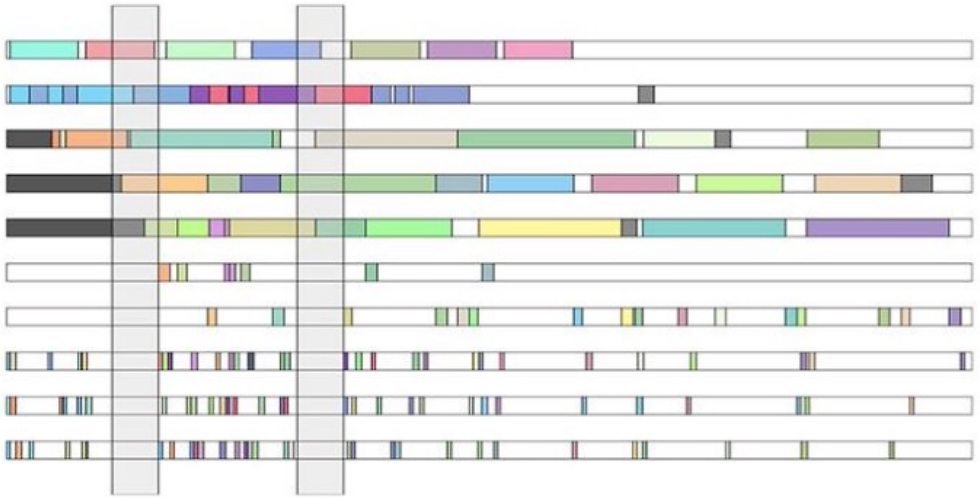
Pianificazione tramite Programmazione Dinamica

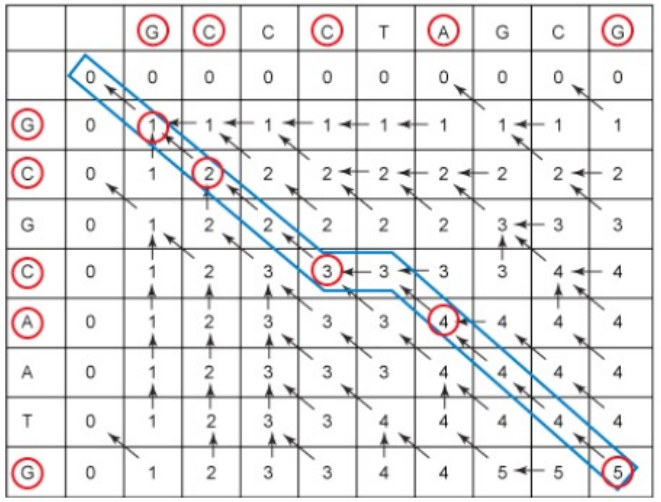
- La programmazione dinamica assume la piena conoscenza del MDP
- Viene utilizzata per la pianificazione in un MDP
  - Un modello dell'ambiente è noto
  - L'agente migliora la sua policy

- Per le predizioni:
  - Input: MDP $\langle S, A, \mathbf{P}, R, \gamma \rangle$ *ed una policy* π
  - Output: value function v_π
- Per il controllo:
  - Input: MDP $\langle S, A, \mathbf{P}, R, \gamma \rangle$
  - Output: optimal value function v_* **e** optimal policy π_*

Altre applicazioni della Programmazione Dinamica

- La programmazione dinamica viene utilizzata per risolvere molti problemi:
  - Algoritmi di pianificazione
  - Algoritmi su stringhe (ad esempio, allineamento di sequenze)
  - Algoritmi su grafi (ad esempio, calcolo del percorso più breve)
  - Modelli basati su grafi (ad esempio, algoritmo di Viterbi)
  - Bioinformatica (ad esempio, modelli reticolari)





Policy Evaluation

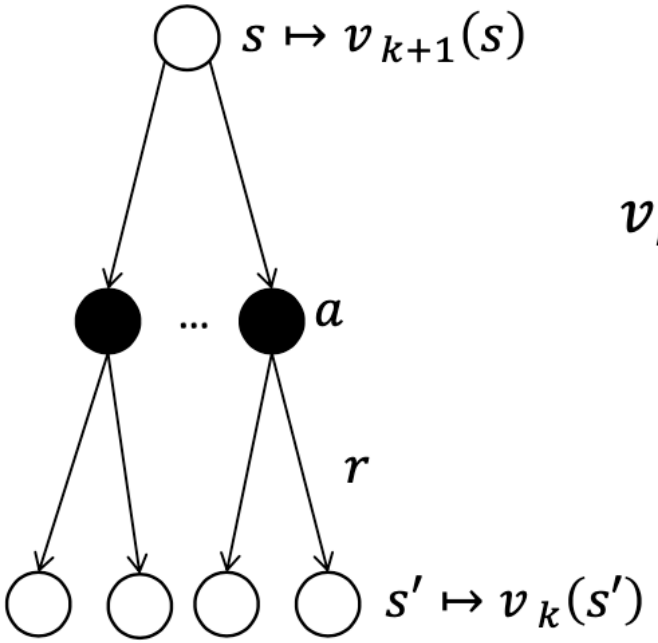
Iterative Policy Evaluation

- Problema: valutare una policy π
- Soluzione: applicazione iterativa del backup della Bellman Expectation

$$v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$$

- Utilizzando backup sincroni,
  - Ad ogni iterazione *k + 1*
  - Per tutti gli stati s ∈ S
  - Aggiornare $v_{k+1}(s)$ da $v_k(s')$
    - dove s' è uno stato successore di s

Iterative Policy Evaluation (2)



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_k(s') \right)$$

$$v_{k+1} = \mathcal{R}^\pi + \gamma \boldsymbol{P}^\pi v_k$$

Iterative Policy Evaluation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(terminal)$ to 0

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma V(s') \right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Valutazione di una Policy casuale nell'esempio Gridworld

actions

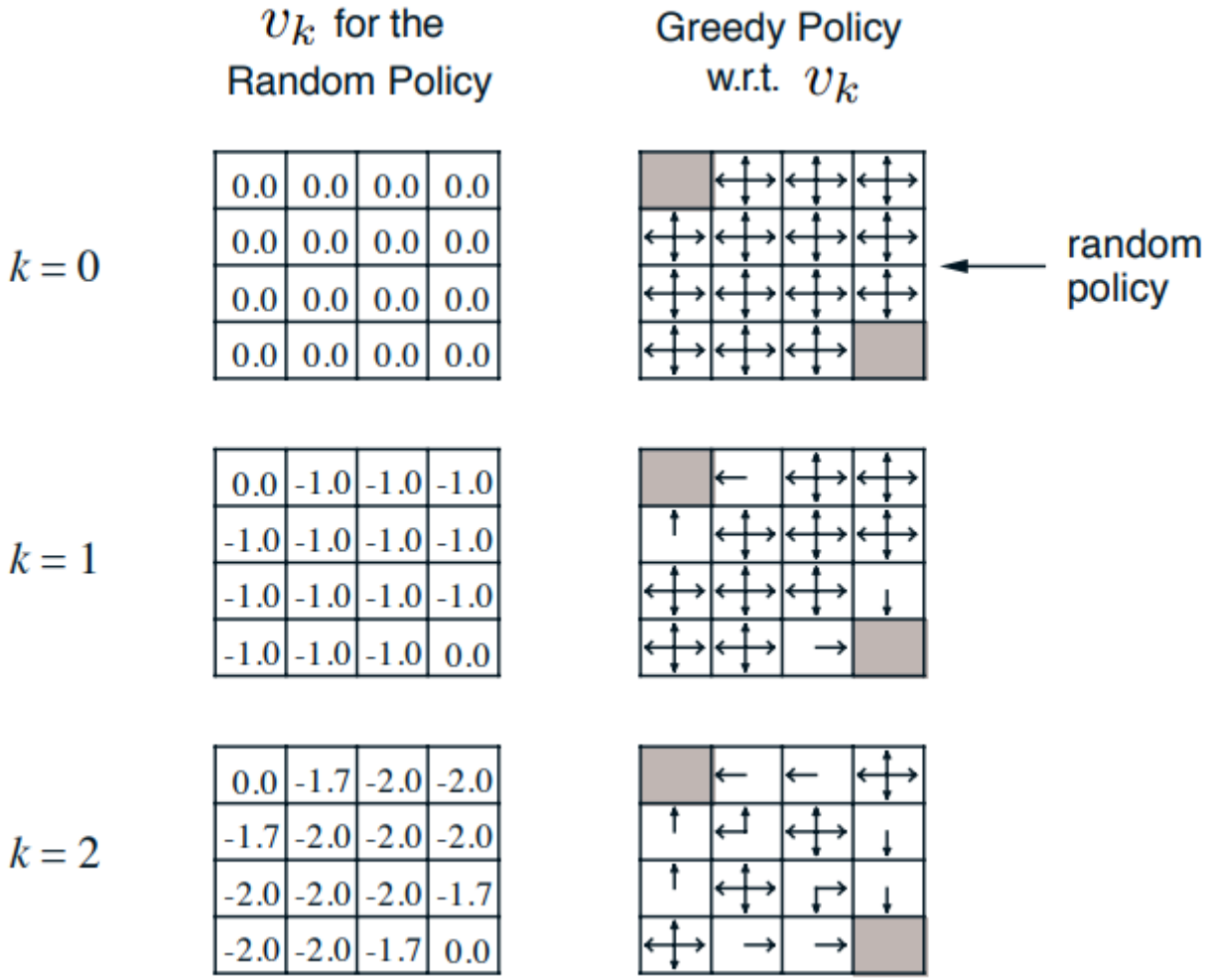
	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R_t = -1$
on all transitions

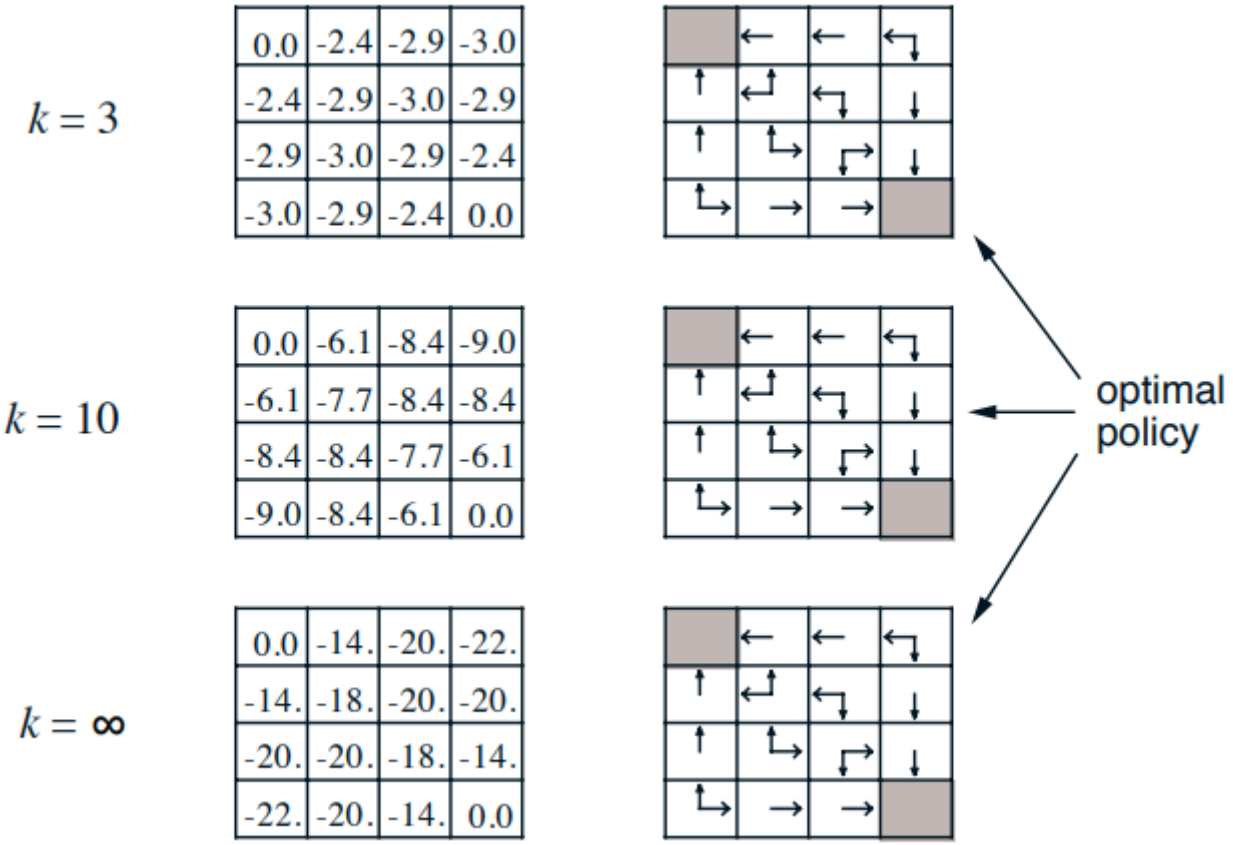
- MDP episodica non scontata (γ = 1)
- Stati non terminali 1, …, 14
- Uno stato terminale (mostrato due volte come cella ombreggiata)
- Le azioni che portano fuori dalla griglia lasciano lo stato invariato
- La ricompensa è -1 fino a quando non viene raggiunto lo stato terminale
- L'agente segue una policy casuale uniforme

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

Iterative Policy Evaluation nell'esempio Gridworld



Iterative Policy Evaluation nell'esempio Gridworld (2)



Policy Iteration

Come migliorare una Policy

- Data una policy π
  - Valutare la policy π

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s\right]$$

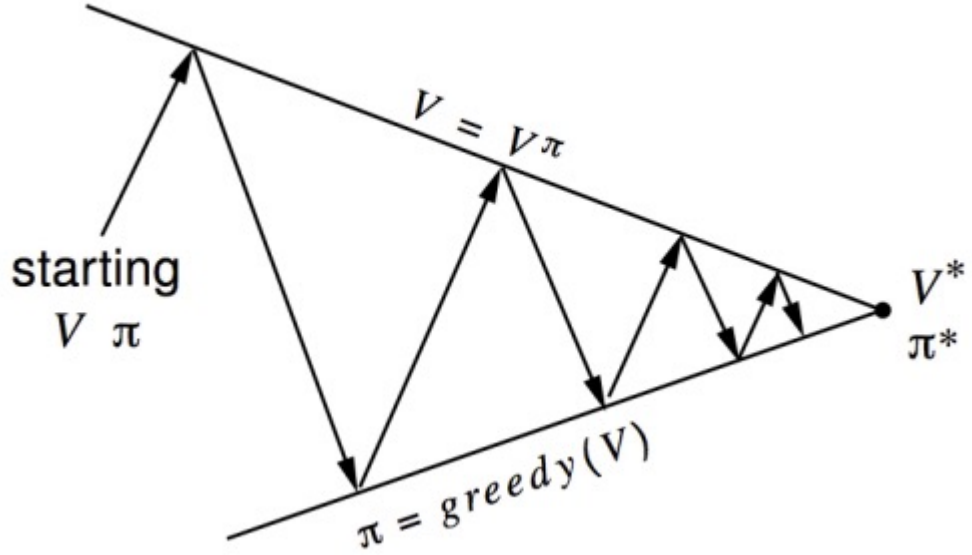
  - Migliorare la policy agendo in maniera greedy rispetto a v_π

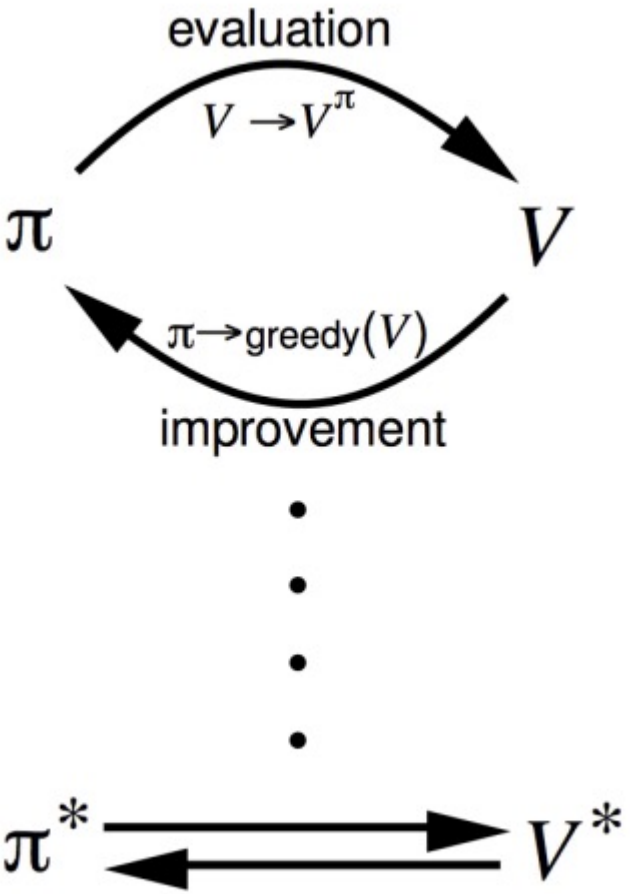
$$\pi' = \text{greedy}(v_\pi)$$

- Nell'esempio Gridworld la policy migliorata era ottimale, $\pi' = \pi^*$
- In generale, sono necessarie diverse iterazioni di miglioramento / valutazione
- Tuttavia, questo processo di Policy Iteration converge sempre a π_*

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

Policy Iteration





Valutazione della Policy Stima di v_π

Iterative Policy Evaluation

Miglioramento della Policy Generazione di $\pi' \geq \pi$

Greedy Policy Improvement

Policy Improvement

- Consideriamo una policy deterministica a = π(s)
- Possiamo *migliorare* la policy agendo in modo greedy

$$\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_\pi(s, a)$$

- Questo *migliora il valore di qualsiasi stato s* rispetto ad uno step,

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

- Pertanto, migliora la value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s\right] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s\right] \\
&= v_{\pi'}(s).
\end{aligned}$$

Policy Improvement (2)

- Se i miglioramenti terminano,

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

- Allora la Bellman Optimality Equation è stata soddisfatta

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

- Pertanto, $v_\pi(s) = v_*(s)$ per tutti gli $s \in S$
- Quindi π è una policy ottimale

Policy Iteration

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

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(\mathit{terminal}) \doteq 0$

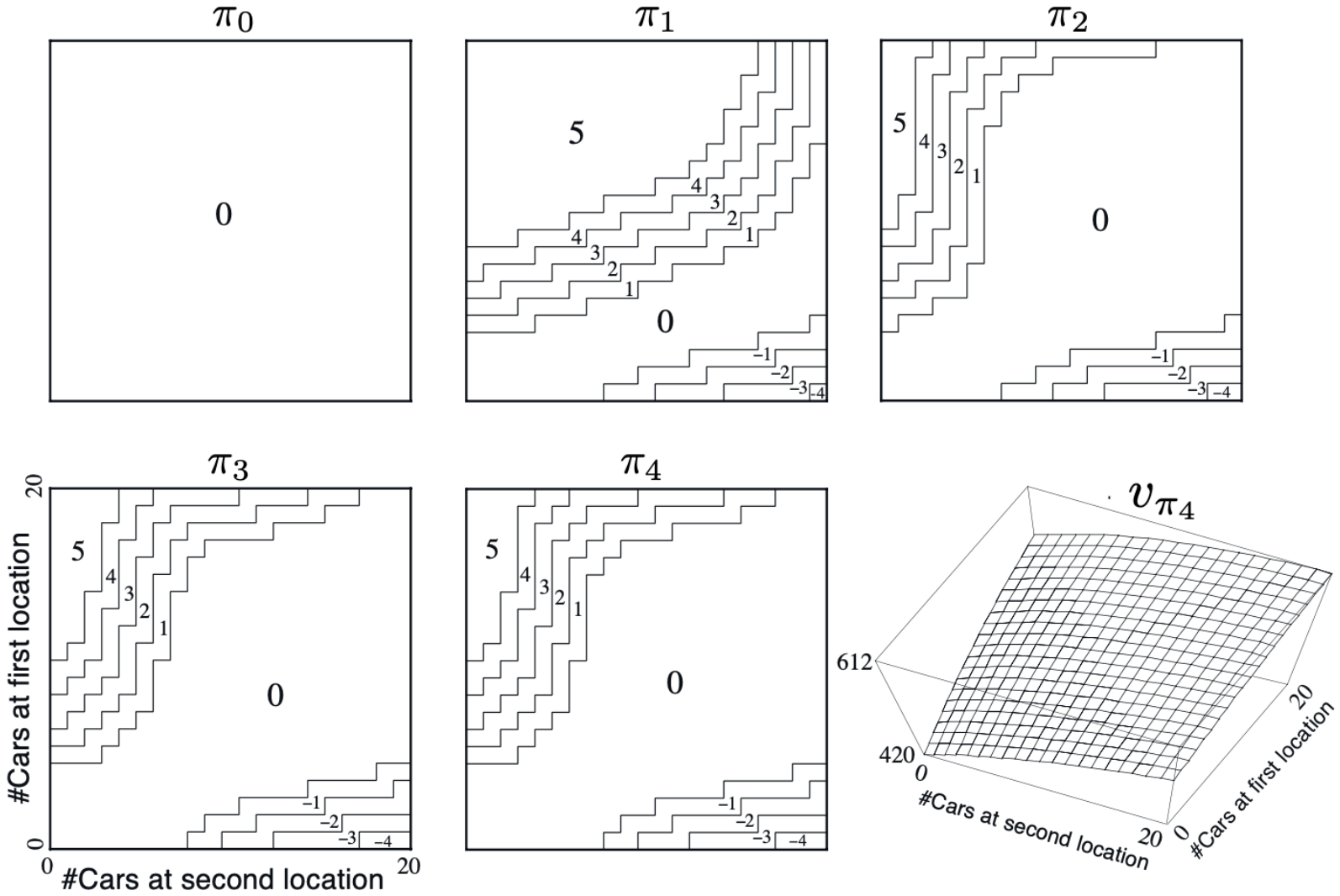
2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number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

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$
   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

Esempio: Autonoleggio

- Stati: Due sedi, massimo 20 auto in ciascuna sede
- Ricompensa: 10$ per ogni auto noleggiata (deve essere disponibile)
- Azioni: Spostare fino a 5 auto da una sede all'altra durante la notte (costo 2$ per ogni auto)
- Transizioni: Auto restituite e richieste in modo casuale
  - Distribuzione di Poisson, *n* restituzioni/richieste con probabilità $\sim \frac{\lambda^n}{n!} e^{-\lambda}$
  - Prima sede: media delle richieste = 3, media delle restituzioni = 3
  - Seconda sede: media delle richieste = 4, media delle restituzioni = 2

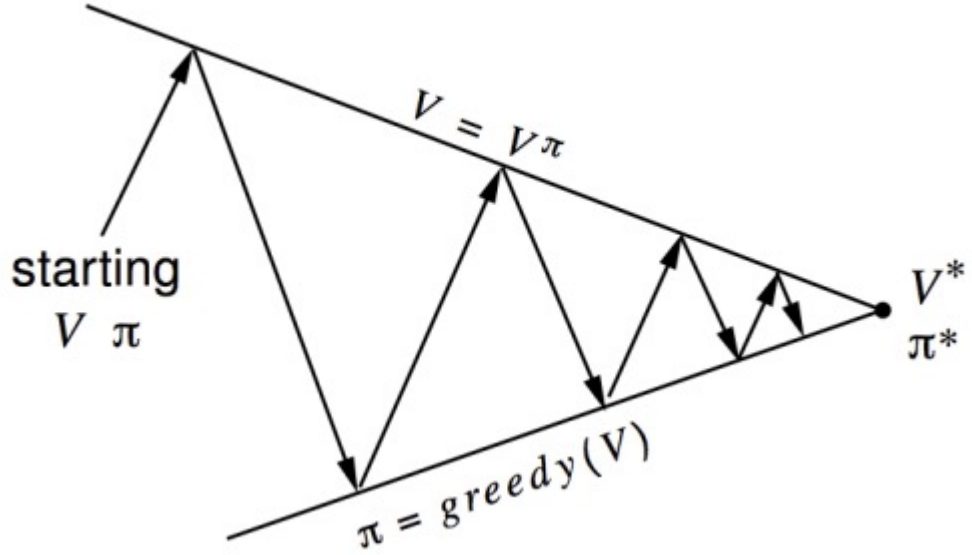
Policy Iteration

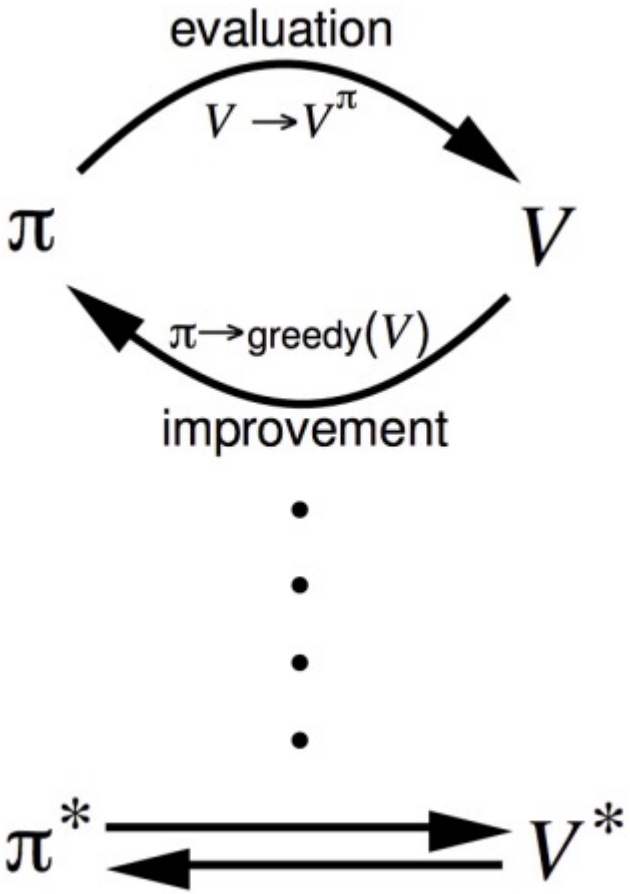


Modified Policy Iteration

- La policy evaluation deve convergere verso $v_{\pi *}$?
  - Dobbiamo introdurre una condizione di arresto?
  - Ad esempio, ε-convergence della value function
  - Oppure dobbiamo semplicemente fermarci dopo k iterazioni della policy evaluation?
  - Ad esempio, nell'esempio gridworld *k = 3* è stato sufficiente per ottenere una policy ottimale
- Perché non aggiornare la policy ad ogni iterazione? cioè fermarsi dopo *k = 1*
  - Ciò corrisponde alla value iteration (mostrata di seguito)

Policy Iteration Generalizzata





Valutazione della Policy Stima di v_π

Qualsiasi policy evaluation

Miglioramento della Policy Generazione di $\pi' \geq \pi$

Qualsiasi algoritmo di policy improvement

Value Iteration

Principio di Ottimalità

- Ogni policy ottimale può essere suddivisa in due componenti:
  - Una prima azione ottimale A_*
  - Seguita da una policy ottimale dallo stato successore s'

Teorema (Principio di Ottimalità)

- Una policy $\pi(a|s)$ *raggiunge il valore ottimale* dallo stato s (cioè $v_\pi(s) = v_*(s)$) se e solo se per ogni stato s' raggiungibile da s
  - π raggiunge il valore ottimale dallo stato s', $v_\pi(s') = v_*(s')$

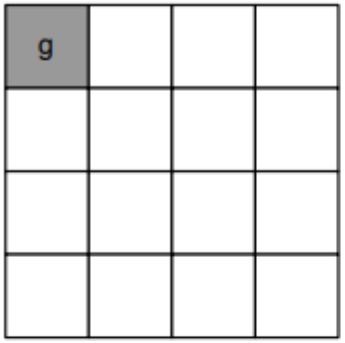
Deterministic Value Iteration

- Se si conosce la soluzione dei sottoproblemi $v_*(s')$
- Allora la soluzione $v_*(s)$ può essere identificata con un lookahead di un solo passo

$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

- L'idea della value iteration consiste nell'applicare questi aggiornamenti iterativamente
- Intuizione: iniziare con le ricompense finali e lavorare all'indietro
  - Funziona anche con MDP stocastici

Esempio: Shortest Path



Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

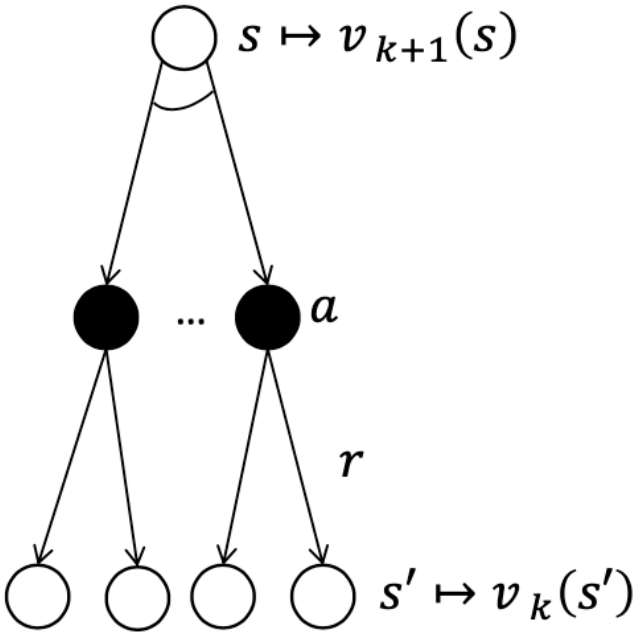
Value Iteration

- Problema: trovare la policy ottimale π
- Soluzione: applicazione iterativa del backup della Bellman Optimality

$$v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_*$$

- Utilizzando backup sincroni,
  - Ad ogni iterazione k + 1
  - Per tutti gli stati s ∈ S
  - Aggiornare v_{k+1} da $v_k(s')$
- A differenza della policy iteration, non c'è una policy esplicita
- Le intermediate value function potrebbero non corrispondere a nessuna policy

Value Iteration (2)



$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_k(s') \right)$$

$$v_{k+1} = \max_{a \in \mathcal{A}} (\boldsymbol{\mathcal{R}}^a + \gamma \boldsymbol{P}^a v_k)$$

Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

Esempio Programmazione Dinamica

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Algoritmi di programmazione dinamica sincrona

Problema	Bellman Equation	Algoritmo
Predizione	Bellman Expectation Equation	Iterative Policy Evaluation
Controllo	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Controllo	Bellman Optimality Equation	Value Iteration

- Gli algoritmi sono basati sulla state-value function $v_\pi(s)$ o $v_*(s)$
  - Complessità $O(mn^2)$ per iterazione, per m azioni ed n stati
- Potrebbero essere applicati anche alla action-value function $q_\pi(s, a) = q_*(s, a)$
  - Complessità $O(m^2n^2)$ per iterazione

Estensioni

Programmazione dinamica asincrona

- I metodi di DP descritti finora utilizzano backup *sincroni*
  - Tutti gli stati sono sottoposti a backup parallelamente
  - Se l'insieme di stati è molto ampio, anche un singolo sweep può essere proibitivo. Ad esempio nel backgammon 10^{20} stati.
- La DP asincrona esegue il backup degli stati singolarmente, in qualsiasi ordine
  - Per ogni stato selezionato, bisogna applicare un backup appropriato
  - Può ridurre significativamente l'elaborazione
  - Convergenza garantita se tutti gli stati continuano ad essere selezionati

Programmazione dinamica asincrona

- Tre semplici approcci:
  - *Programmazione dinamica in-place*
  - *Prioritised sweeping*
  - *Programmazione dinamica real-time*

Programmazione dinamica in-place

- La value iteration sincrona memorizza due copie della value function

  Per ogni s in S

$$v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_{old}(s')$$

$$v_{old}(s) \leftarrow v_{new}(s)$$

- La value iteration in-place memorizza solo una copia della value function

  Per ogni s in S

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right)$$

Prioritised Sweeping

- Utilizza il Bellman error per guidare la selezione degli stati, ad esempio

$$\left| \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right) - v(s) \right|$$

- Esegue il backup dello stato con il Bellman error residuo più grande
- Aggiorna il Bellman error degli stati interessati dopo ogni backup
- Richiede la conoscenza delle dinamiche inverse (stati predecessori)
- Può essere implementato in modo efficiente tramite l'utilizzo di una coda di priorità

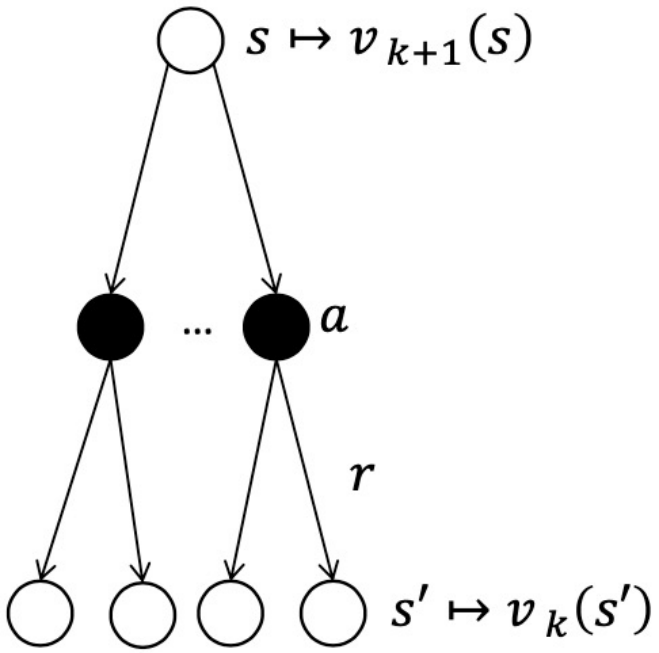
Programmazione dinamica real-time

- Idea: considerare solo gli stati rilevanti per l'agente
- Utilizza l'esperienza dell'agente per guidare la selezione degli stati
  - Dopo ogni time-step S_t, A_t, R_{t+1}
  - Esegue il backup dello stato S_t

$$v(S_t) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}^a_{S_t} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{S_t s'} v(s') \right)$$

Backup Full-Width

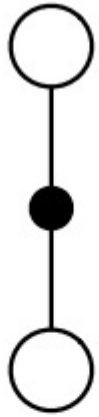
- La DP utilizza backup *full-width*
- Per ogni backup (sincrono o asincrono)
  - Viene considerato ogni stato e azione successiva
  - Si utilizza la conoscenza delle transizioni del MDP e della funzione di ricompensa
- La DP è efficace per problemi di medie dimensioni (milioni di stati)
- Per problemi di grandi dimensioni la DP è soggetta alla *curse of dimensionality*
  - Il numero di stati $n = |S|$ cresce esponenzialmente con il numero di variabili di stato
- Anche un solo backup può essere troppo costoso

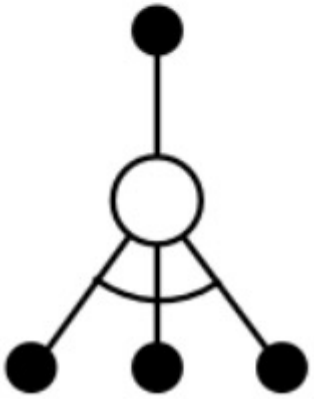


Backup Sample

- Utilizzo di ricompense e transizioni campionarie

  $<S, A, R, S'>$
- Al posto della funzione di ricompensa R e delle transizioni dinamiche P
- Vantaggi:
  - Model-free: non è richiesta alcuna conoscenza prelimina del MDP
  - Evita la curse of dimensionality attraverso il campionamento
  - Il costo del backup è costante, indipendente da $n = |S|$





Programmazione dinamica approssimata

- Approssima la value function
  - Utilizza un approssimatore di funzioni $\hat{v}(s, \mathbf{w})$
  - Applica la programmazione dinamica a $\hat{v}(\cdot, \mathbf{w})$
- Ad esempio, la Fitted Value Iteration si ripete ad ogni iterazione k,
  - Stati sample $\tilde{S} \subseteq S$
  - Per ogni stato s $\in \tilde{S}$, viene stimato il valore target utilizzando la Bellman Optimality Equation

$$\tilde{v}_k(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \hat{v}(s', \mathbf{w_k}) \right)$$

  - Addestra la value function successiva $\hat{v}(\cdot\,; \boldsymbol{w}_{k+1})$ usando i target $\{(s, \hat{v}_k(s))\}$