Aprendizaje por refuerzo

Programación dinámica

Antonio Manjavacas Lucas

manjavacas@ugr.es

Índice

Programación dinámica

Programación dinámica

La programación dinámica (*dynamic programming*, **DP**) compende un conjunto de algoritmos empleados para resolver problemas complejos dividiéndolos en subproblemas más pequeños.

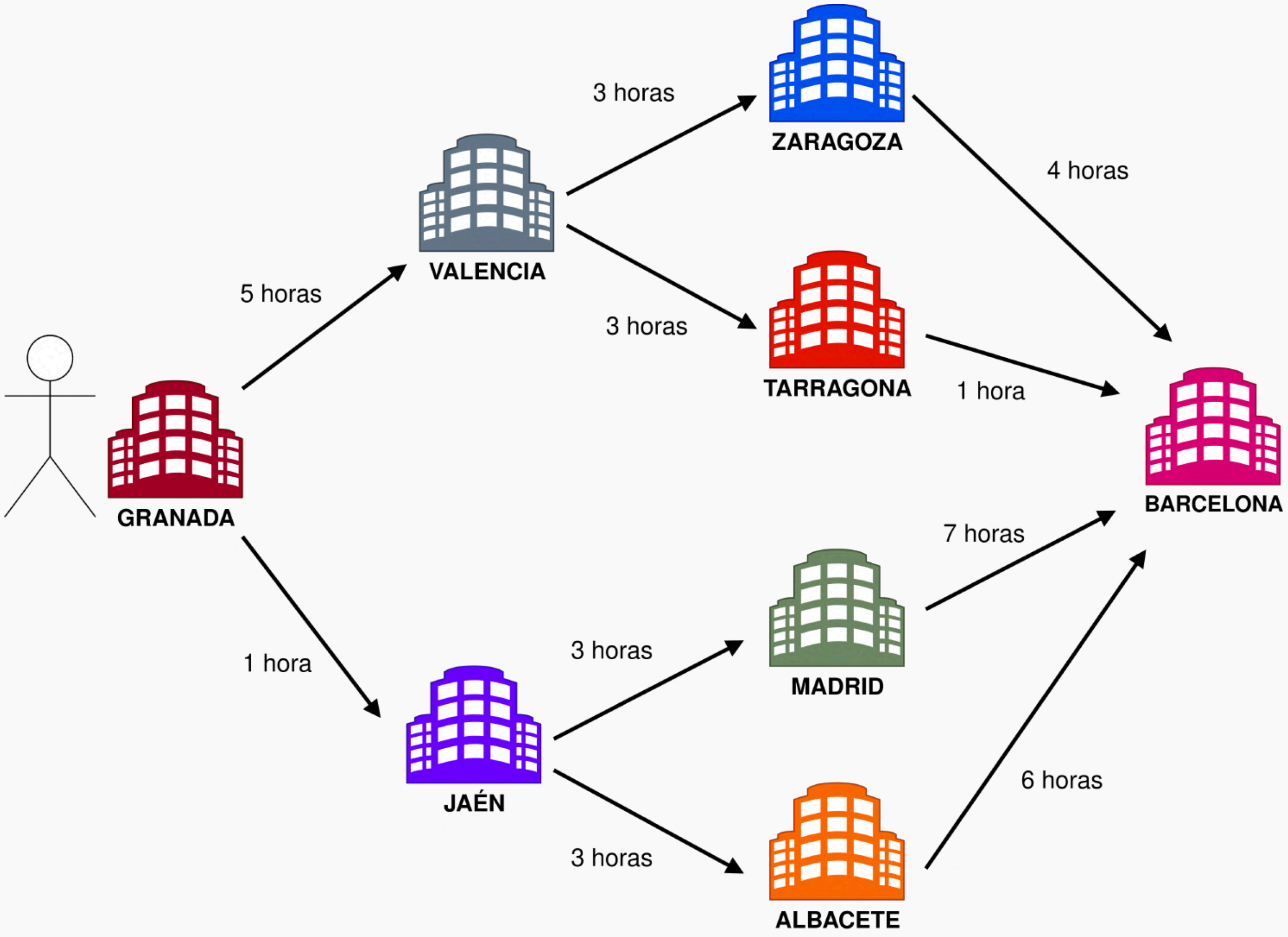
Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions.

THE THEORY OF DYNAMIC PROGRAMMING

Richard Bellman

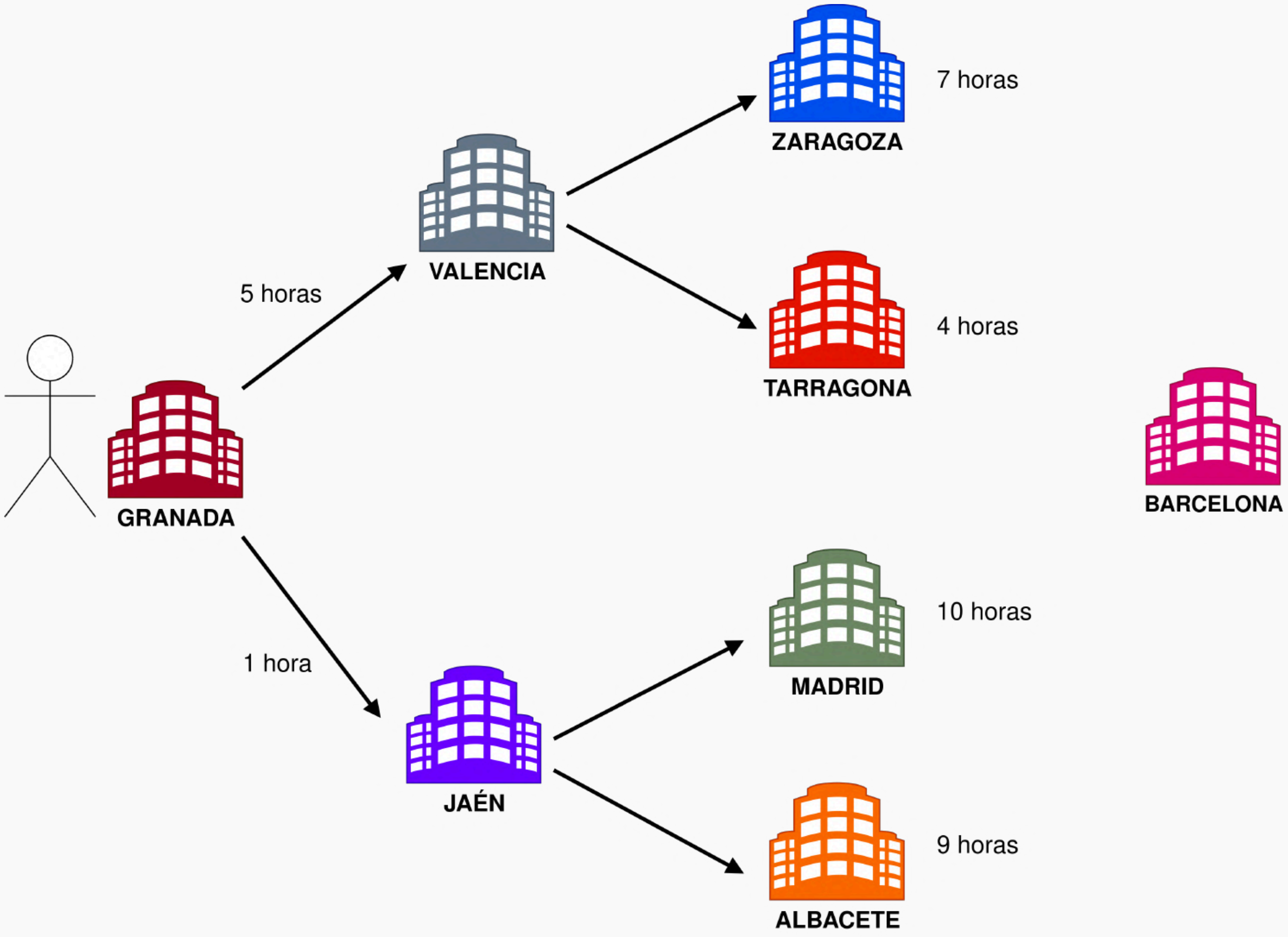
30 July 1954

Ejemplo



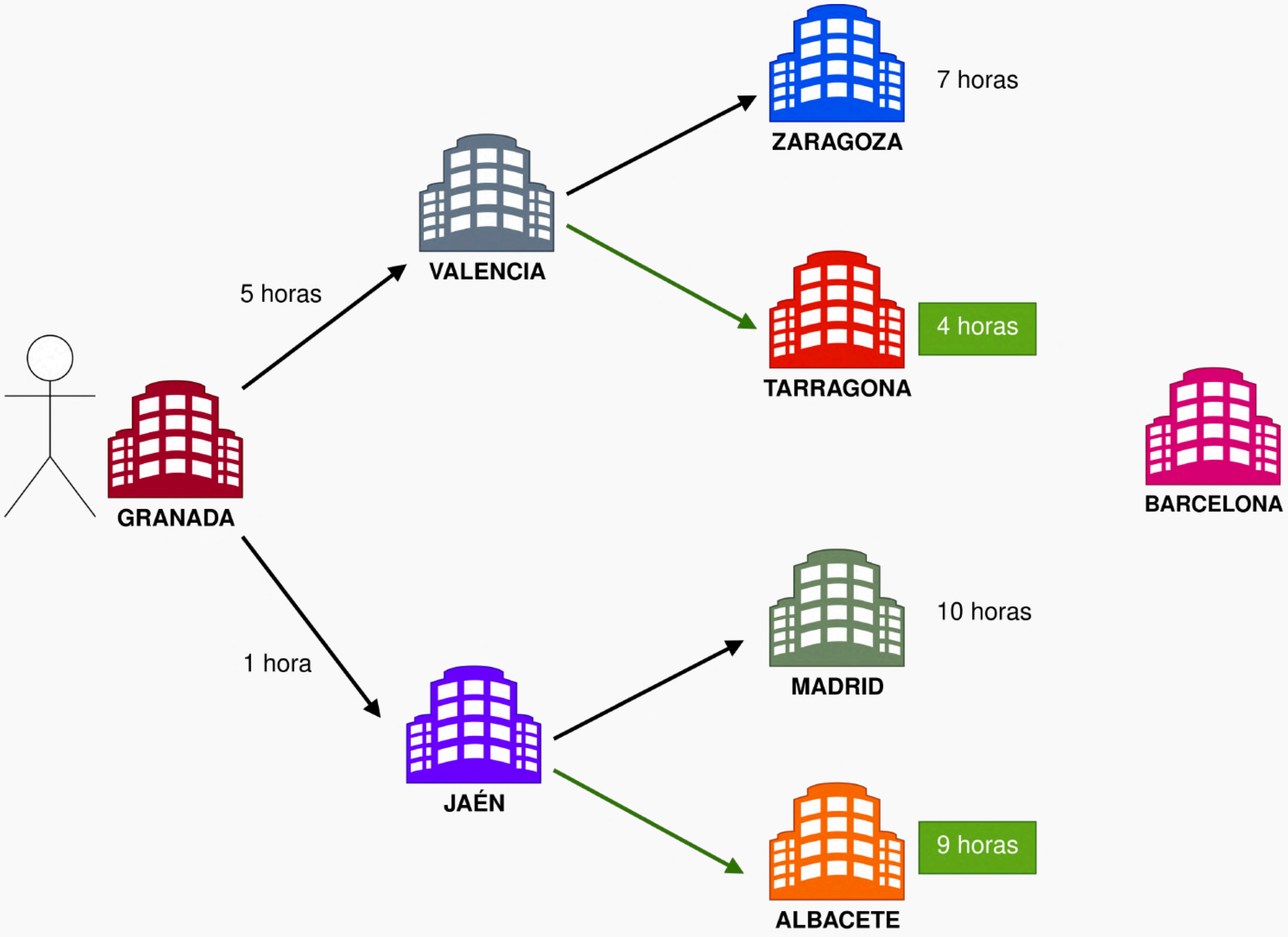
3 horas
ZARAGOZA
4 horas
VALENCIA
5 horas
3 horas
TARRAGONA
1 hora
GRANADA
BARCELONA
7 horas
1 hora
3 horas
MADRID
JAÉN
6 horas
3 horas
ALBACETE

Ejemplo



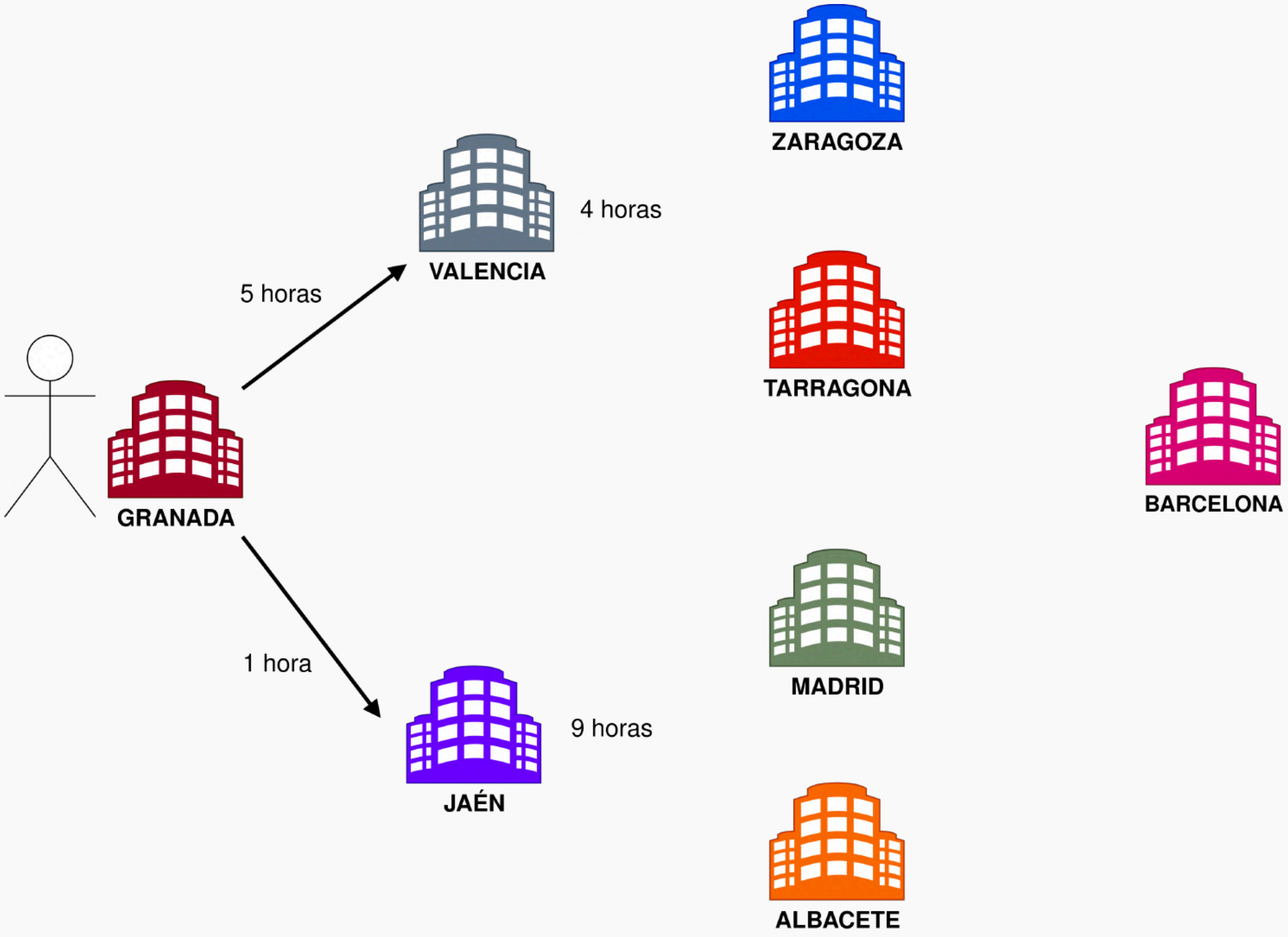
ZARAGOZA
7 horas
VALENCIA
5 horas
TARRAGONA
4 horas
GRANADA
BARCELONA
1 hora
MADRID
10 horas
JAÉN
ALBACETE
9 horas

Ejemplo



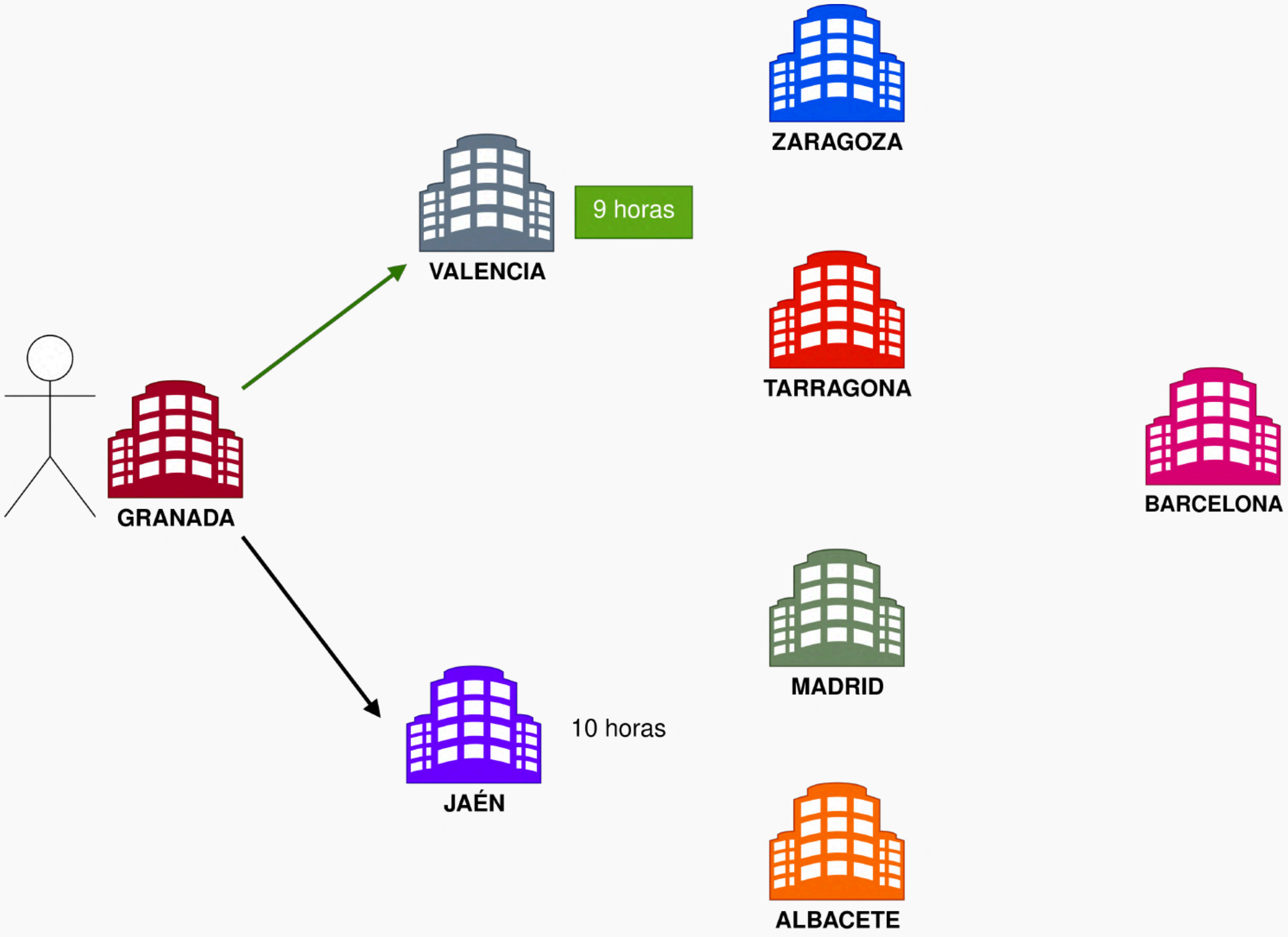
7 horas
ZARAGOZA
VALENCIA
5 horas
4 horas
TARRAGONA
GRANADA
BARCELONA
10 horas
1 hora
MADRID
JAÉN
9 horas
ALBACETE

Ejemplo



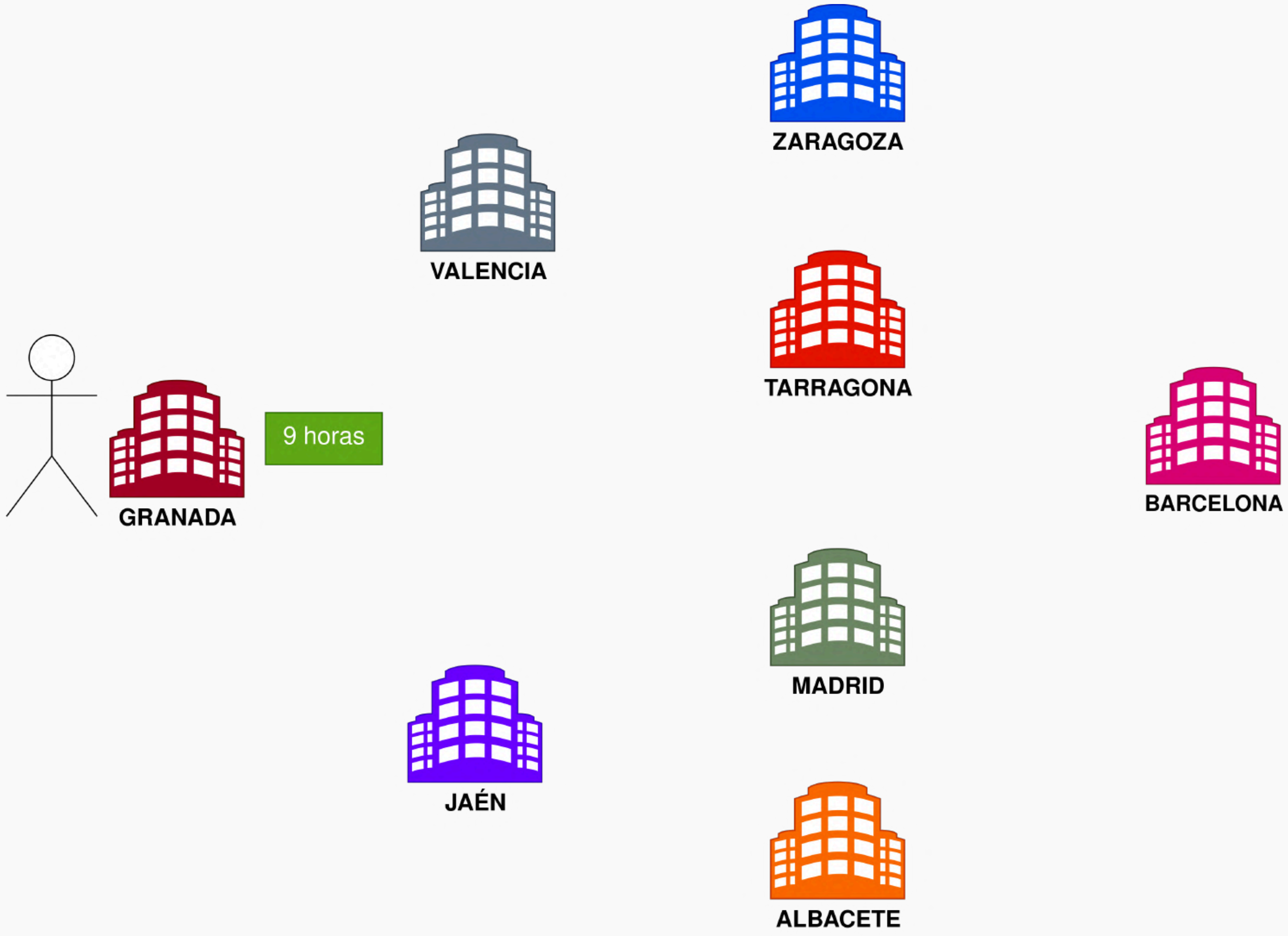
ZARAGOZA
VALENCIA
4 horas
TARRAGONA
5 horas
BARCELONA
GRANADA
MADRID
1 hora
9 horas
JAÉN
ALBACETE

Ejemplo

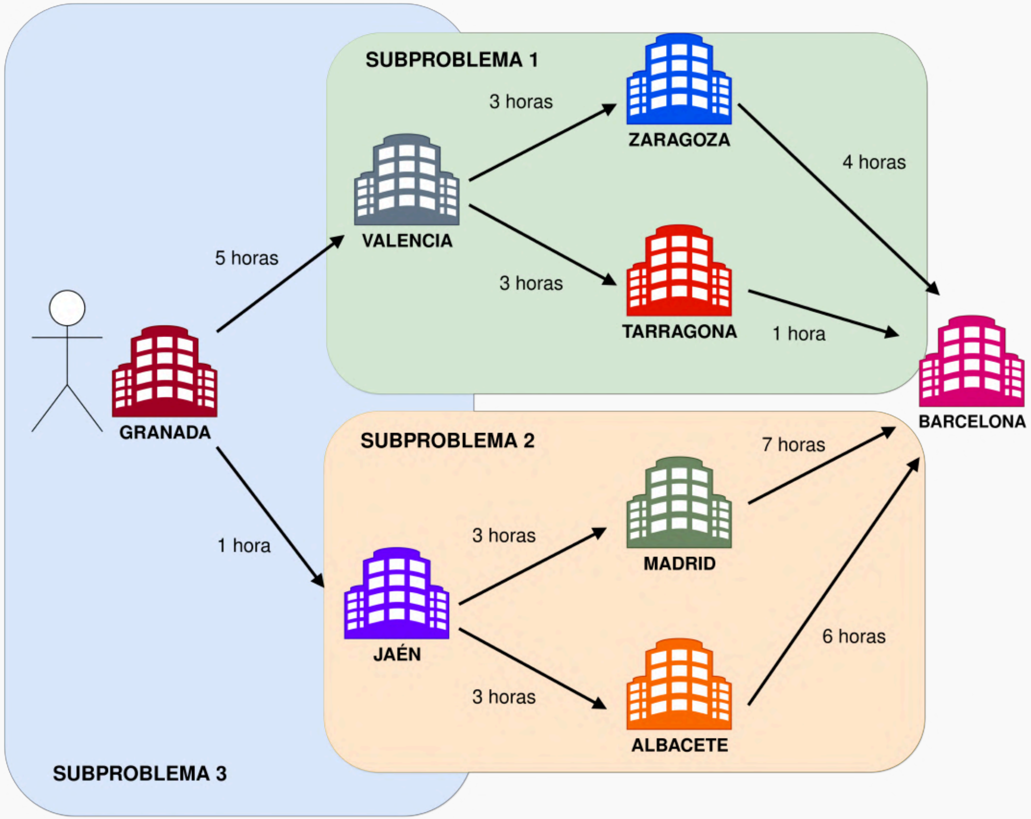


ZARAGOZA
VALENCIA
9 horas
TARRAGONA
GRANADA
BARCELONA
MADRID
JAÉN
10 horas
ALBACETE

Ejemplo


ZARAGOZA
VALENCIA
TARRAGONA
9 horas
GRANADA
BARCELONA
MADRID
JAÉN
ALBACETE

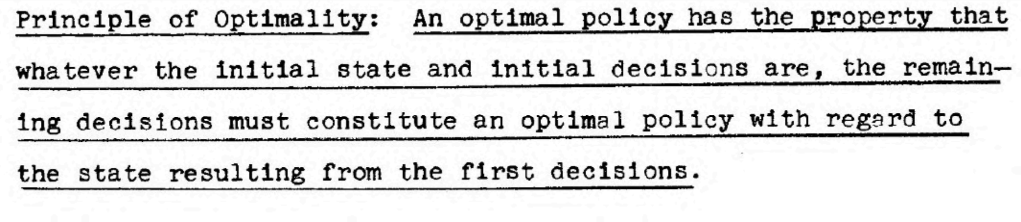
Ejemplo



Estamos descomponiendo un problema complejo en varios subproblemas más sencillos.

La solución óptima general es igual a la composición de las soluciones óptimas a los diferentes subproblemas.

Si en cada decisión elegimos la acción óptima, el resultado final será el óptimo.


Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions.

Programación dinámica

La programación dinámica permite el cálculo de políticas óptimas a partir de un MDP.

👎 Son algoritmos poco eficientes en la práctica, especialmente cuando el número de estados y/o acciones es elevado.

👍 No obstante, ofrecen una importante base teórica.

Más adelante estudiaremos otros algoritmos que tratan de imitar a la programación dinámica con menor coste computacional y sin asumir un modelo perfecto del entorno.

Programación dinámica

Emplearemos las funciones de valor (v, q) para emprender la búsqueda de políticas óptimas. Recordemos las ecuaciones de optimalidad de Bellman:

$$
\begin{aligned}
v^*(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v^*\left(S_{t+1}\right) \mid S_t = S, A_t = a\right] \\
&= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v^*(s')]
\end{aligned}
$$

$$
\begin{aligned}
q^*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q^*\left(S_{t+1}\right) \mid S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s', r | s, a)\left[r + \gamma \max_{a'} q^*(s', a')\right]
\end{aligned}
$$

Predicción vs. control

Dividiremos los algoritmos a estudiar en dos categorías:

Predicción vs. control

Dividiremos los algoritmos a estudiar en dos categorías:

Predicción

Obtener la función de valor (v, q) a partir de una política dada.

- ***Evaluación de la política***.

Predicción vs. control

Dividiremos los algoritmos a estudiar en dos categorías:

Predicción

Obtener la función de valor (v, q) a partir de una política dada.

- *Evaluación de la política*.

Control

Encontrar una política que maximice la recompensa acumulada.

- *Mejora de la política.*

Predicción vs. control

Dividiremos los algoritmos a estudiar en dos categorías:

Predicción

Obtener la función de valor (v, q) a partir de una política dada.

- ***Evaluación de la política***.

Control

Encontrar una política que maximice la recompensa acumulada.

- ***Mejora de la política.***

☑ El **control** es el principal objetivo en RL.

Iteración de la política

Iteración de la política

Iteración de la política (*policy iteration*) es un método empleado para aproximar progresivamente la política óptima en un MDP.

- Consiste en la alternancia **evaluación** y **mejora** de la política:

Iteración de la política

Iteración de la política (*policy iteration*) es un método empleado para aproximar progresivamente la política óptima en un MDP.

- Consiste en la alternancia **evaluación** y **mejora** de la política:

Evaluación de la política (*policy evaluation*). Obtención de la función de valor v_π asociada a la política actual $\longrightarrow$ **predicción**.

Iteración de la política

Iteración de la política (*policy iteration*) es un método empleado para aproximar progresivamente la política óptima en un MDP.

- Consiste en la alternancia **evaluación** y **mejora** de la política:

Evaluación de la política (*policy evaluation*). Obtención de la función de valor v_π asociada a la política actual $\longrightarrow$ **predicción.**

Mejora de la política (*policy improvement*). Obtención de la política *greedy* correspondiente a v_π $\longrightarrow$ **control.**

Evaluación de la política

Policy evaluation

Evaluación de la política – predicción

Objetivo: calcular la función estado-valor v_π a partir de una política arbitraria π

- Recordemos que calcular v_π para cada estado puede ser computacionalmente costoso (incluso inviable).
- Es por esto por lo que empleamos **métodos iterativos** que nos permitan aproximar estos valores:

$$v_0 \longrightarrow v_1 \longrightarrow v_2 \longrightarrow \dots \longrightarrow v_\pi$$

Evaluación de la política – predicción

¿CÓMO EVALUAR UNA POLÍTICA?

1. Los valores iniciales (v_0) se asignan de forma arbitraria, excepto para los estados terminales, con valor = 0.
2. Consideramos una secuencia de funciones de valor aproximadas: $v_0, v_1, v_2, \ldots$, donde cada una establece un mapeo $\mathcal{S}^+ \longrightarrow \mathbb{R}$.
3. Cada aproximación sucesiva se obtiene empleando la ecuación de Bellman para v_π como una regla de actualización, esto es:

$$\begin{aligned} v_{k+1}(S_t) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_k(s')\right] \end{aligned}$$

Evaluación de la política – predicción

La actualización alcanza un **punto fijo** cuando $v_k = v_\pi$.

- De hecho, la convergencia de la secuencia de v_k en v_π se da cuando $k \longrightarrow \infty$.

Como la política se evalúa en múltiples iteraciones, es más correcto denominar a este algoritmo **EVALUACIÓN ITERATIVA DE LA POLÍTICA** (*iterative policy evaluation*).

Cada iteración en la evaluación de la política actualiza el valor de cada estado para producir una nueva función de valor aproximada v_{k+1}.

Todas las actualizaciones en programación dinámica se denominan ***esperadas*** (*expected updates*) porque están basadas en la expectación sobre todos los posibles siguientes estados (*vs.* en un posible estado aleatorio).

Evaluación síncrona *vs.* evaluación asíncrona

Computacionalmente, la implementación **síncrona** de la evaluación iterativa de la política requiere de dos vectores:

- Vector de valores originales $v_k(s)$:

$v_k(s_0)$	$v_k(s_1)$	...	$v_k(s_n)$

- Vector de valores actualizados $v_{k+1}(s)$:

$v_{k+1}(s_0)$	$v_{k+1}(s_1)$	...	$v_{k+1}(s_n)$

... ya que el cálculo de $v_k(s)$ requiere del valor de $v_{k+1}(s)$.

No obstante, podemos simplificar el algoritmo y emplear **un único vector** donde los valores se sobreescriban.

- Es lo que denominamos una versión *in-place* o **asíncrona** del algoritmo.

Evaluación iterativa de la política

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 $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

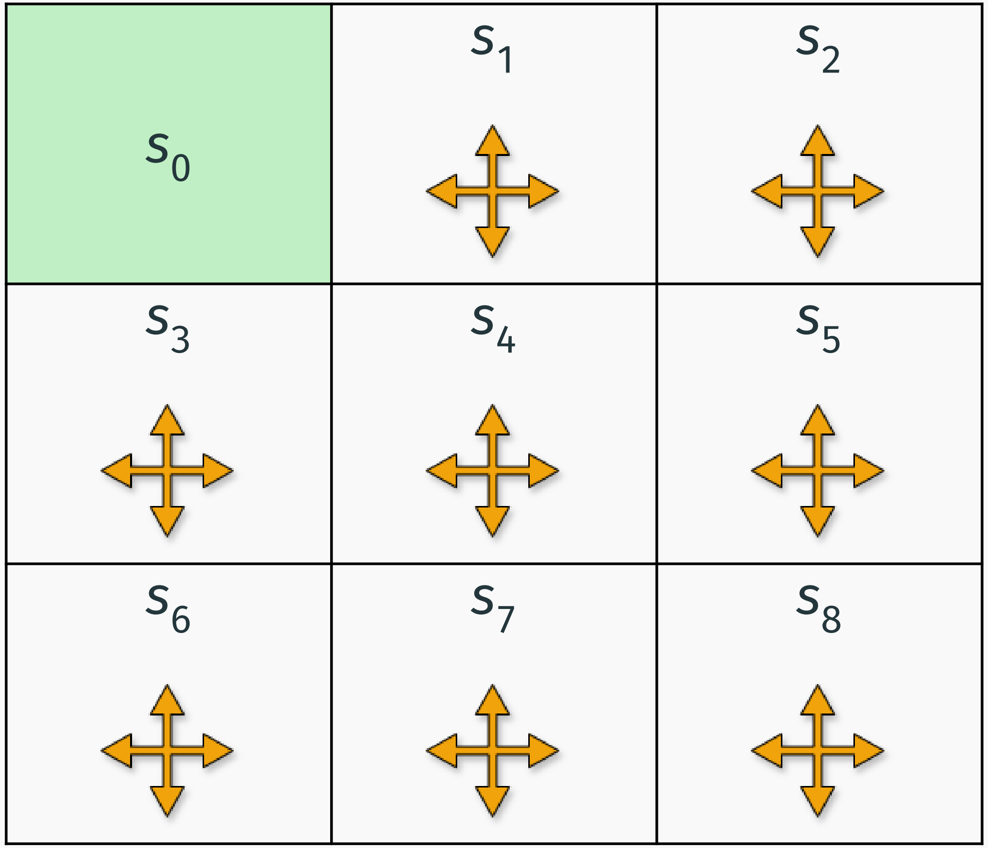
Loop:
    $\Delta \leftarrow 0$
    Loop for each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Ejemplo – *Gridworld* 3x3

s_0	s_1	s_2
s_3	s_4	s_5
s_6	s_7	s_8

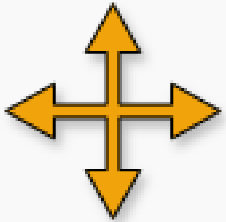
- Toda transición tiene recompensa = −1.
- s_0 es el estado final a alcanzar.
- Chocar contra una pared supone volver al mismo estado.
- Asumimos $\gamma = 1,\ \theta = 0.1$
- Evaluación **síncrona**.

Ejemplo – *Gridworld* 3x3

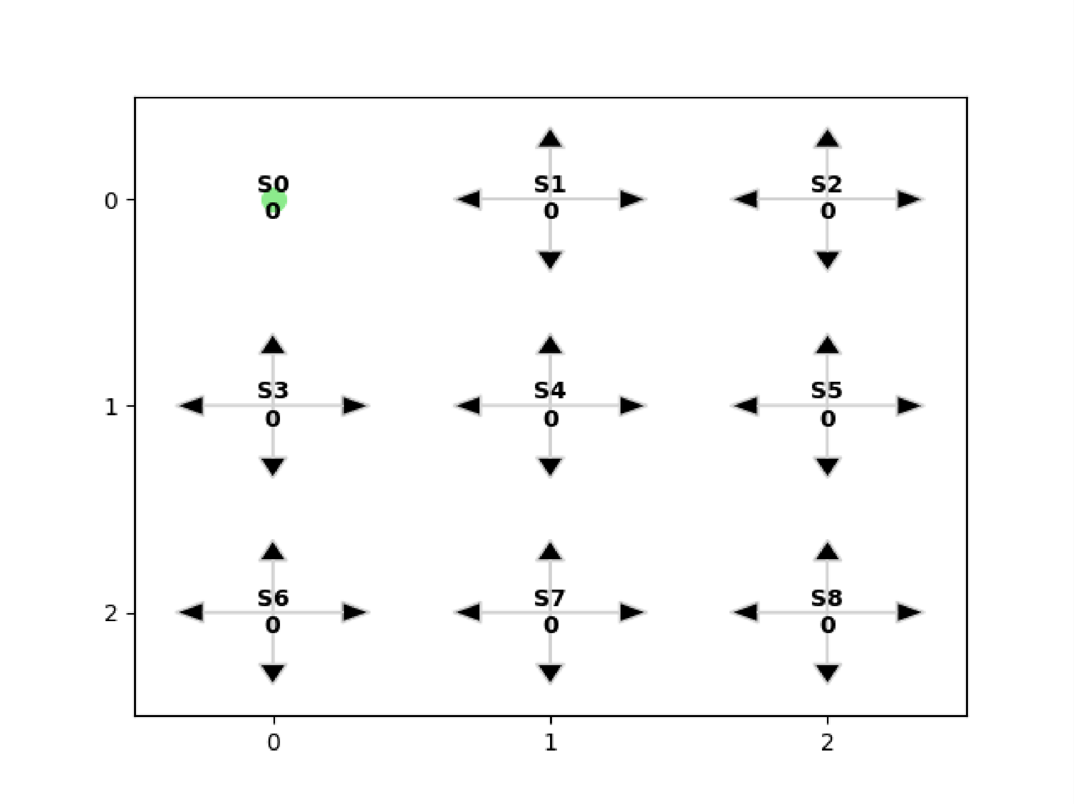


Evaluaremos una política π que asigna la **misma probabilidad** a todas las acciones:

$$\pi(a|s) = 0.25, \forall a \in \mathcal{A}(s)$$



Ejemplo – *Gridworld* 3x3



Inicialmente:

$$v_\pi^0(s_i) = 0, \quad \forall i \in \{0...8\}$$

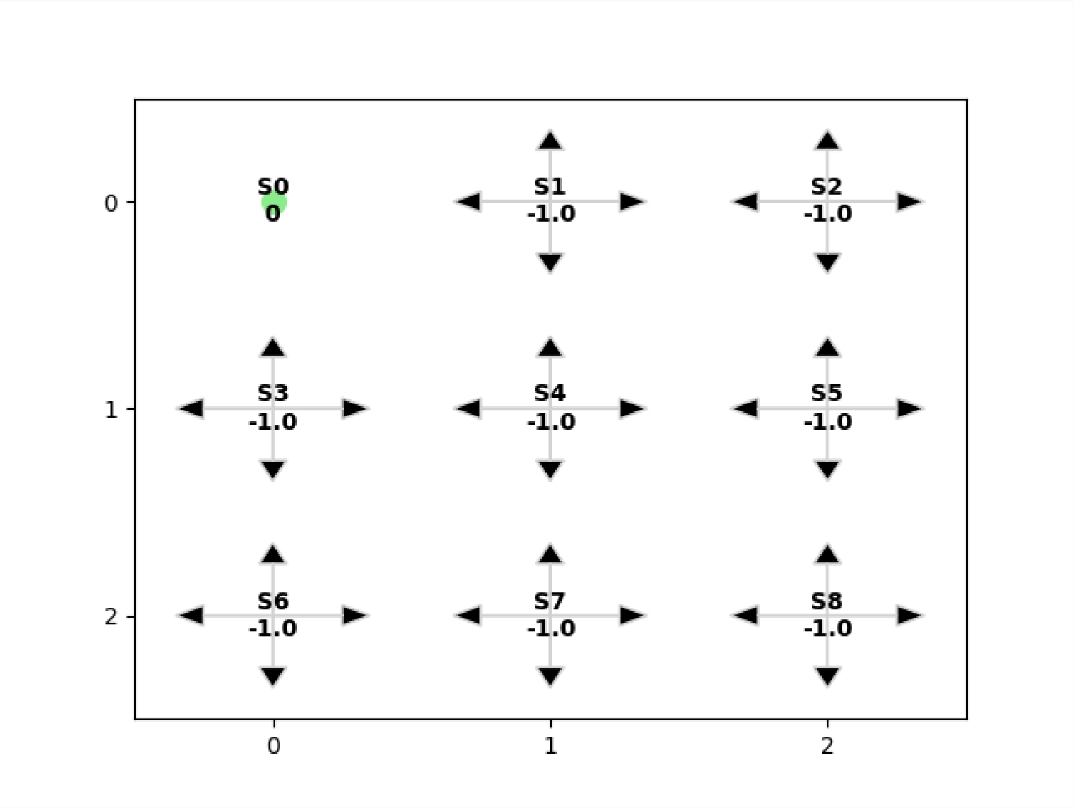
Aplicaremos iterativamente la regla de actualización:

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

hasta que: $\quad |v_{k+1} - v_k| < \theta$.

Ejemplo – *Gridworld* 3x3

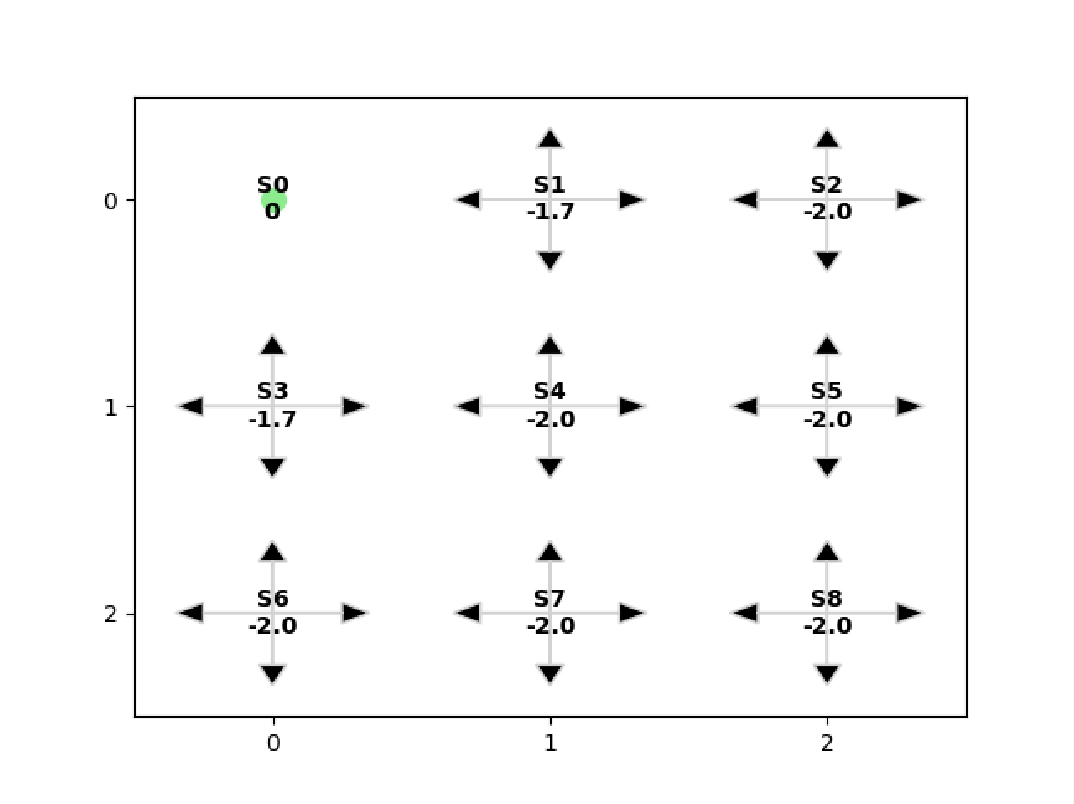
Primera iteración: $v_\pi^1(s_i)$



$$
\begin{aligned}
v_\pi^1(s_1) = 0.25\left[-1 + \gamma v_\pi^0(s_8)\right] \\
+0.25\left[-1 + \gamma v_\pi^0(s_7)\right] \\
+0.25\left[-1 + \gamma v_\pi^0(s_6)\right] \\
+0.25\left[-1 + \gamma v_\pi^0(s_4)\right] = \mathbf{-1}
\end{aligned}
$$

$$
\begin{aligned}
v_\pi^1(s_8) = 0.25\left[-1 + \gamma v_\pi^0(s_3)\right] \\
+0.25\left[-1 + \gamma v_\pi^0(s_1)\right] \\
+0.25\left[-1 + \gamma v_\pi^0(s_0)\right] \\
+0.25\left[-1 + \gamma v_\pi^0(s_0)\right] = \mathbf{-1}
\end{aligned}
$$

Segunda iteración: $v_\pi^2(s_i)$

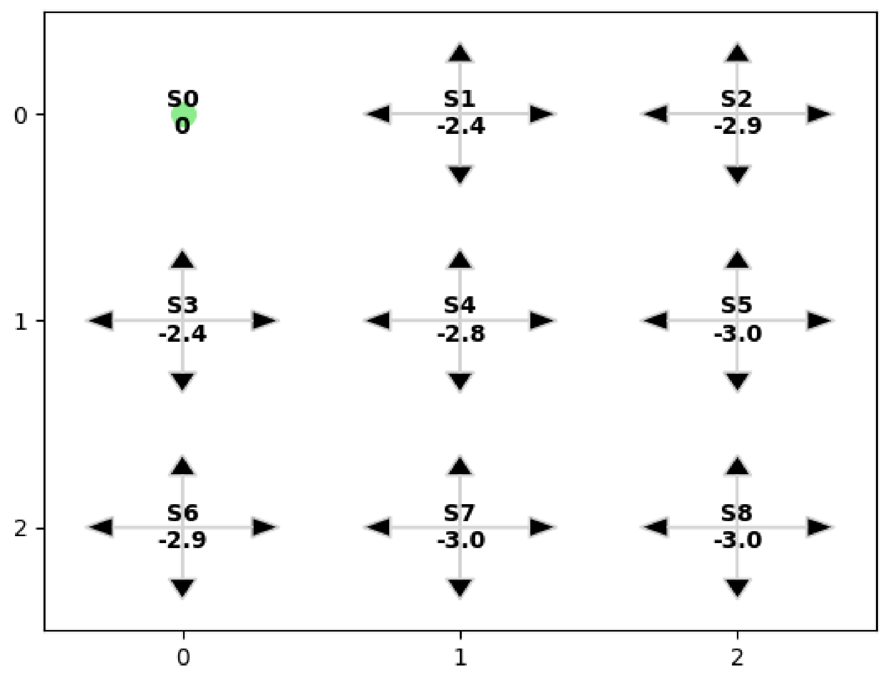


$$v_\pi^2(s_1) = 0.25\left[-1 + \gamma \underbrace{v_\pi^1(s_0)}_{0}\right]$$
$$+0.25\left[-1 + \underbrace{v_\pi^1(s_0)}_{-1}\right]$$
$$+0.25\left[-1 + \gamma v_\pi^1(s_2)\right]$$
$$+0.25\left[-1 + \gamma v_\pi^1(s_4)\right] = \mathbf{1.75}$$

$$v_\pi^2(s_4) = 0.25\left[-1 + \gamma \underbrace{v_\pi^1(s_0)}_{-1}\right]$$
$$+0.25\left[-1 + \gamma v_\pi^1(s_3)\right]$$
$$+0.25\left[-1 + \gamma v_\pi^1(s_5)\right]$$
$$+0.25\left[-1 + \gamma v_\pi^1(s_7)\right] = \mathbf{-2}$$

Ejemplo – *Gridworld* 3x3

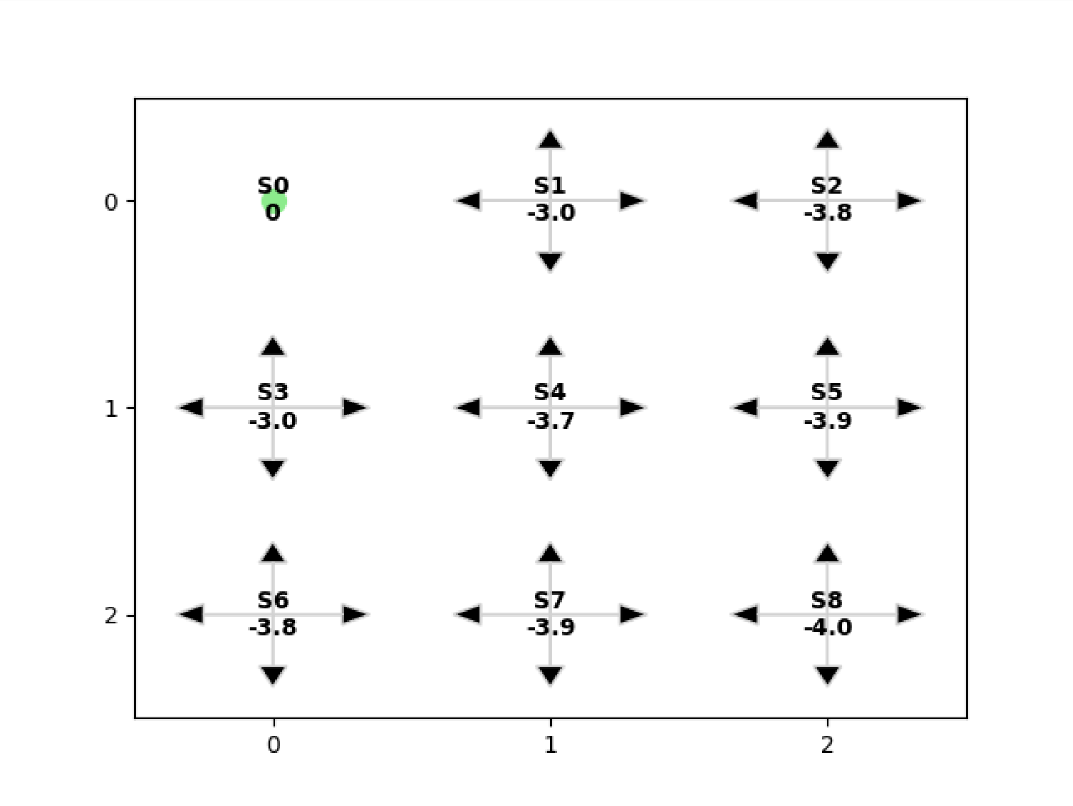
Tercera iteración: $v_\pi^3(s_i)$



Continuamos iterando... ⌛

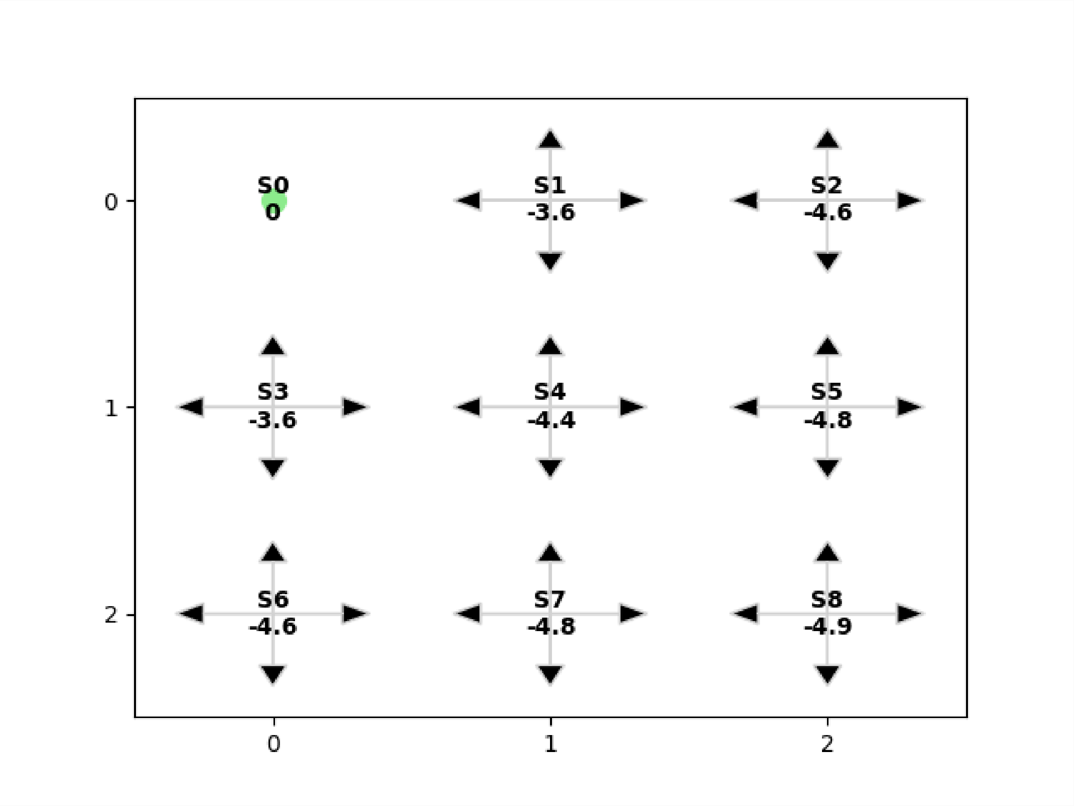
Ejemplo – *Gridworld* 3x3

Cuarta iteración: $v_\pi^4(s_i)$



Continuamos iterando... ⌛

Quinta iteración: $v_\pi^5(s_i)$

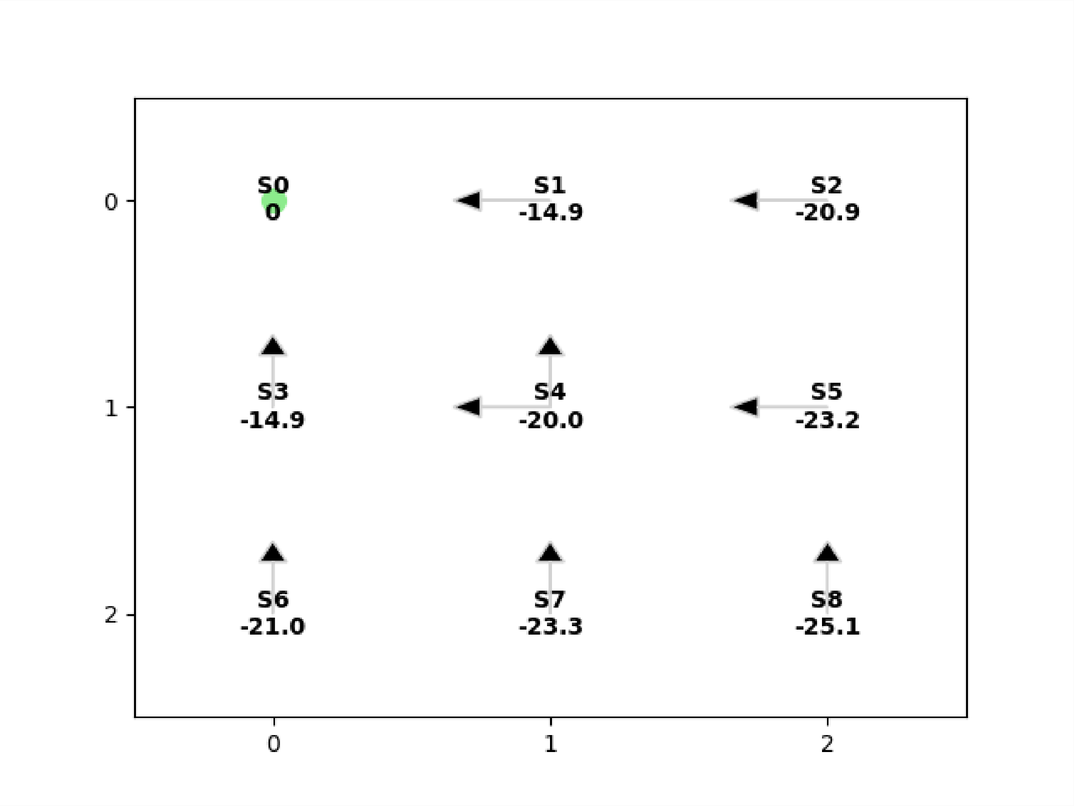


Continuamos iterando... ⏳

Ejemplo – *Gridworld* 3x3

Ejemplo – *Gridworld* 3x3

Última iteración: $v_\pi^{78}(s_i)$



Convergencia alcanzada:

$$v_\pi^{78} \qquad (\Delta < \theta)$$

Ejemplo – *Gridworld* 3x3

Última iteración: $v_\pi^{78}(s_i)$

$v(s_0) = 0$	$v(s_1) \simeq -14.9$	$v(s_2) \simeq -20.9$
$v(s_3) \simeq -14.9$	$v(s_4) \simeq -20$	$v(s_5) \simeq -23.2$
$v(s_6) \simeq -21$	$v(s_7) \simeq -23.3$	$v(s_8) \simeq -25.1$

Convergencia alcanzada:

$$v_\pi^{78} \qquad (\Delta < \theta)$$

Los valores obtenidos son una aproximación de los pasos necesarios para alcanzar s_0 desde cualquier estado *siguiendo una política aleatoria*.

Ejemplo – *Gridworld* 3x3

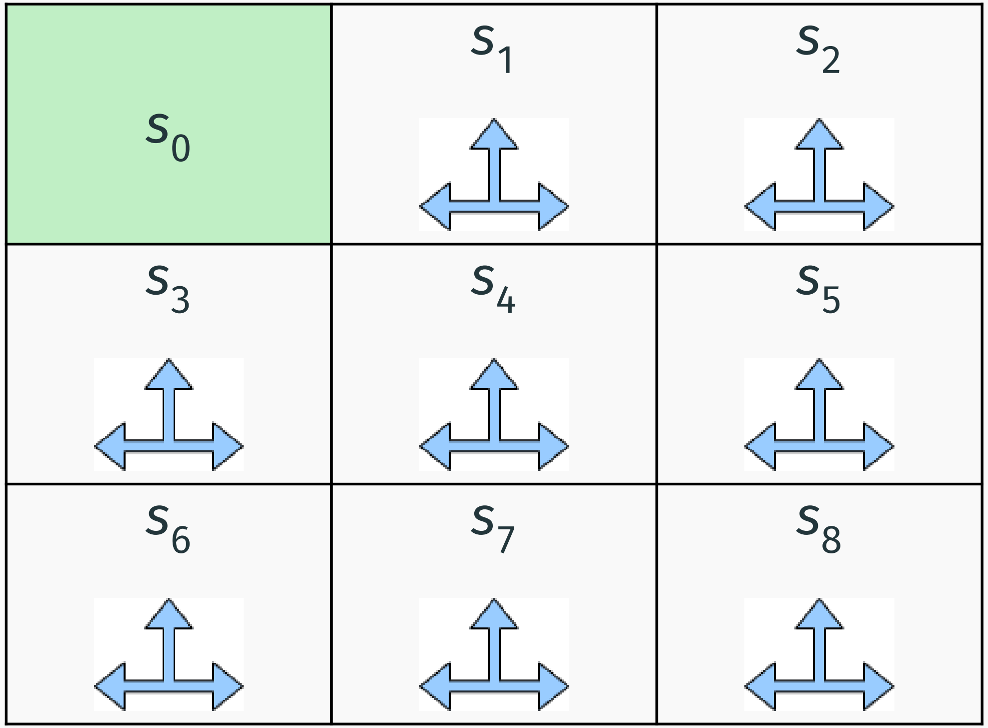
Si la actualización de valores empleada es síncrona, todos los valores tomados como referencia para obtener $v_\pi^{k+1}(s_i)$ son $v_\pi^k(s)$.

Por el contrario, la actualización asíncrona permite que los valores actualizados se puedan utilizar tan pronto como son calculados.

- Sólo se requiere un vector de valores.
- La convergencia es más rápida.

En el ejemplo anterior, se emplean **52** iteraciones para converger con el método asíncrono *vs.* las **78** iteraciones empleando actualizaciones síncronas.

Ejemplo – *Gridworld* 3x3



Evaluemos ahora una política π' que **no permite ir hacia abajo**:

$$\pi'\left(\boxed{\uparrow}\ |s\right) = 0.33$$

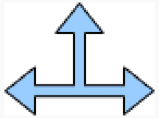
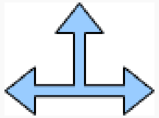
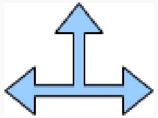
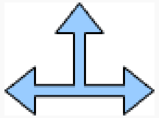
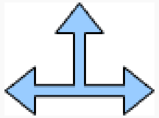
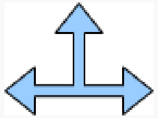
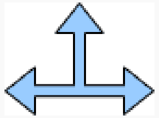
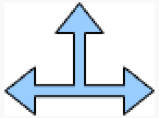
$$\pi'\left(\boxed{\leftarrow}\ |s\right) = 0.33$$

$$\pi'\left(\boxed{\rightarrow}\ |s\right) = 0.33$$

$$\pi'\left(\boxed{\downarrow}\ |s\right) = 0$$

Ejemplo – *Gridworld* 3x3

Convergencia: $v_{\pi'}^{27}(s_i)$

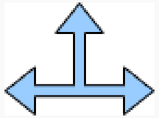
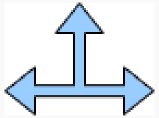
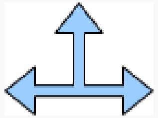
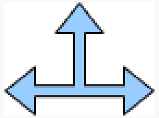
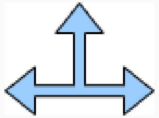
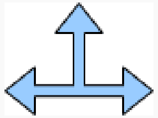
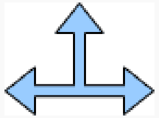
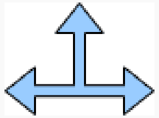
$v(s_0) = 0$	$v(s_1) \simeq -5.5$ 	$v(s_2) \simeq -8.2$ 
$v(s_3) \simeq -5.2$ 	$v(s_4) \simeq -7.6$ 	$v(s_5) \simeq -9.3$ 
$v(s_6) \simeq -9.1$ 	$v(s_7) \simeq -10.2$ 	$v(s_8) \simeq -11.2$ 

Se converge en un menor número de iteraciones (27 < 78).

¿Por qué?

Ejemplo – *Gridworld* 3x3

Convergencia: $v_{\pi'}^{27}(s_i)$

$v(s_0) = 0$	$v(s_1) \simeq -5.5$ 	$v(s_2) \simeq -8.2$ 
$v(s_3) \simeq -5.2$ 	$v(s_4) \simeq -7.6$ 	$v(s_5) \simeq -9.3$ 
$v(s_6) \simeq -9.1$ 	$v(s_7) \simeq -10.2$ 	$v(s_8) \simeq -11.2$ 

Se converge en un menor número de iteraciones (27 < 78).

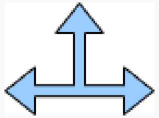
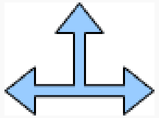
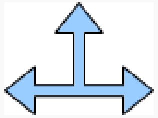
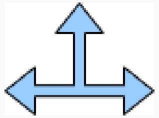
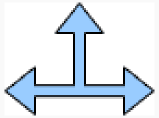
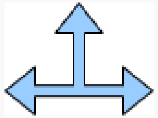
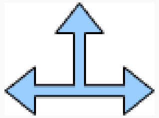
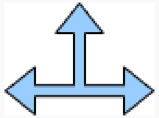
¿Por qué?

- Se reducen las acciones que alejan al agente del estado final.
- El número de acciones que influyen sobre el valor de los estados es menor.

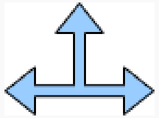
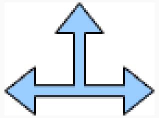
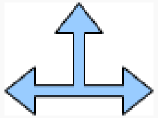
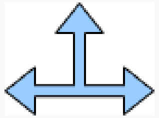
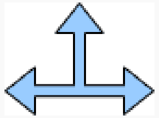
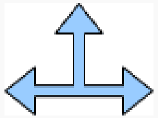
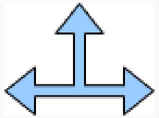
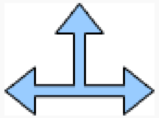
Ejemplo – *Gridworld* 3x3

Convergencia: $v_{\pi'}^{27}(s_i)$

¿Es mejor política que π?

$v(s_0) = 0$	$v(s_1) \simeq -5.5$ 	$v(s_2) \simeq -8.2$ 
$v(s_3) \simeq -5.2$ 	$v(s_4) \simeq -7.6$ 	$v(s_5) \simeq -9.3$ 
$v(s_6) \simeq -9.1$ 	$v(s_7) \simeq -10.2$ 	$v(s_8) \simeq -11.2$ 

Convergencia: $v_{\pi'}^{27}(s_i)$

$v(s_0) = 0$	$v(s_1) \simeq -5.5$ 	$v(s_2) \simeq -8.2$ 
$v(s_3) \simeq -5.2$ 	$v(s_4) \simeq -7.6$ 	$v(s_5) \simeq -9.3$ 
$v(s_6) \simeq -9.1$ 	$v(s_7) \simeq -10.2$ 	$v(s_8) \simeq -11.2$ 

¿Es mejor política que π?

Sí, ya que:

$$v_{\pi'}(s) \geq v_{\pi}(s), \quad \forall s \in \mathcal{S}$$

El número de pasos necesarios para alcanzar s_0 siguiendo π' desde cualquier estado es menor o igual a los necesarios siguiendo π.

- Hay menos acciones que, con cierta probabilidad, alejen al agente de s_0 provocándole ir hacia abajo.

📈 Mejora de la política

Policy improvement

Mejora de la política – control

Objetivo: mejorar una política π a partir de su funcion de valor v_π

- Problema de **control** $\longrightarrow$ mejora de una política dada.
- El objetivo perseguido calculando v_π para una política π es buscar cómo mejorarla.

Podemos mejorar π actuando de forma voraz (*greedy*) con respecto a los valores previamente calculados mediante evaluación iterativa de la política.

Mejora de la política – control

La actualización de la política π a π' se hace de la siguiente forma:

$$\begin{aligned}
\pi'(s) &= \operatorname{argmax}_a q_\pi(s,a) \\
&= \operatorname{argmax}_a \mathbb{E}\big[R_{t+1} + \gamma v_\pi\big(S_{t+1}\big) \mid S_t = s, A_t = a\big] \\
&= \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_\pi(s')\big]
\end{aligned}$$

Mejora de la política – control

La actualización de la política π a π' se hace de la siguiente forma:

$$\pi'(s) = \underbrace{\text{argmax}_a\, q_\pi(s,a)}_{\text{Política que maximiza } q_\pi(s,a)}$$

$$= \text{argmax}_a\, \mathbb{E}\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) \mid S_t = s, A_t = a\right]$$

$$= \text{argmax}_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma \underbrace{v_\pi(s')}_{\substack{\text{Calculado mediante} \\ \text{policy evaluation}}}\right]$$

Mejora de la política – control

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

- π' es una nueva política que elige las acciones asociadas a mayores recompensas esperadas de acuerdo con v_π.
- De acuerdo con el teorema de mejora de la política, π' *será siempre mejor o igual que* π.

Mejora de la política – control

Al proceso de obtención de una política mejorada a partir de una política anterior lo denominamos **mejora de la política** (*policy improvement*).

Si la política que tratamos de mejorar ya es óptima, entonces se cumplirá que:

$$V_{\pi} = V_{\pi'} = V^*$$

Esto se cumple tanto para políticas **deterministas** como para **estocásticas**.

En resumen...

1. Evaluamos la política actual π, aproximando iterativamente su función de valor v_π:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_k(s')\big]$$

2. Utilizamos v_π para obtener π', tal que:

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

1. ***Policy evaluation*** $(\pi, v_\pi^{78} \simeq v_\pi)$

$v(s_0) = 0$	$v(s_1) \simeq -14.9$	$v(s_2) \simeq -20.9$
$v(s_3) \simeq -14.9$	$v(s_4) \simeq -20$	$v(s_5) \simeq -23.2$
$v(s_6) \simeq -21$	$v(s_7) \simeq -23.3$	$v(s_8) \simeq -25.1$

2. ***Policy improvement*** (π')

s_8	s_7 ←	s_6 ←
s_5 ↑	s_4 ← ↑	s_3 ←
s_2 ↑	s_1 ↑	s_0 ↑

Hay varias políticas óptimas

Iteración de la política

Policy iteration

Hasta el momento, dada una política inicial π, hemos obtenido su función de valor v_π y la hemos mejorado tal que:

$$\pi_0 \underbrace{\xrightarrow{E}}_{\text{evaluation}} v_{\pi_0} \underbrace{\xrightarrow{I}}_{\text{improvement}} \pi_1$$

Iteración de la política

Hasta el momento, dada una política inicial π, hemos obtenido su función de valor v_π y la hemos mejorado tal que:

$$\underbrace{\pi_0 \underbrace{\xrightarrow{E}}_{\text{evaluation}} v_{\pi_0} \underbrace{\xrightarrow{I}}_{\text{improvement}} \pi_1}_{\text{Iteración de la política}}$$

Denominamos **iteración de la política** (*policy iteration*) a la aplicación de una evaluación iterativa de la política seguida de una mejora de la política.

Iteración de la política

Si repetimos este proceso iterativamente, obtenemos una secuencia de políticas que mejoran de forma monotónica:

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1$$

Si repetimos este proceso iterativamente, obtenemos una secuencia de políticas que mejoran de forma monotónica:

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \longrightarrow \dots$$

Iteración de la política

Si repetimos este proceso iterativamente, obtenemos una secuencia de políticas que mejoran de forma monotónica:

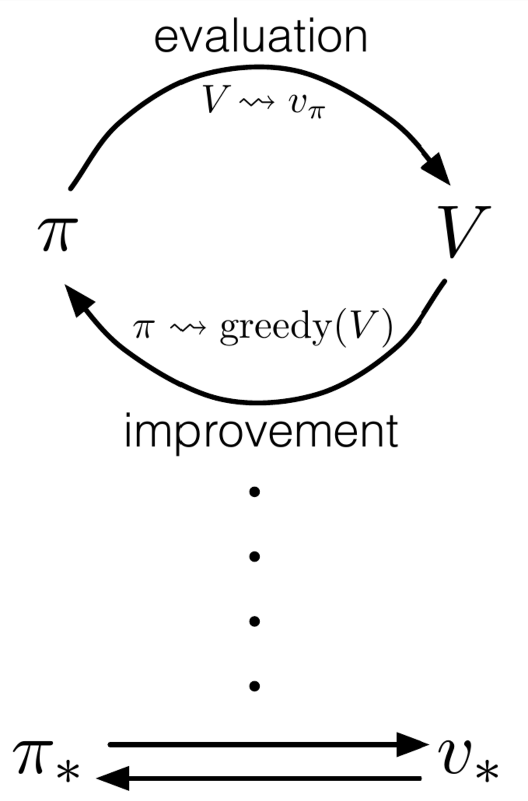
$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \longrightarrow \dots \xrightarrow{I} \pi^* \xrightarrow{E} v_{\pi^*}$$

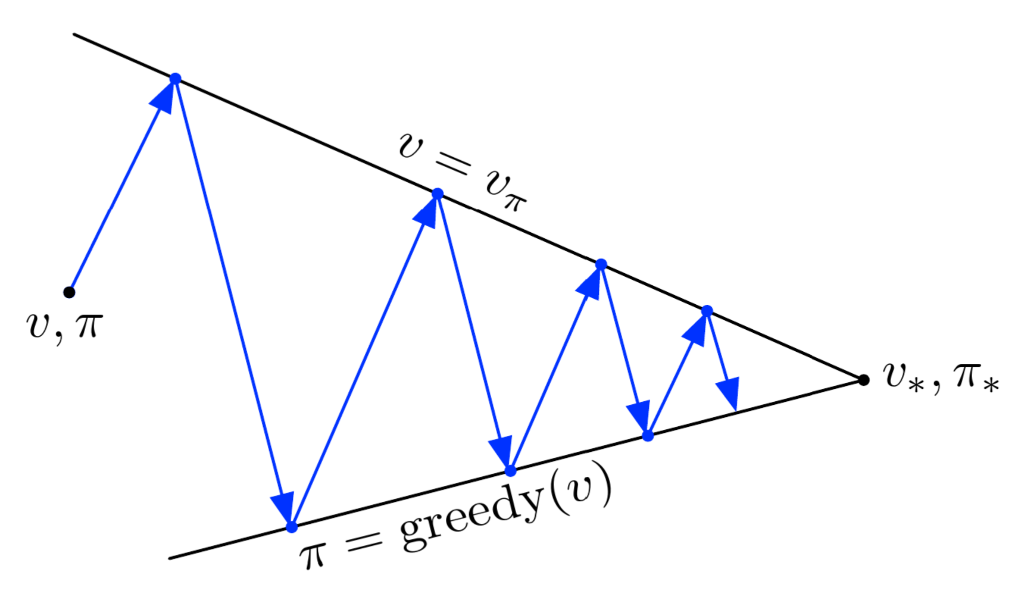
Finalmente se converge en la **política óptima**.

🔄 Iteración de la política

🔄 = (∞ × 📊 + 📈) × n_{iter}

Iteración de la política





$\pi, p, \gamma \longrightarrow$ POLICY EVALUATION $\longrightarrow v_\pi$

$\pi, \gamma \longrightarrow$ POLICY IMPROVEMENT $\longrightarrow \pi^*$

Iteración de la política

POLICY ITERATION

POLICY EVALUATION $(\pi_0, v_{\pi_1}^{78} \simeq v_{\pi_0})$

$v(s_0) = 0$	$v(s_1) \simeq -14.9$	$v(s_2) \simeq -20.9$
$v(s_3) \simeq -14.9$	$v(s_4) \simeq -20$	$v(s_5) \simeq -23.2$
$v(s_6) \simeq -21$	$v(s_7) \simeq -23.3$	$v(s_8) \simeq -25.1$

POLICY IMPROVEMENT (π_1)

s_8	s_7 ⇦	s_6 ⇦
s_5 ⇧	s_4 ⇦ ⇧	s_3 ⇦
s_2 ⇧	s_1 ⇧	s_0 ⇧

Iteración de la política

POLICY ITERATION

POLICY EVALUATION $(\pi_1, v_{\pi_1}^5 \simeq v_{\pi_1})$

$v(s_0) = 0$	$v(s_1) \simeq -1$	$v(s_2) \simeq -2$
$v(s_3) \simeq -1$	$v(s_4) \simeq -2$	$v(s_5) \simeq -3$
$v(s_6) \simeq -2$	$v(s_7) \simeq -3$	$v(s_8) \simeq -4$

POLICY IMPROVEMENT $(\pi_2 \simeq \pi^*)$

s_8	s_7 ⇦	s_6 ⇦
s_5 ⇧	s_4 ⇦ ⇧	s_3 ⇦ ⇧
s_2 ⇧	s_1 ⇦ ⇧	s_0 ⇦ ⇧

Son necesarias **2** iteraciones de la política para converger en π^*.

Problemas más complejos requerirán un mayor número de iteraciones.

Iteración de la política

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 $s \in \mathcal{S}$

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

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

Hablando en Python…

Evaluación (síncrona) de la política en Python

```
def sync_policy_eval(states, pi, theta):
  while (True):
      delta = 0
      states_old = copy.deepcopy(states)
      for state in states:
          if not is_terminal(state):
              v_old = state.value
              v_new = 0
              for action in pi[state]:
                  action_prob = pi[state][action]
                  if action_prob > 0:
                      next_state, reward = get_transition(
                          state, action, states_old)
                      v_new += action_prob * \
                          (reward + GAMMA * next_state.value)
              state.value = v_new
              delta = max(delta, abs(v_old - v_new))
      if (delta < theta):
          break
```

Evaluación (asíncrona) de la política en Python

```python
def async_policy_eval(states, pi, theta):
    while (True):
        delta = 0
        for state in states:
            if not is_terminal(state):
                v_old = state.value
                v_new = 0
                for action in pi[state]:
                    action_prob = pi[state][action]
                    if action_prob > 0:
                        next_state, reward = get_transition(
                            state, action, states)
                        v_new += action_prob *
                          (reward + GAMMA * next_state.value)
                state.value = v_new
                delta = max(delta, abs(v_old - v_new))
        if (delta < theta):
            break
```

Mejora de la política en Python

```
def policy_improvement(states, pi):
  policy_stable = True
  for state in states:
      if not is_terminal(state):
          old_actions = [action for action,
                         prob in pi[state].items() if prob > 0]
          action_values = {}
          for action in pi[state]:
              action_prob = pi[state][action]
              if action_prob > 0:
                  next_state, reward = get_transition(
                      state, action, states)
                  action_values[action] = reward + GAMMA * next_state.value
          best_actions = [action for action, value in action_values.items(
          ) if value == max(action_values.values())]
          for action in ACTIONS:
              if action in best_actions:
                  pi[state][action] = 1 / len(best_actions)
              else:
                  pi[state][action] = 0
          if old_actions != best_actions:
              policy_stable = False
  return policy_stable
```

Iteración de la política en Python

```python
# states
states = []
for i in range(3):
    for j in range(3):
        states.append(State(i, j))

# policy
pi = {}
for state in states:
    pi[state] = {}
    for action in ACTIONS:
        pi[state][action] = 1/len(ACTIONS)

def policy_iteration(states, pi):
    policy_stable = False
    while not policy_stable:
        sync_policy_eval(states, pi)
        policy_stable = policy_improvement(states, pi)
```

Iteración de valor

Limitaciones de iteración de la política

El algoritmo de iteración de la política presenta un **problema**:

> **Cada iteración supone evaluar la política actual hasta alcanzar la convergencia,** siendo necesarias múltiples iteraciones.

Limitaciones de iteración de la política

El algoritmo de iteración de la política presenta un **problema**:

> **Cada iteración supone evaluar la política actual hasta alcanzar la convergencia,** siendo necesarias múltiples iteraciones.

⚠ **Dicha evaluación puede ser un proceso demasiado lento...**

$$\pi_0 \xrightarrow{\boldsymbol{E}} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{\boldsymbol{E}} V_{\pi_1} \xrightarrow{I} \pi_2 \longrightarrow \dots$$

Limitaciones de iteración de la política

La *evaluación iterativa de la política* puede truncarse sin pérdida de información.

💡 No es necesario esperar a que los valores converjan.

$$v_{\pi_1}, v_{\pi_2}, v_{\pi_3} \ldots, \underbrace{v_{\pi_{k-2}}, v_{\pi_{k-1}}}, v_{\pi_k} = v_{\pi}$$

Las funciones de valor aproximadas $v_{\pi_{k-2}}$ y $v_{\pi_{k-1}}$ pueden no haber convergido pero ser **estimaciones suficientes** para mejorar π.

- Es decir, $v_{\pi_{k-2}}$, $v_{\pi_{k-1}}$ y v_{π_k} pueden conducir a la misma política π'.

Limitaciones de iteración de la política

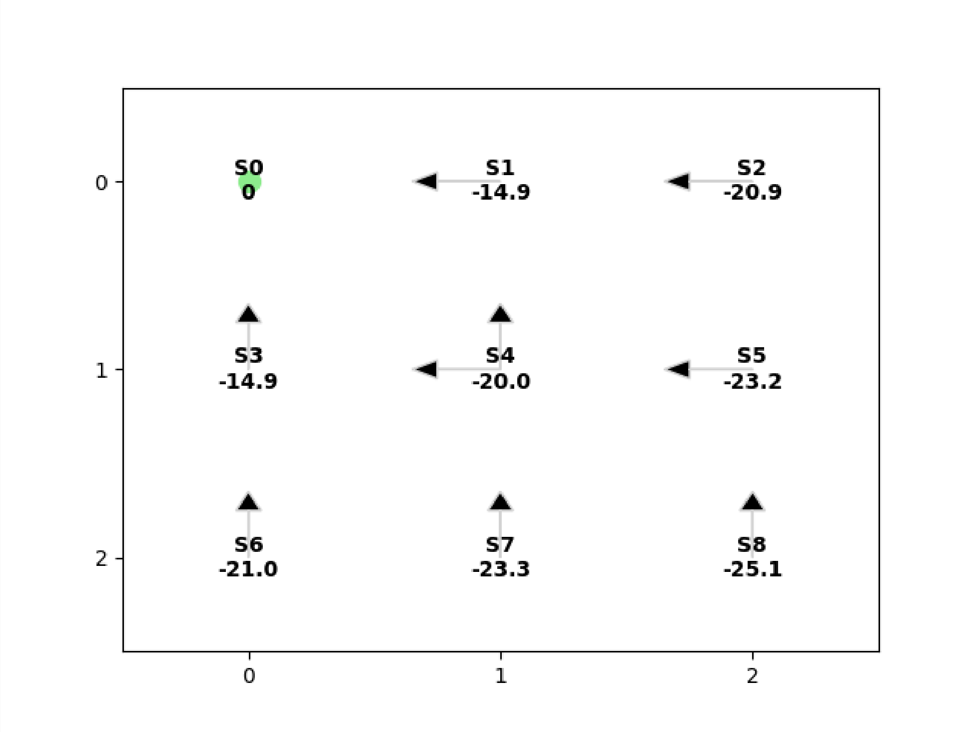
En el ejemplo visto, la política obtenida mediante $\pi = \text{greedy}(V)$ **no varía** desde la iteración 4 hasta la 78. Es decir:

$$\pi = \text{greedy}\left(v_\pi^{78}\right) = \text{greedy}\left(v_\pi^{4}\right)$$

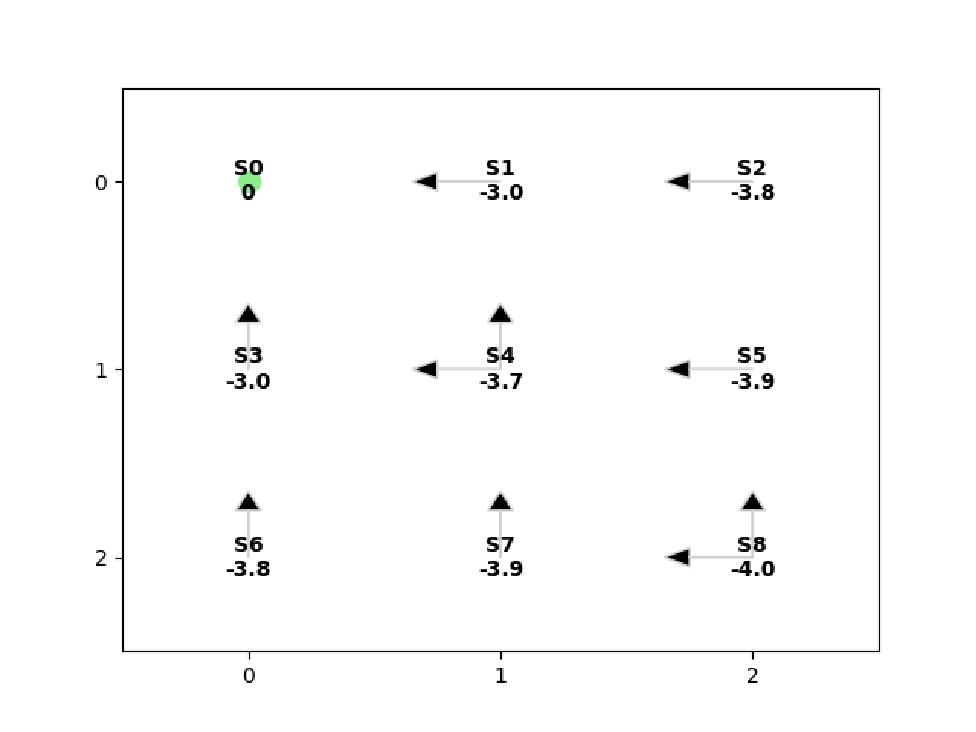
Aunque las funciones de valor son diferentes, la política a la que conducen es la misma. Por tanto, no es necesario esperar a la convergencia de los valores para mejorar la política actual.

Limitaciones de iteración de la política

$\pi = \text{greedy}\left(v_\pi^{78}\right)$



$\pi = \text{greedy}\left(v_\pi^{4}\right)$



Iteración de valor

Value iteration

Iteración de valor

La **iteración de valor** (*value iteration*) es un caso extremo en el que solamente realizamos una iteración de *policy evaluation*, y seguidamente mejoramos nuestra política.

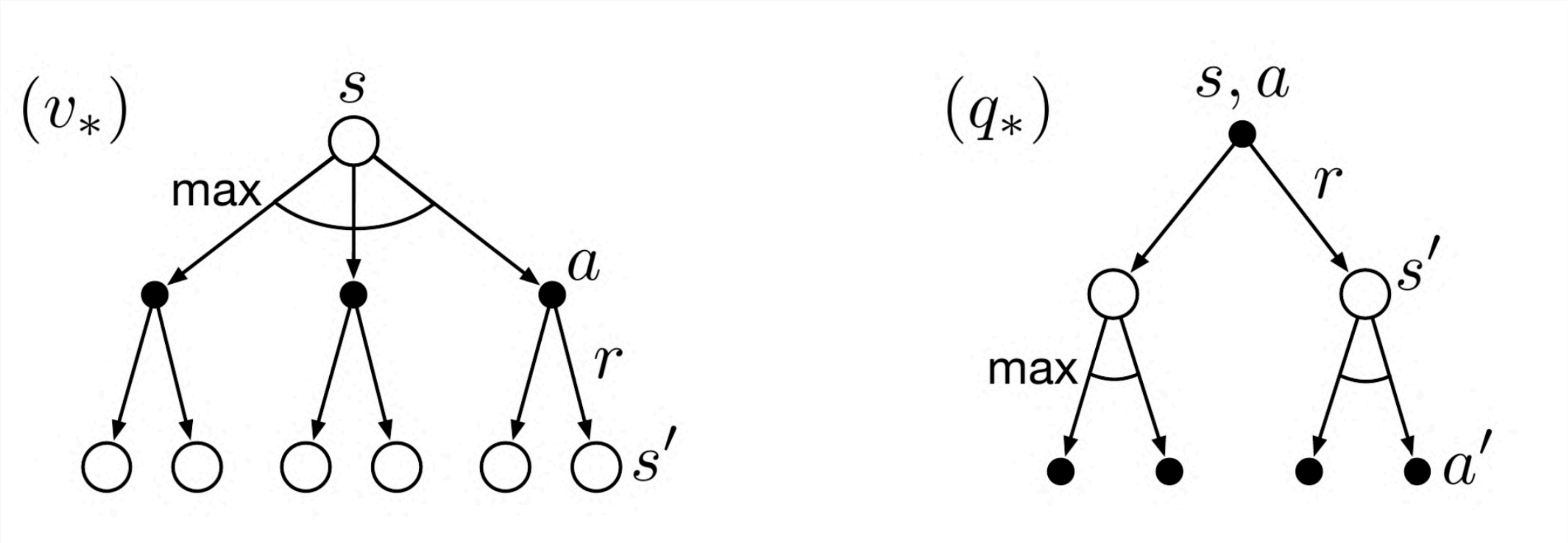
- Es decir, el valor de cada estado se evalúa/actualiza una única vez.

La iteración de valor equivale a convertir la **ecuación de optimalidad de Bellman** en una regla de actualización:

$$v_{k+1} = \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_k(s')\right]$$

- La convergencia está teóricamente garantizada.
- En la práctica, el algoritmo de *value iteration* converge cuando la diferencia etre v_k y v_{k+1} es lo suficientemente pequeña.

Recordemos los diagramas correspondientes a las ecuaciones de optimalidad de Bellman...



Iteración de valor

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 $V(\mathit{terminal}) = 0$

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

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

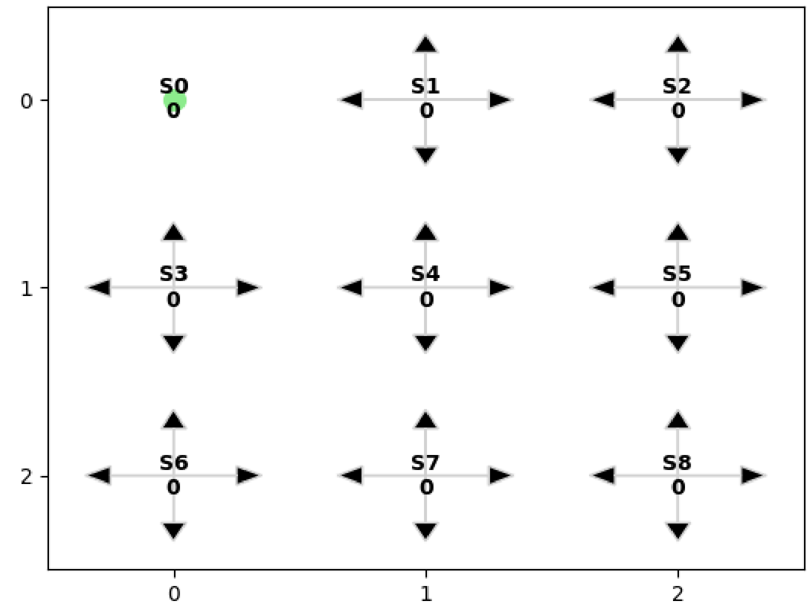
 $=\big($  $+$  $\big) \times n_{\text{iter}}$

Ejemplo

Veamos cómo funciona *value iteration* en el ejemplo visto.

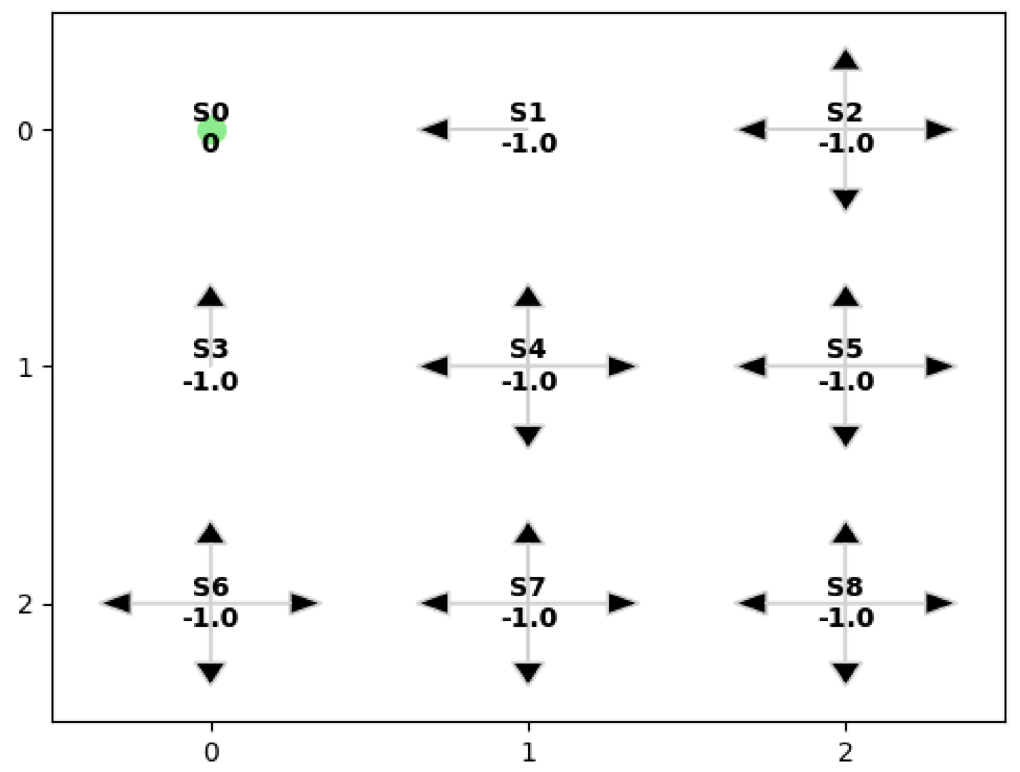
- En este caso consideramos $\gamma = 0.9$

Aunque no tendrá efectos notables la política final obtenida.



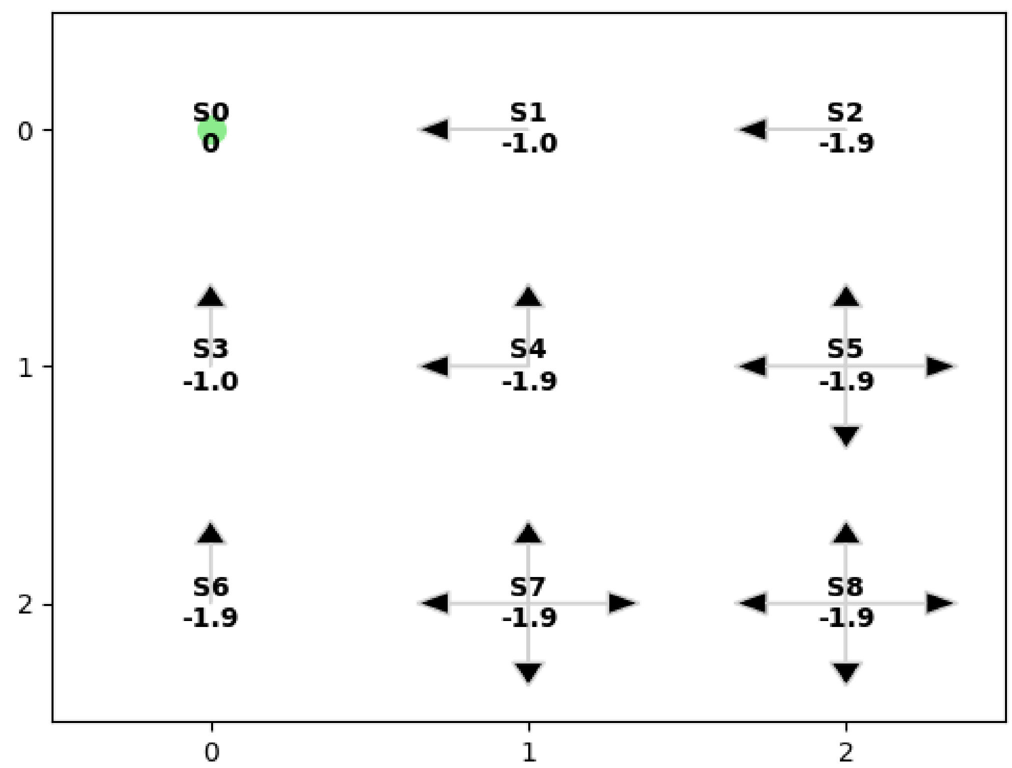
Ejemplo

Evaluación de π_0 **Mejora**: $\pi_1 = \text{greedy}\left(v_{\pi_0}^1\right)$



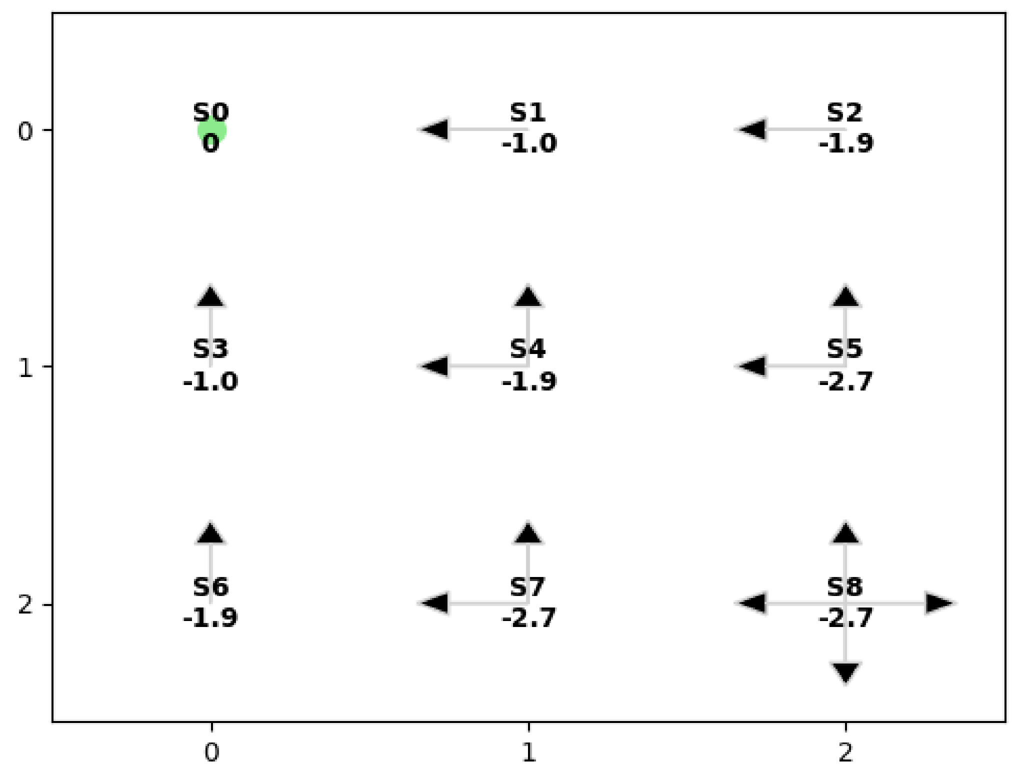
Ejemplo

Evaluación de π_1 **Mejora**: $\pi_2 = \text{greedy}\left(v^1_{\pi_1}\right)$



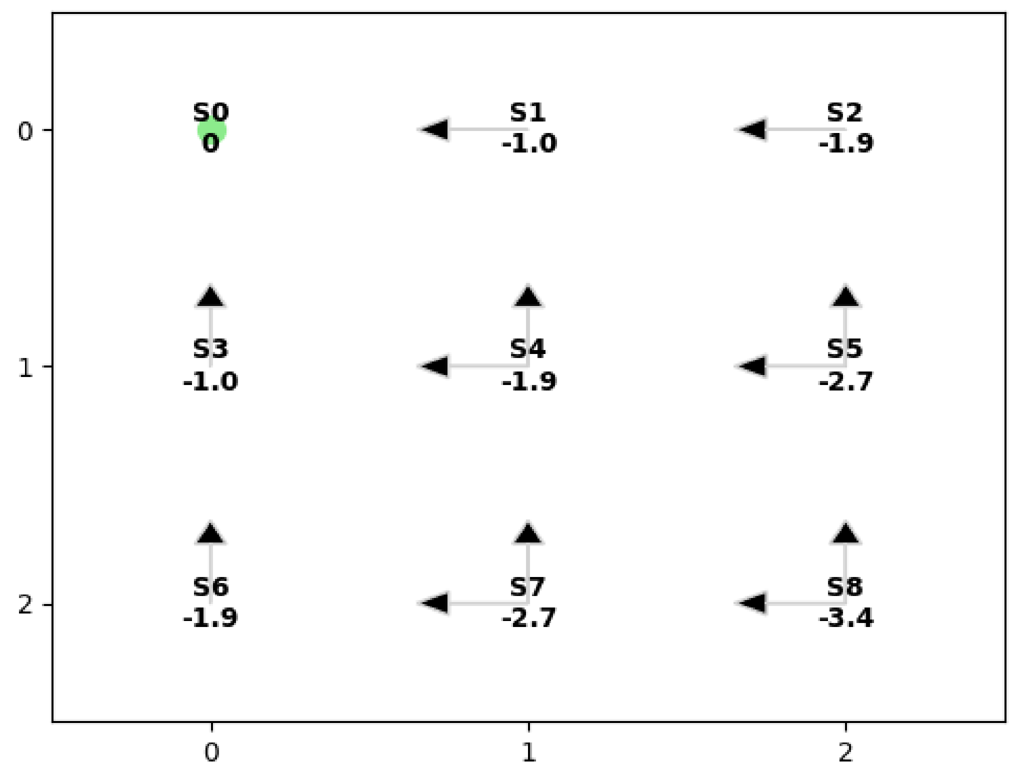
Ejemplo

Evaluación de π_2 **Mejora**: $\pi_3 = \text{greedy}\left(v^1_{\pi_2}\right)$



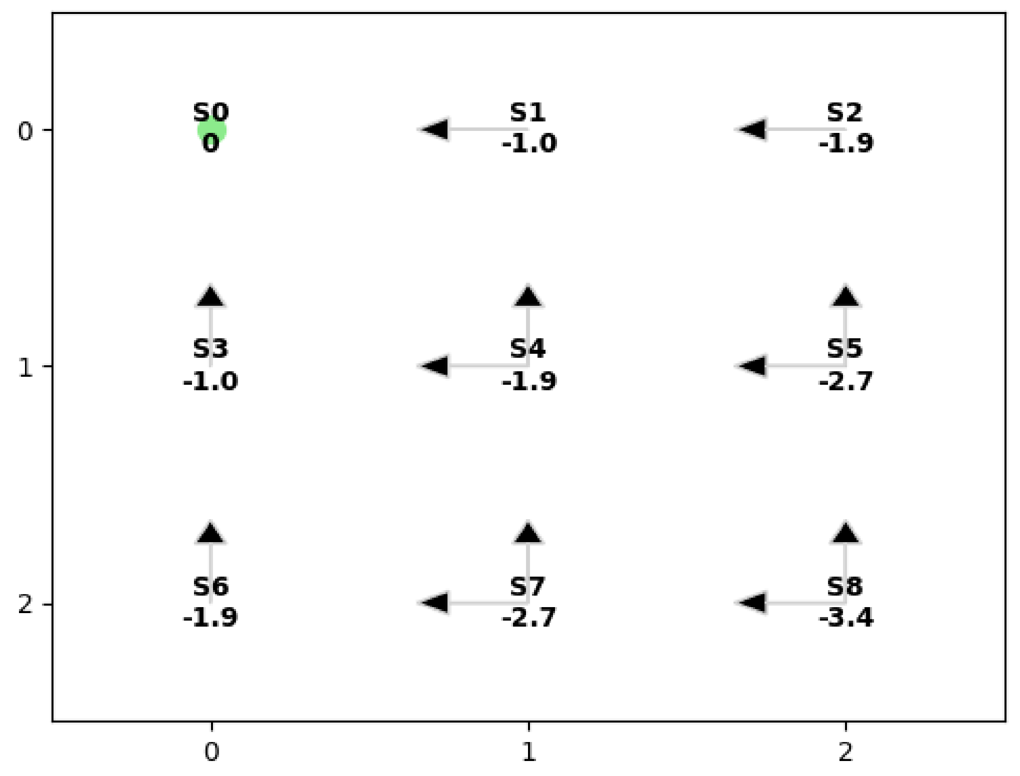
Ejemplo

Evaluación de π_3 **Mejora**: $\pi_4 = \text{greedy}\left(v^1_{\pi_3}\right)$



Ejemplo

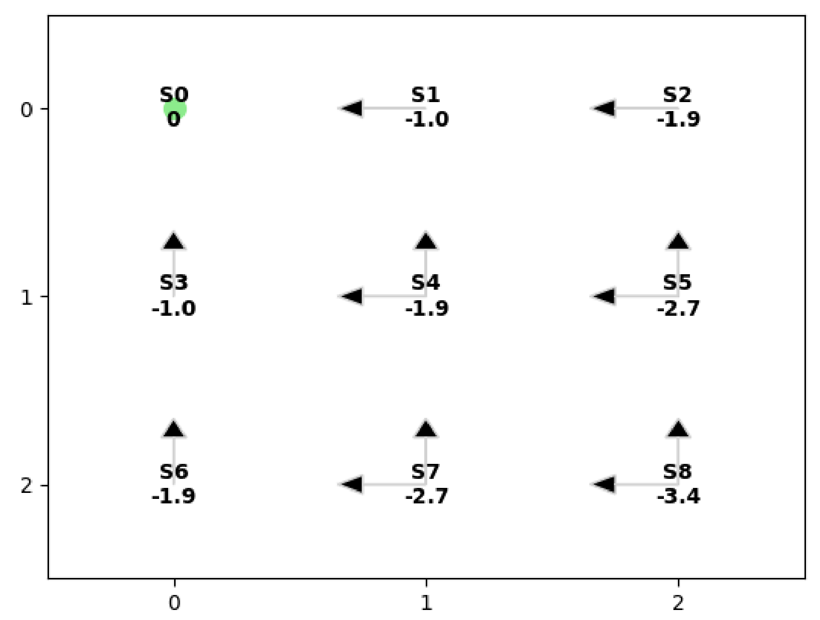
Evaluación de π_4 **Mejora**: $\pi_5 = \text{greedy}\left(v^1_{\pi_4}\right) = \pi_4$ **CONVERGENCIA** ☑



Ejemplo

☑ En 5 *iteraciones de valor* hemos obtenido la **política óptima**.

- Son más iteraciones totales que en el caso de *policy iteration* (5 > 2), pero son mucho más cortas.
- La elección de un método u otro dependerá del problema, aunque generalmente *value iteration* es más rápido.



¿Qué método elegir?

Comparativa

Iteración de la política	Iteración de valor
Parte de una política aleatoria	Parte de una función de valor aleatoria
Algoritmo más complejo. Implica: (1) evaluación hasta convergencia (2) mejora de la política en varias iteraciones	Algoritmo más simple: un sólo paso incluye evaluación y mejora
Requiere pocas iteraciones para converger	Requiere más iteraciones para converger, aunque generalmente es más rápido

Programación dinámica asíncrona

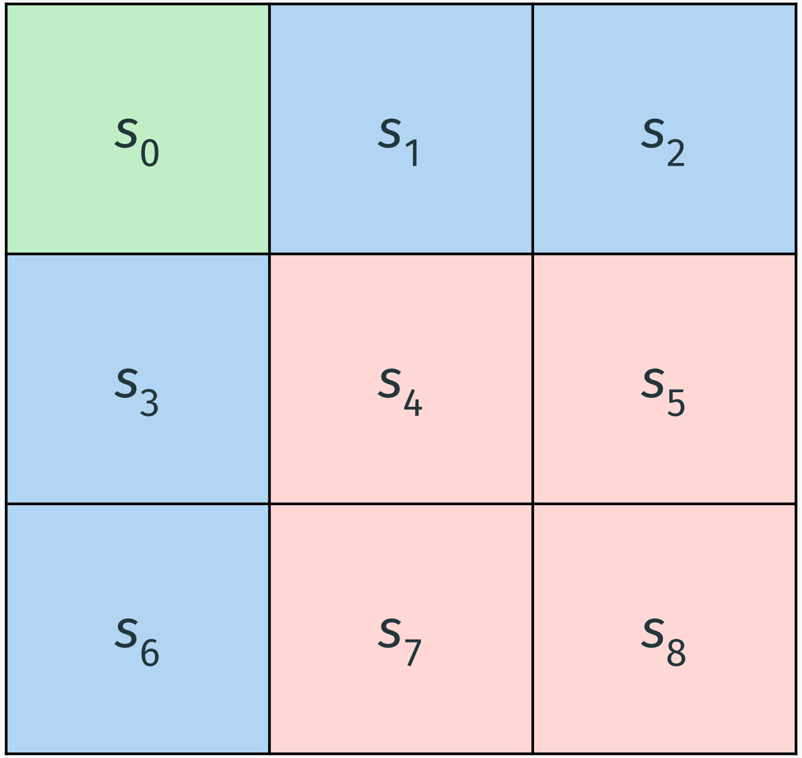
Programación dinámica asíncrona

Los algoritmos estudiados pueden converger en una política óptima sin necesidad de actualizar todos los estados en cada iteración.

Esto puede ser útil para reducir el coste computacional por iteración en entornos con un gran número de estados.

La **programación dinámica asíncrona** supone actualizar valores de forma irregualr.

La aplicación de métodos asíncronos dependerá de la naturaleza del problema abordado.



EJEMPLOS DE APLICACIÓN

- Alternancia en la actualización de estados en posiciones pares/impares.
- Actualizar más frecuentemente los estados cercanos a estados finales.
- Actualizar con menor frecuencia aquellos estados con más transiciones posibles (mayor coste computacional).
- *Etc*.

Iteración de la política generalizada

Iteración de la política generalizada

Hemos visto que la 🔄 **Iteración de la política** consiste en la alternancia entre:

- 📊 **Evaluación iterativa de la política**: hacer consistente la función de valor con la política actual.
- 📈 **Mejora de la política – control**: construir una política *greedy* con respecto a la función de valor actual.

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \longrightarrow \dots \xrightarrow{I} \pi^* \xrightarrow{E} v_{\pi^*}$$

$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \longrightarrow \dots \xrightarrow{I} \pi^* \xrightarrow{E} q_{\pi^*}$$

Iteración de la política generalizada

Podemos generalizar el proceso de evaluación + mejora empleando el término iteración de la política generalizada (*generalized policy iteration*, o **GPI**).

$$\pi \leftrightarrows V$$

Iteración de la política generalizada

Conjunto de algoritmos basados en la alternancia entre *policy evaluation* y *policy improvement* para obtener una política óptima.

- Independientemente de la granularidad y detalles de implementación.

Iteración de la política generalizada

- ↩️ **Iteración de valor** sólo utiliza una iteración de *policy evaluation* entre cada *policy improvement*: ↩️ $=$ (📊 + 📈) $\times n_{\text{iter}}$
- 🔄 **Iteración de la política** utiliza los valores de convergencia de *policy evaluation* antes de cada *policy improvement*: 🔄 $=$ ($\infty \times$ 📊 + 📈) $\times n_{\text{iter}}$
- **Cualquier método intermedio entre iteración de valor e iteración de la política** utiliza un número arbitrario K de evaluaciones antes de cada mejora de la política:

❓ $=$ ($K \times$ 📊 + 📈) $\times n_{\text{iter}}$

- También tenemos que considerar los **métodos asíncronos** que siguen una actualización de valores irregular.

Iteración de la política generalizada

¡ TODOS ESTÁN DENTRO DE GPI !

- ↩️ **Iteración de valor** sólo utiliza una iteración de *policy evaluation* entre cada *policy improvement*: ↩️ = (📊 + 📈) × n_{iter}
- 🔄 **Iteración de la política** utiliza los valores de convergencia de *policy evaluation* antes de cada *policy improvement*: 🔄 = (∞ × 📊 + 📈) × n_{iter}
- **Cualquier método intermedio entre iteración de valor e iteración de la política** utiliza un número arbitrario K de evaluaciones antes de cada mejora de la política:

❔ = (K × 📊 + 📈) × n_{iter}

- También tenemos que considerar los **métodos asíncronos** que siguen una actualización de valores irregular.

Iteración de la política generalizada

De hecho, la mayoría de métodos de RL se definen en el marco de GPI:

1. Política y función de valor identificadas.
2. La política se mejora a partir de la función de valor.
3. La función de valor se actualiza empleando la política.
4. La función de valor se estabiliza cuando es consistente con la política actual.
5. La política se estabilizac uando es *greedy* con respecto a la función de valor actual.

Se trata de un proceso donde política y función de valor cooperan y compiten al mismo tiempo hasta obtener π^* y v_{π^*}

Eficiencia en programación dinámica

Eficiencia de la programación dinámica

Los algoritmos de programación dinámica son métodos bastante **eficientes** para resolver MDPs en comparación con otras alternativas.

🕐 El tiempo de ejecución es **polinomial** con respecto al número de estados y acciones (en el peor de los casos).

- Mejor que otras alternativas, como **algoritmos de búsqueda** en el espacio de políticas.
- La **programación lineal** es otra alternativa que en la mayoría de situaciones *NO* supera a la programación dinámica.

Eficiencia de la programación dinámica

Los algoritmos de programación dinámica son métodos bastante **eficientes** para resolver MDPs en comparación con otras alternativas.

⏱ El tiempo de ejecución es **polinomial** con respecto al número de estados y acciones (en el peor de los casos).

- Mejor que otras alternativas, como **algoritmos de búsqueda** en el espacio de políticas.
- La **programación lineal** es otra alternativa que en la mayoría de situaciones *NO* supera a la programación dinámica.

No obstante, los algoritmos de programación dinámica **no son prácticos en problemas con espacios de estados/acciones muy grandes.**

Eficiencia de la programación dinámica

Sobre qué algoritmo de DP elegir, no hay un consenso en el uso de **Iteración de la política** o **Iteración de valor**. Dependerá del coste computacional de las evaluaciones.

En problemas con grandes espacios de estados, los **métodos asíncronos** son una buena alternativa.

También existen **alternativas** basadas en GPI que limitan la computación y memoria:

- ¿Para qué actualizar estados por los que no se va a pasar nunca?
- Es lo que denominamos **actualizaciones selectivas** (*selective updates*).

Eficiencia de la programación dinámica

En general, la principal limitación de la programación dinámica es que se ve considerablemente afectada por la **dimensionalidad del problema** (*the curse of dimensionality*).

- El tamaño del espacio de estados crece exponencialmente a medida que aumenta el número de características relevantes en el problema abordado.

Trabajo propuesto

- Probar los **algoritmos** vistos (¡*prueba a hacer tu propia implementación*!):

```
https://github.com/manjavacas/rl-temario/tree/main/codigo/policy_iteration
```

- Implementación propia de ***policy iteration*** y ***value iteration*** utilizando **Gymnasium**.
- Implementar un **método asíncrono** y comprobar su convergencia.
- Comparar los **tiempos de ejecución** de los algoritmos vistos, variando el número de **estados** y **acciones**.
- Profundizar sobre el concepto "***curse of dimensionality***" y conocer su impacto en RL.

Bibliografía y webs recomendadas:

- **Capítulo 4** de Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction.
- https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html
- https://www.youtube.com/watch?v=Nd1-UUMVfz4&t=2148s
- https://www.youtube.com/watch?v=l87rgLg90HI
- https://www.youtube.com/watch?v=sJIFUTITfBc
- https://www.youtube.com/watch?v=_j6pvGEchWU

Aprendizaje por refuerzo

Programación dinámica

Antonio Manjavacas Lucas

manjavacas@ugr.es