4

PROGRAMACIÓN DINÁMICA

¿QUÉ ES LA PROGRAMACIÓN DINÁMICA?

- Programación Dinámica (DP) en RL es un conjunto de algoritmos para resolver Procesos de Decisión de Markov (MDP) cuando el modelo del entorno es completamente conocido. Utiliza la Ecuación de Bellman para calcular funciones de valor y encontrar políticas óptimas mediante métodos como Iteración de Política e Iteración de Valor.
- Descompone un problema en subproblemas y los resuelve de manera recursiva.
- Requiere conocer la probabilidad de transición $p(s', r|s, a)$, lo que lo hace model-based.
- Es computacionalmente costoso en problemas con muchos estados.

- Programación Dinámica (DP) clásica es una técnica de optimización que divide un problema en subproblemas más pequeños y solapa sus soluciones para evitar cálculos redundantes. Se usa en problemas con estructura óptima de subproblemas y superposición de soluciones.
- Se aplica en problemas como Fibonacci, la mochila, cadenas de ADN, caminos más cortos, etc.
- Se puede implementar de forma recursiva con memoization o iterativa con una tabla.

ITERACIÓN DE LA POLÍTICA E ITERACIÓN DE VALOR

- Hasta ahora vimos 2 algoritmos en cuanto a resolución de MDPs:

  ✓ Evaluación de Política (predicción) => dado π (fijo), determino v(s).

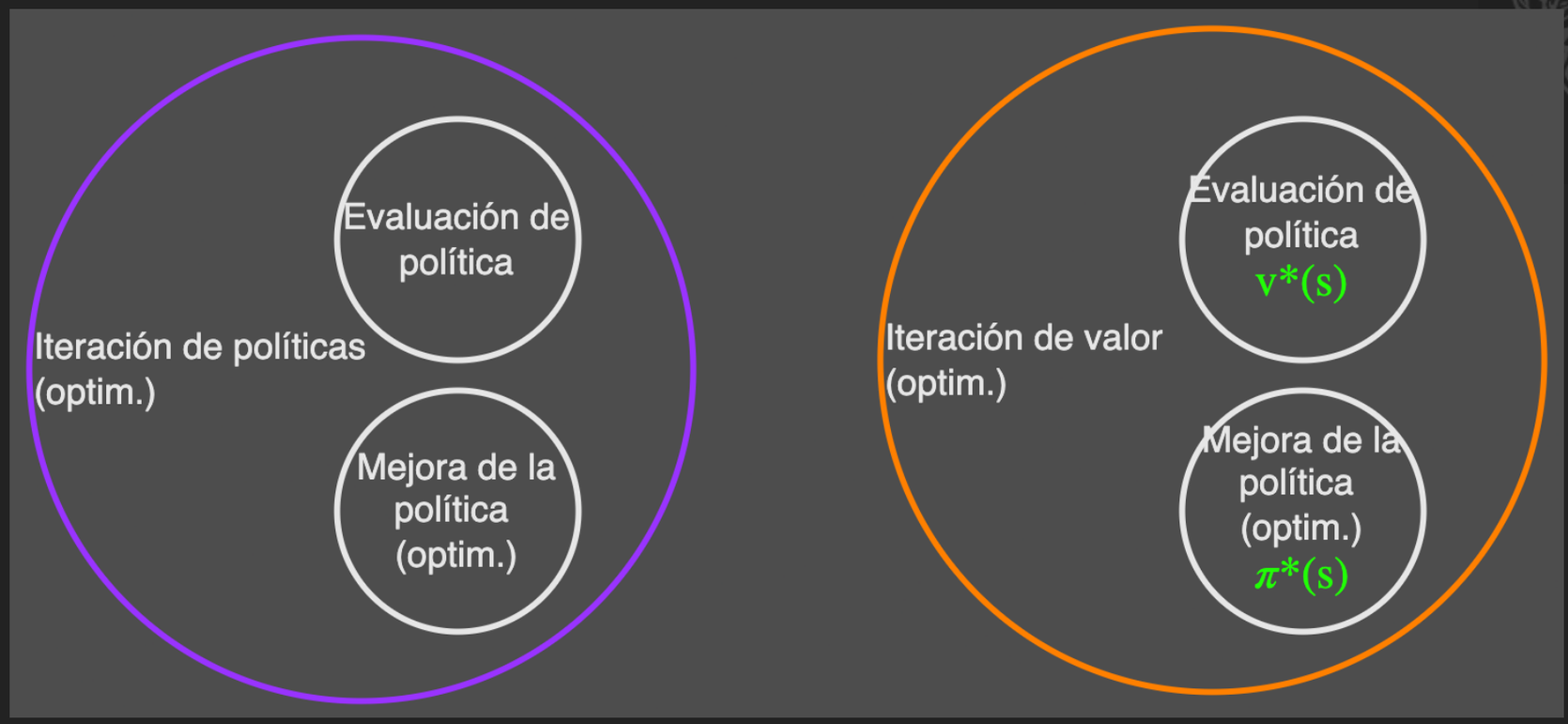
  ✓ Mejora de la Política => cambio π (parcialmente) por π', y luego comparo si $v_{\pi'}(s) >= v_{\pi}(s)$.

ITERACIÓN DE LA POLÍTICA E ITERACIÓN DE VALOR

- ¿Cómo puedo determinar una política óptima en lugar de tener una política fija la cual no sé si es optima?
- Existen 2 formas:

✓ Iteración de Políticas.

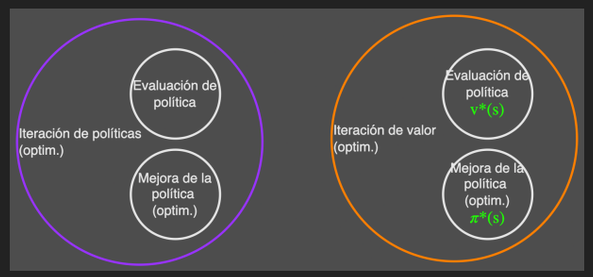
✓ Iteración de Valor.



ITERACIÓN DE LA POLÍTICA E ITERACIÓN DE VALOR

- ¿En qué consiste la Iteración de Políticas y la Iteración de Valor?

Policy Iteration	Value Iteration
fijar un valor de π cualquiera Repetir evaluación de politica() mejora de la politica() hasta que π no cambie	fijar un valor de π cualquiera Repetir calcular v*(s) hasta que v*(s) no cambie Para cada estado de v*(s) calcular π*(s)



ITERACIÓN DE LA POLÍTICA

- ¿En que consiste concretamente la iteración de la política?
- Luego de que mejoramos una política a partir de $v_\pi(s)$ y obtenemos π' , se puede determinar $v_{\pi'}(s)$ y así obtener π''.

$$\pi_0 \xrightarrow{E} v_{\pi 0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi 1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_+}$$

- dónde E es la evaluación de la política y la I es la mejora de la política (Improvement).
- Ambos procesos (E e I) convergen hacia una política óptima y hacia una función de valor óptima.
- Este método se conoce como iteración de la política.

ITERACIÓN DE LA POLÍTICA

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

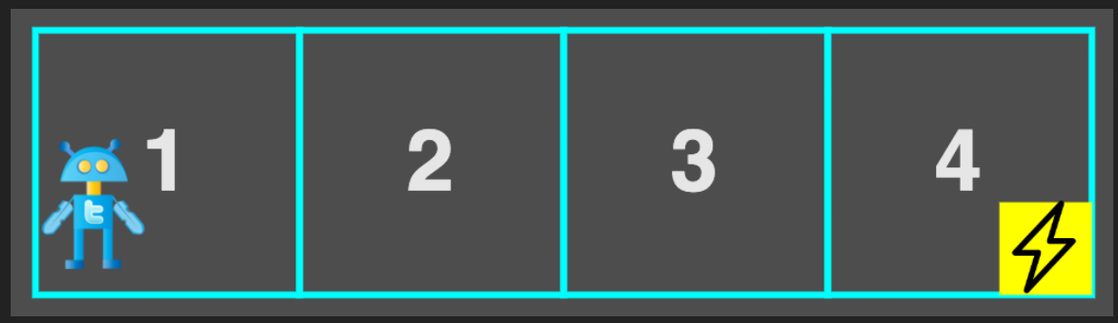
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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 $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
      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

ITERACIÓN DE LA POLÍTICA

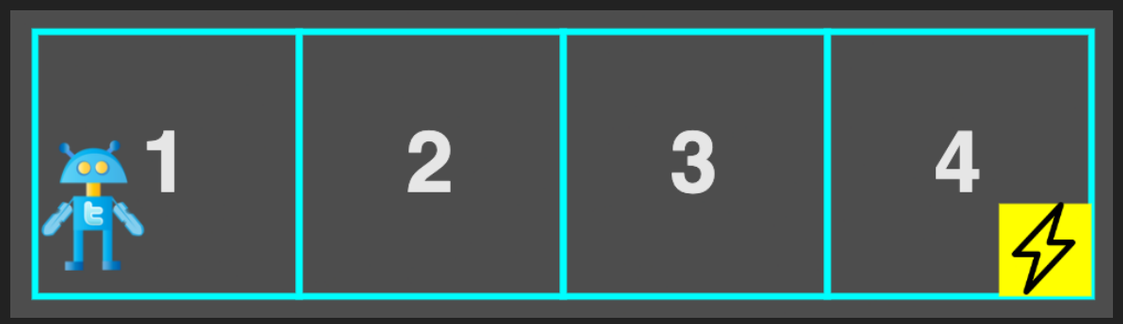
- Ejemplo: Robot limpiador (de un pasillo)



- El robot limpia un pequeño pasillo dividido en 4 casillas numeradas del 1 al 4.
- El robot puede estar en cualquier casilla.
- Su objetivo es llegar a la casilla 4, donde se encuentra la estación de carga.

ITERACIÓN DE LA POLÍTICA

- Estados: Las casillas donde puede estar el robot: {1, 2, 3, 4} (4 es el estado terminal)
- Acciones: El robot solo puede moverse a la izquierda o a la derecha (si es posible).
- Recompensas:
  - ✓ -1: Recibe penalización de -1 en cualquier casilla que no sea la estación de carga (4).
  - ✓ 0: Al llegar a la casilla 4 (estación de carga).
- Transiciones:
  - ✓ Si el robot intenta ir a la izquierda desde la casilla 1, permanece en la casilla 1.
  - ✓ Si el robot intenta ir a la derecha desde la casilla 4, permanece en la casilla 4.
  - ✓ En cualquier otra casilla, el movimiento es determinista (si va a la derecha, se mueve a la derecha; si va a la izquierda, se mueve a la izquierda).
- Factor de Descuento (γ): 0.9 (Damos más valor a las recompensas inmediatas, pero no ignoramos las futuras)



ITERACIÓN DE LA POLÍTICA

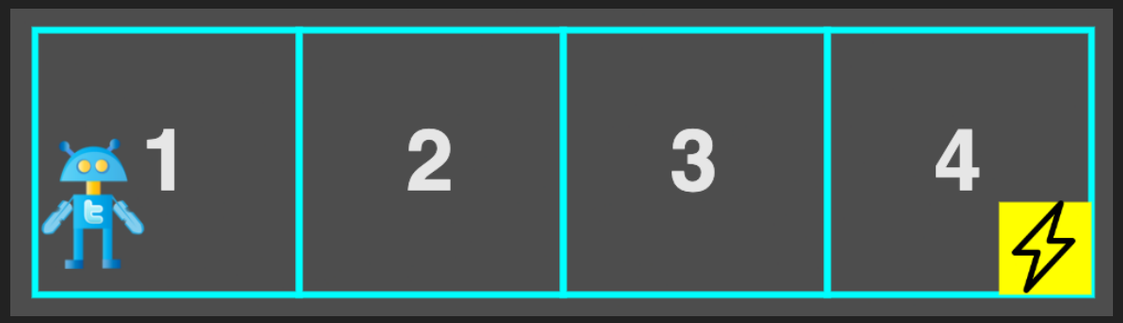
- El proceso de iteración de políticas lo analizaremos en 4 pasos:

✓ Paso 1: Inicialización de la Política y la Función de Valor.

✓ Paso 2: Evaluación de la Política.

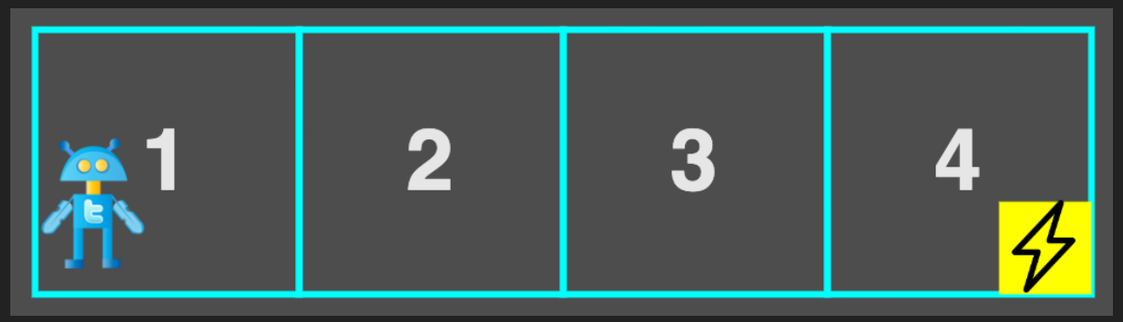
✓ Paso 3: Mejora de la Política.

✓ Paso 4: Repetir Evaluación y Mejora. ↗



ITERACIÓN DE LA POLÍTICA

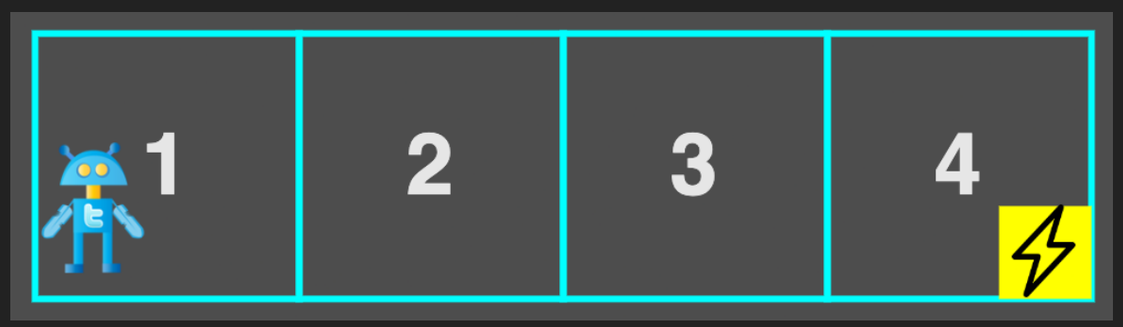
- Paso 1 de 4: Inicialización de la Política y la Función de Valor.
- Política inicial (π): Asignamos una acción aleatoria a cada estado (excepto al estado terminal):
  - ✓ $\pi(1)$ = Derecha (D)
  - ✓ $\pi(2)$ = Izquierda (I)
  - ✓ $\pi(3)$ = Derecha (D)
  - ✓ $\pi(4)$ = No aplica (estado terminal)
- Función de valor inicial (V): Inicializamos los valores de todos los estados en 0.
  - ✓ V(1) = 0
  - ✓ V(2) = 0
  - ✓ V(3) = 0
  - ✓ V(4) = 0



ITERACIÓN DE LA POLÍTICA

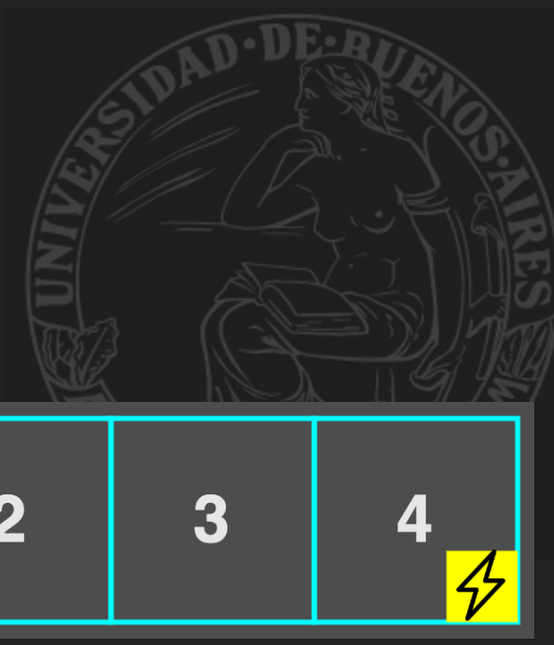
- Paso 2 de 4: Evaluación de la Política.
- Calculamos la función de valor v para la política actual π.

$$V(s) = R(s, \pi(s)) + \gamma * V(s')$$



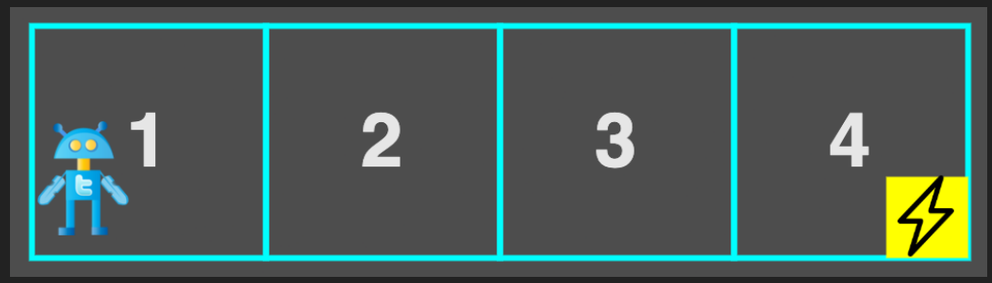
ITERACIÓN DE LA POLÍTICA

- Paso 2 de 4: Evaluación de la Política.
- Iteramos hasta que la función de valor converja (es decir, hasta que los cambios en los valores de los estados sean muy pequeños).
- Iteración 1:
  - ✓ V(1) = -1 + 0.9 * V(2) => V(1) = -1 + 0.9 * 0 = -1
  - ✓ V(2) = -1 + 0.9 * V(1) => V(2) = -1 + 0.9 * -1 = -1.9
  - ✓ V(3) = -1 + 0.9 * V(4) => V(3) = -1 + 0.9 * 0 = -1
  - ✓ V(4) = 0
- Iteración 2:
  - ✓ V(1) = -1 + 0.9 * V(2) => V(1) = -1 + 0.9 * -1.9 = -2.71
  - ✓ V(2) = -1 + 0.9 * V(1) => V(2) = -1 + 0.9 * -2.71 = -3.439
  - ✓ V(3) = -1 + 0.9 * V(4) => V(3) = -1 + 0.9 * 0 = -1
  - ✓ V(4) = 0

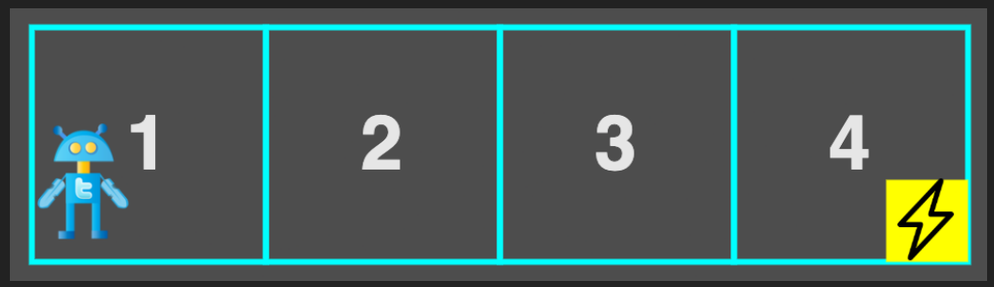


ITERACIÓN DE LA POLÍTICA

- Paso 2 de 4: Evaluación de la Política.
- Iteraciones siguientes... Este proceso continúa hasta que los valores de V(1), V(2) y V(3) dejen de cambiar significativamente.
- Iteración n:
  - ✓ V(1) = -2.71
  - ✓ V(2) = -1.9
  - ✓ V(3) = -1
  - ✓ V(4) = 0



ITERACIÓN DE LA POLÍTICA



- Paso 3 de 4: Mejora de la Política.
- Una vez que tenemos una estimación precisa de la función de valor para la política actual, intentamos mejorar la política. Para cada estado, consideramos todas las posibles acciones y elegimos la acción que maximiza el valor esperado:

$$\pi(s) = \text{argmax\_a } [R(s, a) + \gamma * V(s')]$$

- donde:

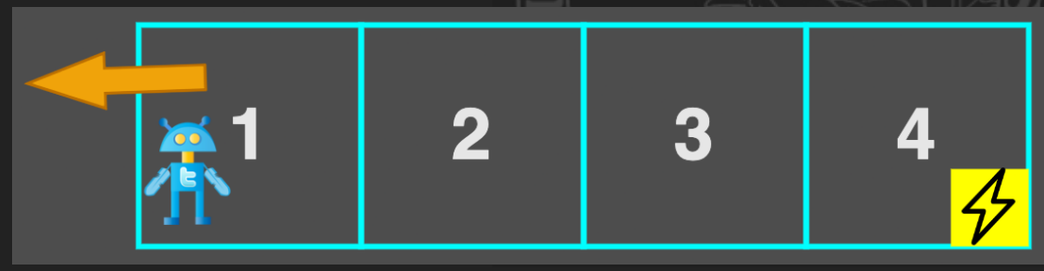
✓ argmax_a significa "la acción a que maximiza la expresión entre corchetes"

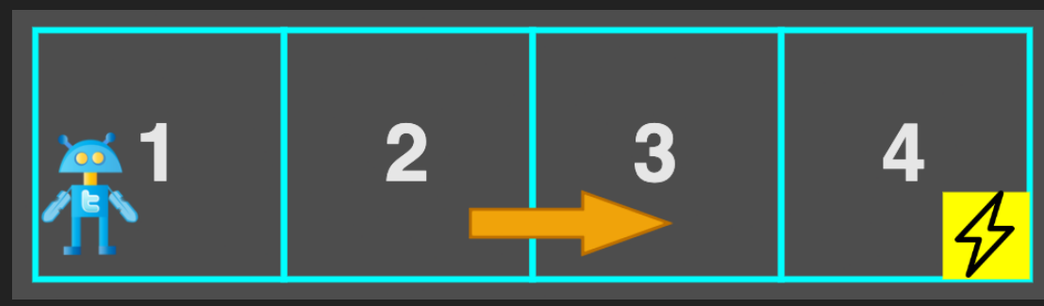
✓ R(s, a): Recompensa inmediata al tomar la acción a en el estado s

✓ V(s'): Valor del estado siguiente después de tomar la acción a en el estado s

ITERACIÓN DE LA POLÍTICA

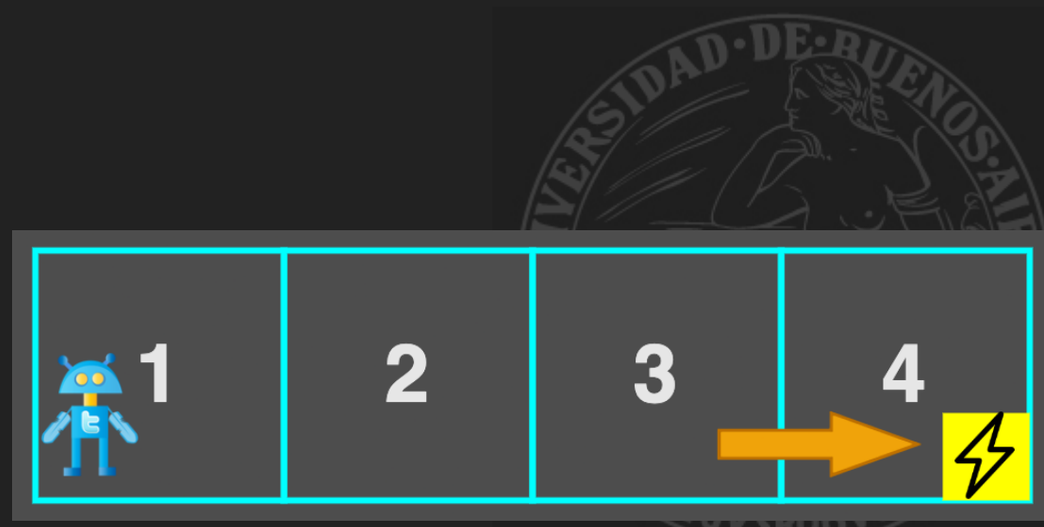
- Paso 3 de 4: Mejora de la Política.
- Realizamos la mejora de la política para cada estado:
- Estado 1:
  - ✓ Acción Izquierda (I): R(1, I) + γ * V(1) = -1 + 0.9 * -7.71 = -7.939
  - ✓ Acción Derecha (D): R(1, D) + γ * V(2) = -1 + 0.9 * -8.439 = -8.5951
  - ✓ Como -7.939 > -8.5951, $\pi(1)$ = Izquierda (I) (La política ahora recomienda ir a la izquierda en el estado 1)
- Estado 2:
  - ✓ Acción Izquierda (I): R(2, I) + γ * V(1) = -1 + 0.9 * -7.71 = -7.939
  - ✓ Acción Derecha (D): R(2, D) + γ * V(3) = -1 + 0.9 * -1 = -1.9
  - ✓ Como -1.9 > -7.939, $\pi(2)$ = Derecha (D) (La política ahora recomienda ir a la derecha en el estado 2)

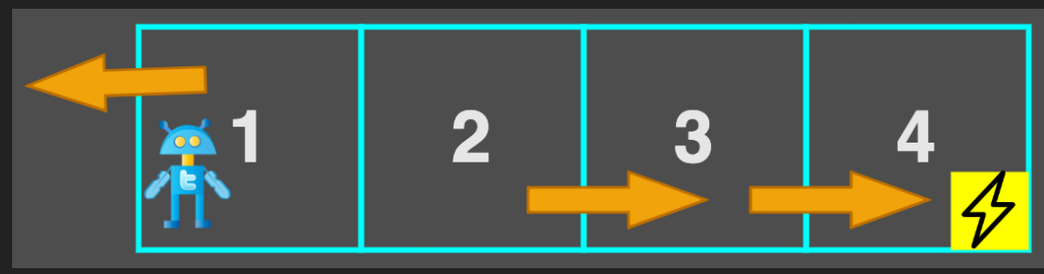




ITERACIÓN DE LA POLÍTICA

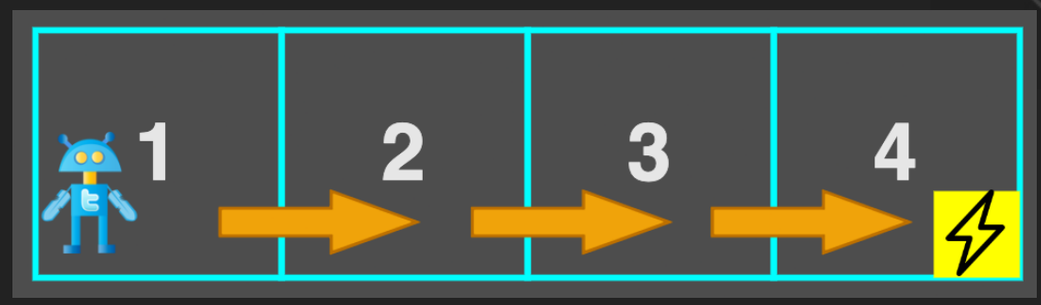
- Paso 3 de 4: Mejora de la Política.
- Estado 3:
  - ✓ Acción Izquierda (I): R(3, I) + γ * V(2) = -1 + 0.9 * -8.439 = -8.5951
  - ✓ Acción Derecha (D): R(3, D) + γ * V(4) = -1 + 0.9 * 0 = -1
  - ✓ Como -1 > -8.5951, $\pi(3)$ = Derecha (D) (La política ahora recomienda ir a la derecha en el estado 3)
- Nuestra nueva política es:
  - ✓ $\pi(1)$ = Izquierda (I)
  - ✓ $\pi(2)$ = Derecha (D)
  - ✓ $\pi(3)$ = Derecha (D)
  - ✓ $\pi(4)$ = No aplica





ITERACIÓN DE LA POLÍTICA

- Paso 4 de 4: Repetir Evaluación y Mejora.
- Volvemos al paso 2 (Evaluación de la Política) y repetimos el proceso con la nueva política. Calculamos una nueva función de valor para esta política mejorada y luego intentamos mejorar la política nuevamente.
- Continuar hasta que la política ya no cambie. Cuando la política se mantiene igual después de una iteración de mejora, significa que hemos encontrado la política óptima.
- Política Óptima (Después de 4 iteraciones):
  - ✓ $\pi(1)$ = Derecha (D)
  - ✓ $\pi(2)$ = Derecha (D)
  - ✓ $\pi(3)$ = Derecha (D)
  - ✓ $\pi(4)$ = No aplica



- Es decir, la política óptima es que el robot siempre se mueva a la derecha hasta llegar a la estación de carga.

ITERACIÓN DE VALOR

- ¿En que consiste concretamente la iteración de valor?
- En la Iteración de Políticas, debemos iterar hasta que la función de valor v(s) converja (o esté "suficientemente cerca" de la convergencia) para la política actual antes de mejorar la política.
- Este proceso de evaluación puede ser costoso computacionalmente, especialmente para problemas grandes.

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

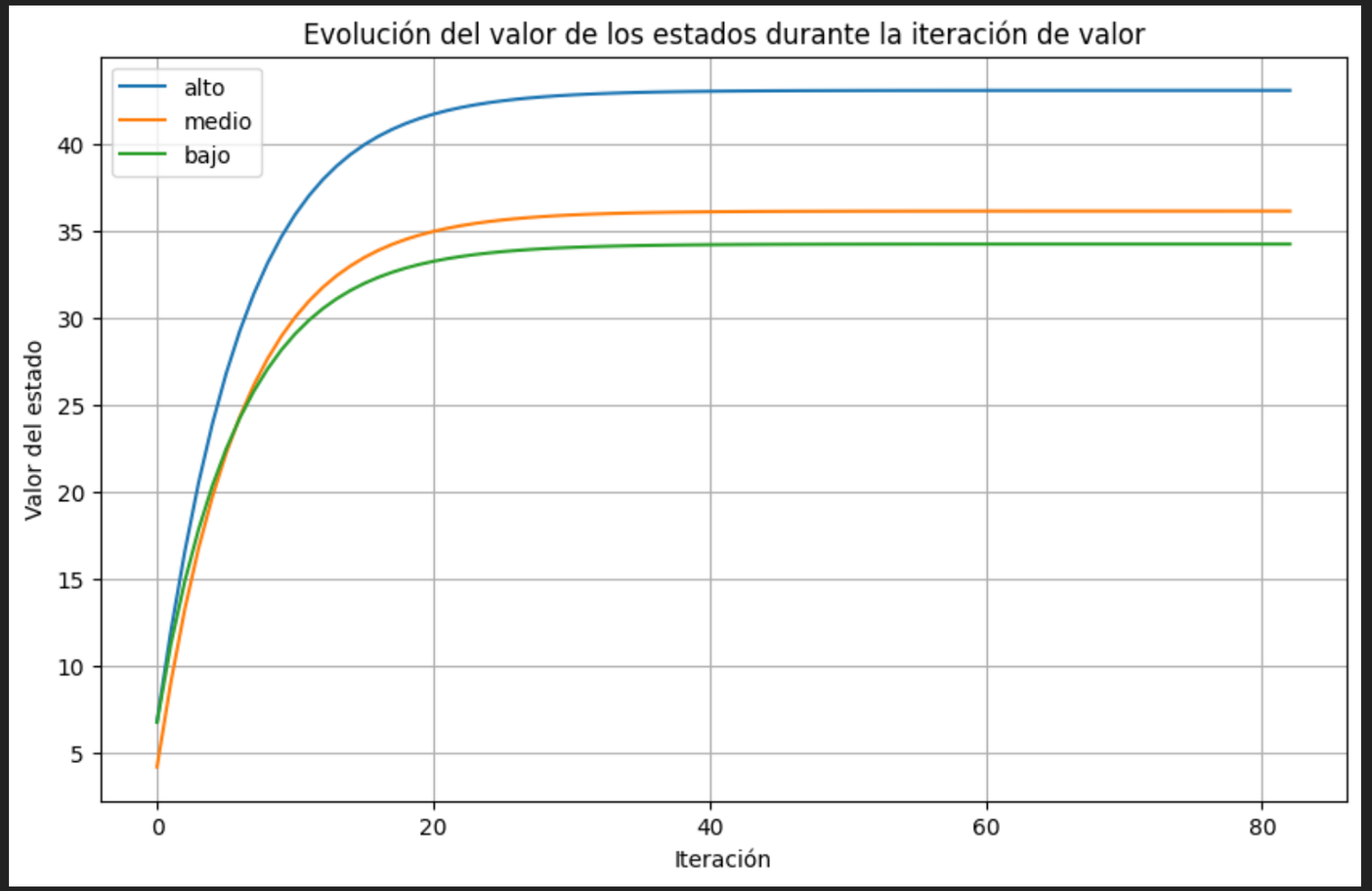
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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 $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
      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

ITERACIÓN DE VALOR

- En la práctica, después de un cierto número de iteraciones, la evaluación de la política puede tener poco efecto sobre la convergencia final y también sobre la determinación de la política óptima.
- Esto significa que podemos truncar el proceso de evaluación de la política sin sacrificar significativamente la calidad de la solución.



ITERACIÓN DE VALOR

- Una forma de truncar el proceso de evaluación de la política es, como dices, calcular la mejor acción posible en cada estado y usar esa acción para actualizar la función de valor v(s).
- Esto es precisamente lo que hace la Iteración de Valor.

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

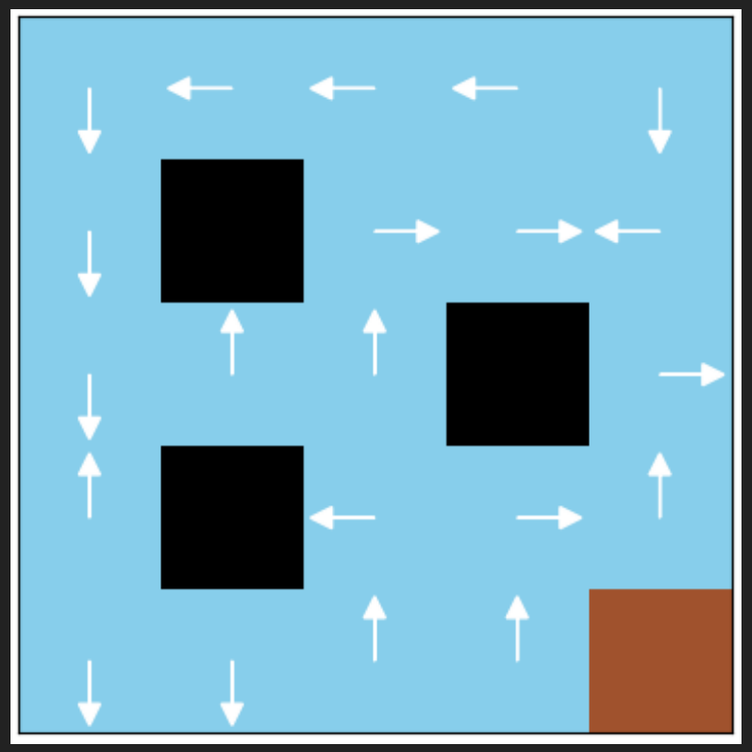
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

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

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

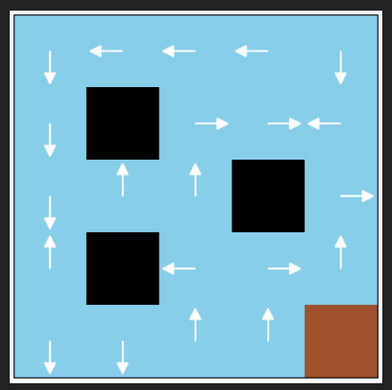
ITERACIÓN DE VALOR

- Ejemplo: Navegación de un Barco en un Océano
- El océano está dividido en una cuadrícula de celdas.
- Algunas celdas son agua navegable, otras son rocas o islotes (obstáculos), y una celda es el muelle (destino final).
- El objetivo del barco es llegar al muelle lo más rápido posible evitando las rocas.



ITERACIÓN DE VALOR

- Estados: Se representan por cada celda navegable en la cuadrícula del océano. El muelle es el estado terminal.
- Acciones: El barco puede moverse en cuatro direcciones: Norte, Sur, Este, Oeste. Si el barco intenta moverse hacia una roca o fuera de los límites del océano, permanece en su celda actual.
- Recompensas:
  - ✓ -1: Por cada movimiento en una celda navegable que no sea el muelle.
  - ✓ 0: Al llegar al muelle.
  - ✓ -100: Si el barco choca con una roca. (Aunque, estrictamente hablando, el barco no puede chocar con una roca porque permanece en su lugar, esta penalización influye en el valor de las celdas adyacentes a las rocas, incentivando al barco a evitarlas).
- Factor de Descuento (γ): 0.9 (Queremos llegar al muelle rápidamente).



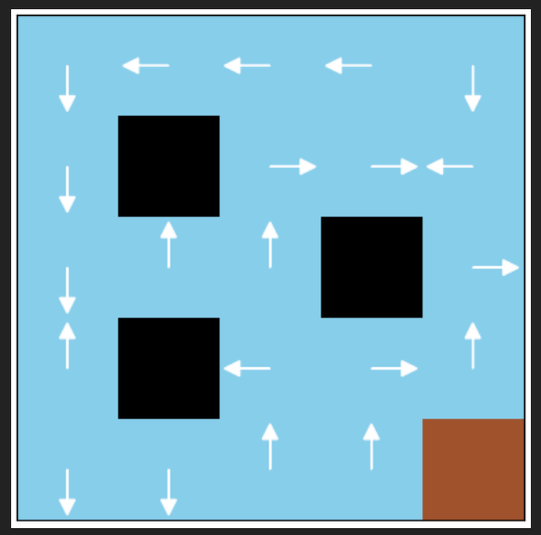
ITERACIÓN DE VALOR

- El proceso de iteración de valor lo analizaremos en 3 pasos:

✓ Paso 1: Inicialización.

✓ Paso 2: Evaluación de la Política.

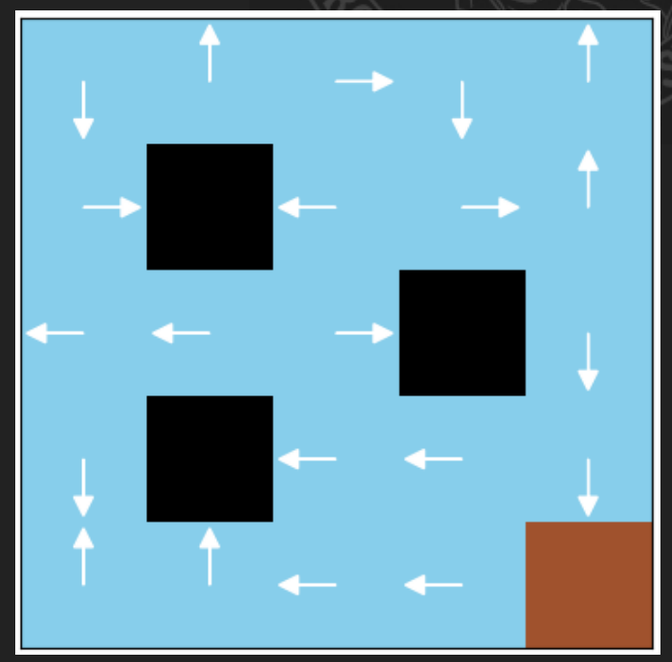
✓ Paso 3: Extracción de la Política Óptima



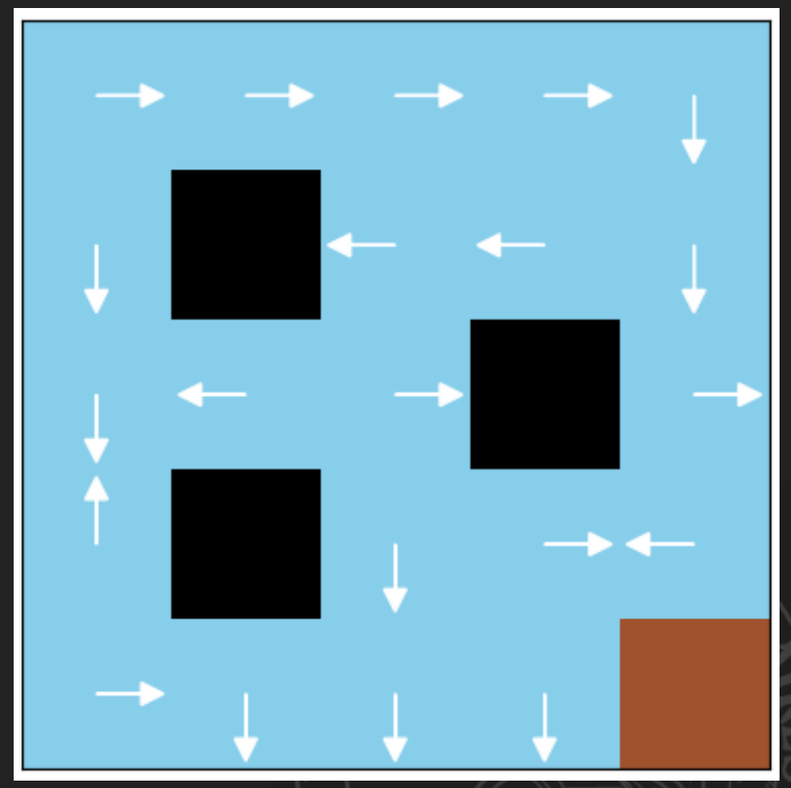
ITERACIÓN DE VALOR

- Paso 1 de 3: Inicialización.
- Entorno: Océano de 5x5, muelle en (4, 4), rocas en (1, 1), (2, 3), (3, 1).
- Factor de Descuento (γ): 0.9
- Función de Valor Inicial (V(s)): 0 para todas las celdas.
- Estado Inicial: (0, 0)

```python
# Definición del Entorno
LADO_OCEANO = 5  # Tamaño del oceano (cuadrado de 5x5)
MUELLE = (4, 4)  # Coordenadas del muelle (fila, columna)
ROCAS = [(1, 1), (2, 3), (3, 1)]  # Coordenadas de las rocas
ACCIONES = ['Norte', 'Sur', 'Este', 'Oeste']
RECOMPENSA_MOVIMIENTO = -1
RECOMPENSA_MUELLE = 0
RECOMPENSA_ROCA = -100
FACTOR_DESCUENTO = 0.9
TOLERANCIA = 1e-6
```

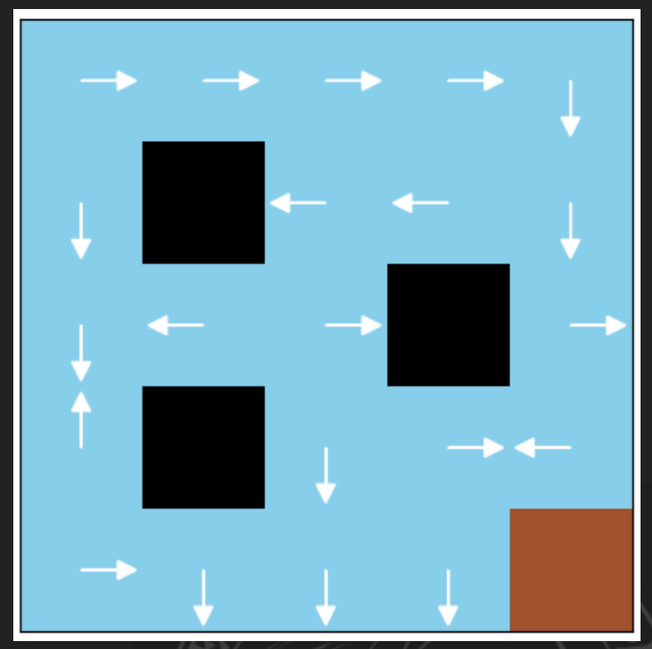


ITERACIÓN DE VALOR



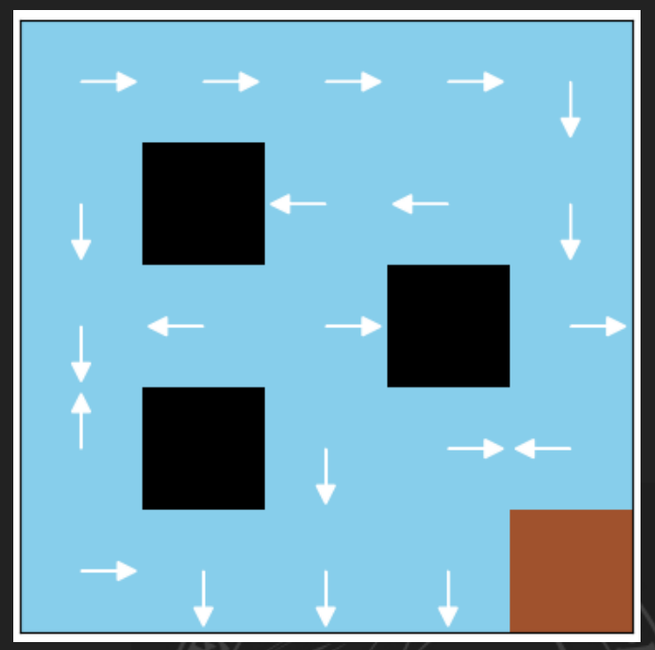
- Paso 2 de 3: Evaluación de la Política.
- Iteración 1:
- 1. Estado (0, 0):
  - ✓ Acciones Posibles: Norte, Sur, Este, Oeste.
- 2. Calculamos el valor de cada acción:
  - ✓ Norte: $R(0, 0, \text{Norte}) + \gamma * V(1, 0) = -1 + 0.9 * 0 = -1$
  - ✓ Sur: $R(0, 0, \text{Sur}) + \gamma * V(0, 1) = -1 + 0.9 * 0 = -1$
  - ✓ Este: $R(0, 0, \text{Este}) + \gamma * V(0, 1) = -1 + 0.9 * 0 = -1$
  - ✓ Oeste: $R(0, 0, \text{Oeste}) + \gamma * V(0, 0) = -1 + 0.9 * 0 = -1$
- $V(0, 0) = \max(-1, -1, -1, -1) = -1$
  En este punto, cualquier acción es igualmente buena (o igualmente mala). Arbitrariamente, elegimos Este. El barco se mueve a (0, 1).

ITERACIÓN DE VALOR



- Paso 2 de 3: Evaluación de la Política.
- Iteración 1:
- 1. Estado (0, 1):
  - ✓ Acciones Posibles: Norte, Sur, Este, Oeste.
- 2. Calculamos el valor de cada acción (usando el valor actualizado de V(0, 0) = -1):
  - ✓ Norte: R(0, 1, Norte) + γ * V(1, 1) = -1 + 0.9 * 0 = -1
  - ✓ Sur: R(0, 1, Sur) + γ * V(0, 0) = -1 + 0.9 * (-1) = -1.9
  - ✓ Este: R(0, 1, Este) + γ * V(0, 2) = -1 + 0.9 * 0 = -1
  - ✓ Oeste: R(0, 1, Oeste) + γ * V(0, 0) = -1 + 0.9 * (-1) = -1.9
- V(0, 1) = max(-1, -1.9, -1, -1.9) = -1
  En este punto, Norte, Este y Oeste son igualmente buenas. Arbitrariamente, elegimos Este. El barco se mueve a (0, 2).

ITERACIÓN DE VALOR



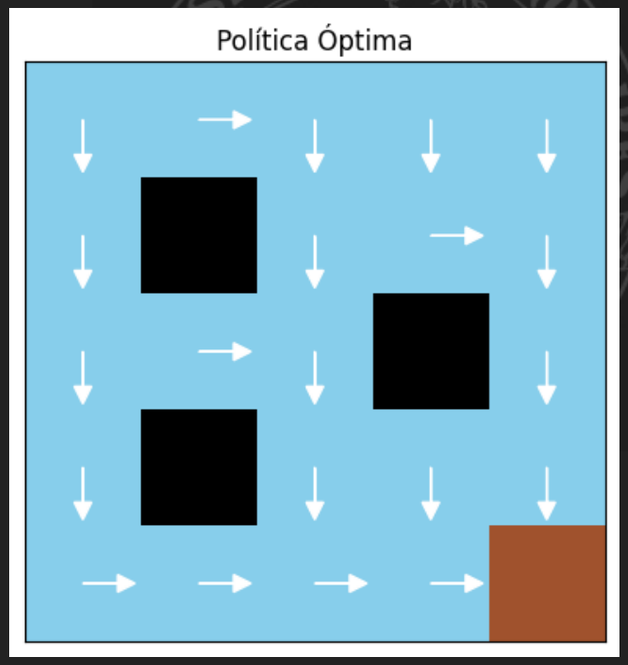
- Paso 2 de 3: Evaluación de la Política.
- Iteración 1:
- 1. Estado (0, 2):
  - ✓ Acciones Posibles: Norte, Sur, Este, Oeste.
- 2. Calculamos el valor de cada acción (usando el valor actualizado de V(0, 0) = -1 y V(0, 1) = -1):
  - ✓ Norte: R(0, 2, Norte) + γ * V(1, 2) = -1 + 0.9 * 0 = -1
  - ✓ Sur: R(0, 2, Sur) + γ * V(0, 1) = -1 + 0.9 * (-1) = -1.9
  - ✓ Este: R(0, 2, Este) + γ * V(0, 3) = -1 + 0.9 * 0 = -1
  - ✓ Oeste: R(0, 2, Oeste) + γ * V(0, 1) = -1 + 0.9 * (-1) = -1.9
- V(0, 2) = max(-1, -1.9, -1, -1.9) = -1
  En este punto, cualquier acción es igualmente buena. Arbitrariamente, elegimos Este. El barco se mueve a (0, 3).

ITERACIÓN DE VALOR

- Paso 3 de 3: Extracción de la Política Óptima.
- Una vez que la función de valor V(s) ha convergido, podemos extraer la política óptima para cada celda navegable:

$$\pi(s) = \text{argmax\_a } [R(s, a) + \gamma * V(s')]$$

- Esto significa que para cada celda s, la política óptima $\pi(s)$ elige la acción a (Norte, Sur, Este, Oeste) que maximiza el valor esperado de la celda resultante s'.
- Ejemplo: Si después de la iteración de valor, encontramos que para una celda s:
  - ✓ (-1 + 0.9 * V(norte)) = -2
  - ✓ (-100 + 0.9 * V(s)) = -105
  - ✓ (-1 + 0.9 * V(sur)) = -1.5
  - ✓ (-1 + 0.9 * V(oeste)) = -1.8
- Entonces, la política óptima para la celda s sería ir al Sur (porque -1.5 es el valor más alto).



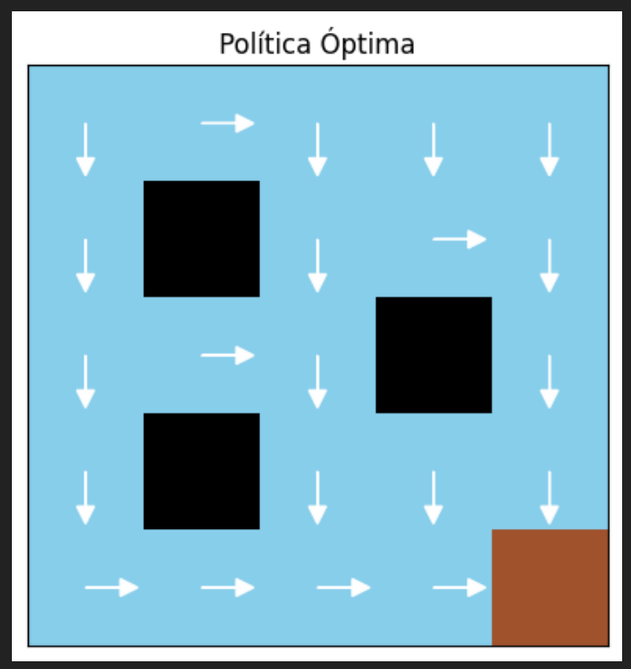
ITERACIÓN DE VALOR

- Observaciones

✓ Con cada iteración, los valores de los estados (celdas) se ajustan y se vuelven más negativos. Esto es porque estamos penalizando cada movimiento con una rcompensa de -1.

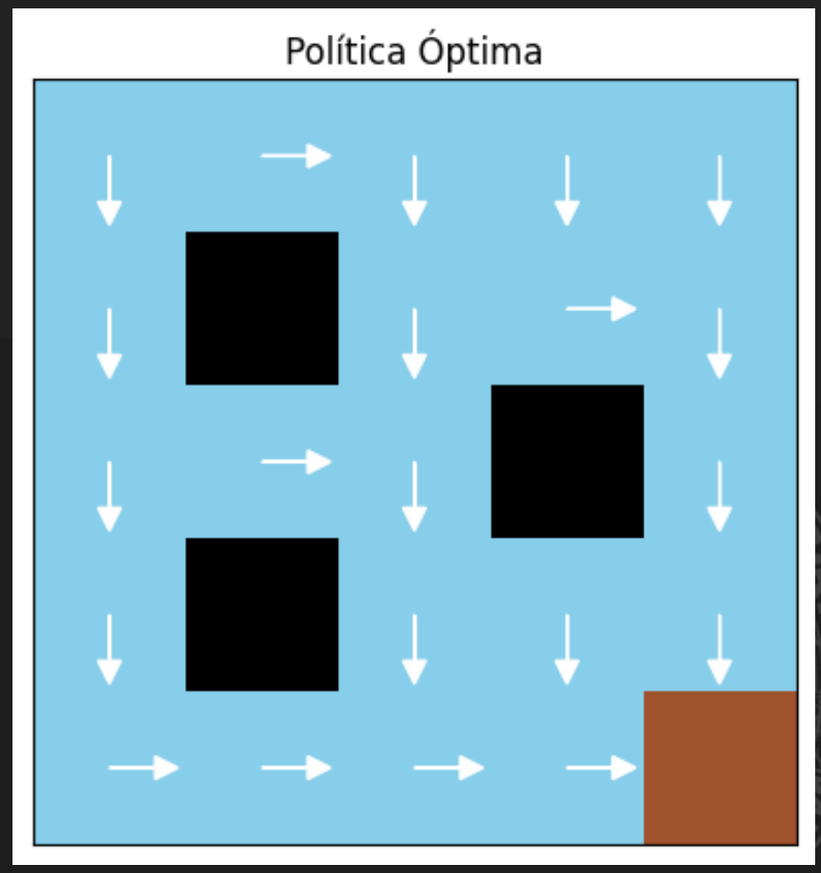
✓ Después de varias iteraciones, la política (la acción que el barco elige en cada estado) comenzará a converger hacia la ruta óptima al muelle. Por ejemplo, el barco comenzará a aprender a evitar las rocas y a moverse hacia el este y el sur para llegar al muelle.

✓ Este proceso de iteración continúa hasta que los valores de los estados dejen de cambiar significativamente. En este punto, hemos encontrado la función de valor óptima y podemos extraer la política óptima.



ITERACIÓN DE VALOR

- Implementación en Python




```python
# algoritmo de iteracion de valor
iteracion = 0
while True:
    iteracion += 1
    delta = 0  # para verificar la convergencia

    # copia de la nueva finco de valor v(s)
    funcion_valor_anterior = funcion_valor.copy()

    for fila in range(LADO_OCEANO):
        for columna in range(LADO_OCEANO):
            if es_roca(fila, columna):
                continue  # no se actualizan las rocas
            if (fila, columna) == MUELLE:
                continue # no actualiza el valor del muelle
            # aqui se calcula el valor de cada celda con la ecuación de Bellman
            valores_acciones = []
            for accion in ACCIONES:
                nueva_fila, nueva_columna = transicion(fila, columna, accion) # transicion me devuelve la nueva posicion de barco (nuevo estado)
                recompensa_inmediata = recompensa(fila, columna, accion) # recompensa devuelve 0: muelle, -1: avance, -100:colision con roca
                valores_acciones.append(recompensa_inmediata + FACTOR_DESCUENTO * funcion_valor[nueva_fila, nueva_columna]) # ecuacion de bellman

            funcion_valor[fila, columna] = max(valores_acciones)
            delta = max(delta, abs(funcion_valor[fila, columna] - funcion_valor_anterior[fila, columna]))

    print(f"Iteración {iteracion}: Delta = {delta}")
    # impresion poe consola del valor de cada celda en formato matricial
    for i in range(LADO_OCEANO):
        print(funcion_valor[i,:])
    if delta < TOLERANCIA:
        break
```

ITERACIÓN DE LA POLÍTICA GENERAL (GPI)

- Muchos de los algoritmos de Aprendizaje por Refuerzo comparten una estructura común (en general):

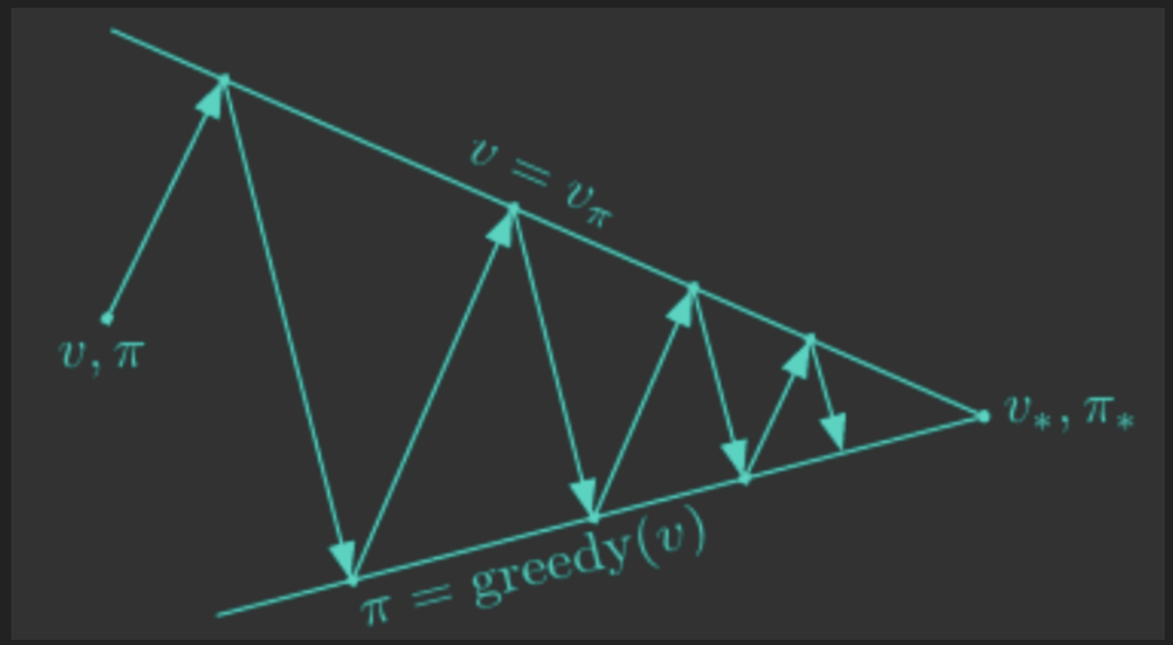
    1. Tienen un proceso para evaluar la "bondad" de la política actual (policy evaluation).
    2. Tienen un proceso para mejorar la política basándose en esta evaluación (policy improvement).

- El concepto de GPI establece que muchos algoritmos de RL (tales como Iteración de Políticas, Iteracion de Valor, SARSA, Q-Learning, entre otros) pueden ser entendidos como instancias que se caracterizan por la interacción de dos procesos: Evaluación de la Política y Mejora de la Política.

- La forma en que se implementan estos procesos puede variar ampliamente, pero la idea central de iterar entre la evaluación y la mejora de la política sigue siendo la misma.

ITERACIÓN DE LA POLÍTICA GENERAL (GPI)

- La condición de parada se produce cuando la política no se puede mejorar al realizar los procesos de actualización.
- Si ambos procesos (Evaluación y Mejora) se estabilizan significa que v y π son óptimos.
- La función de valor v es estable cuando esta de acuerdo con la política actual y π es estable cuando codiciosa (greedy) con respecto a v.



CONCLUSIONES

- Programación Dinámica (DP) se usa para resolver Procesos de Decisión de Markov (MDP).
- Iteración de Valor e Iteración de Políticas son algoritmos fundamentales y son la base de muchos algoritmos en Aprendizaje por Refuerzo (RL).
- Iteración de Valor e Iteración de Política son aplicados en áreas como control óptimo, robótica, planificación* y economía cuando el modelo es conocido.

* Implica usar un modelo de un entorno para analizar y mejorar decisiones antes de ejecutarlas en el entorno real.

CLASIFICACIÓN DE TÉCNICAS DE RL

- En RL decir modelo o dinámica del entorno son sinónimos.
- La dinámica del entorno es el conjunto de reglas que determinan cómo cambia el estado en función de las acciones y qué recompensas se obtienen.
- Un modelo describe las transiciones de estado: $p(s', r \mid s, a)$, que nos dice con qué probabilidad terminaremos en el estado s' y recibiremos la recompensa r tras tomar la acción a en el estado s.
- En RL tenemos 2 tipos de algoritmos:

  ✓ Based-Model <– (conozco $p(s', r \mid s, a)$)

  ✓ Free-Model <– (no conozco $p(s', r \mid s, a)$)

CLASIFICACIÓN DE TÉCNICAS DE RL

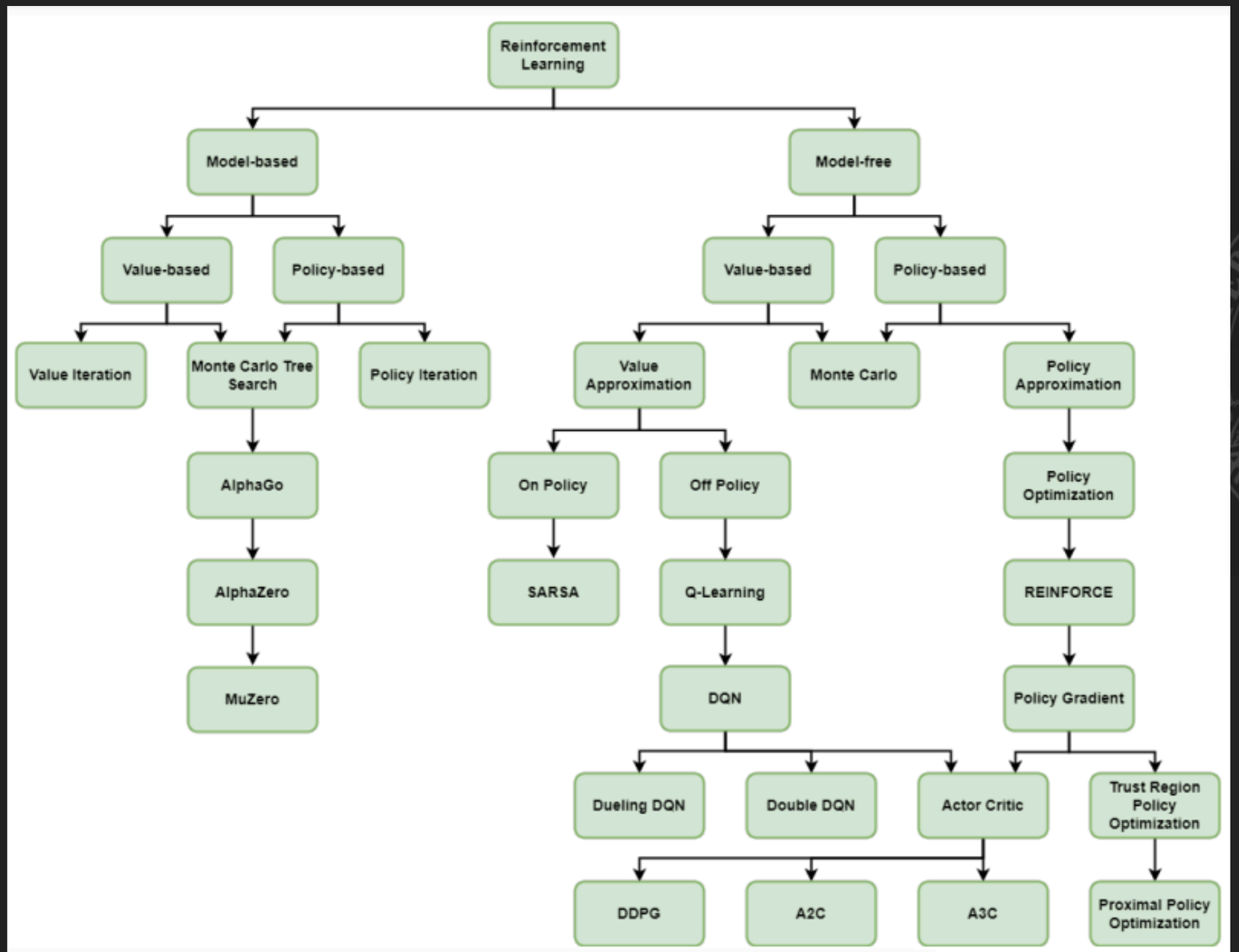


Figure 5: A classification of deep reinforcement learning methods (adopted from (Mazyavkina et al., 2021))

REFERENCIAS BIBLIOGRÁFICAS Y WEB (I)

- R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
- Khadivi, M., Charter, T., Yaghoubi, M., Jalayer, M., Ahang, M., Shojaeinasab, A., & Najjaran, H. (2025). Deep reinforcement learning for machine scheduling: Methodology, the state-of-the-art, and future directions. Computers & Industrial Engineering, 110856.
- N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. Reinforcement learning for combinatorial optimization: A survey. Computers & Operations Research, 134:105400, 2021.