

## Planificación automática

La planificación es el proceso de razonamiento explícito en el que, anticipando el resultado esperado de las posibles acciones a realizar, se seleccionan y organizan estas de tal manera que permitan alcanzar algún objetivo preestablecido.

La planificación automática es el campo de la inteligencia artificial que estudia este proceso de razonamiento de manera computacional. Por un lado, la planificación es un componente importante del comportamiento racional, por lo que desde un punto de vista científico su estudio es necesario para poder entender qué es la inteligencia. Por otro lado, desde el punto de vista de la ingeniería, el desarrollo de planificadores (es decir, programas que son capaces de planificar) es fundamental en la construcción de sistemas autónomos (un ejemplo de estos sistemas serían los robots exploradores de Marte, que no pueden ser controlados en tiempo real y, por tanto, deben ser capaces de planificar por sí mismos la mejor manera de alcanzar el objetivo establecido desde el centro de control).

Existen tres tipos principales de planificadores:

- **Planificadores específicos de un dominio:** están optimizados para un dominio concreto como, por ejemplo, la planificación del movimiento de un robot. Usan técnicas particulares para el problema en cuestión, que son difíciles de generalizar a otros dominios.
- **Planificadores independientes del dominio:** no usan ningún conocimiento específico del problema, por lo que, en principio, se pueden emplear en cualquier dominio. En la práctica, es necesario considerar simplificaciones acerca del dominio para que su utilización sea viable.
- **Planificadores configurables:** son planificadores independientes del dominio, pero que requieren información específica del problema como parte de los datos de entrada.

Este tema se enmarca dentro de lo que se conoce como planificación clásica, en la que, para facilitar el desarrollo de planificadores independientes del dominio, se consideran únicamente dominios que satisfacen las siguientes ocho restricciones:

1. Solo hay una cantidad finita de posibles situaciones en las que se puede encontrar el problema y solo es posible aplicar un conjunto finito de acciones (**sistema finito**).
2. Siempre se dispone del conocimiento completo acerca del estado en que se encuentra el problema (**sistema completamente observable**).
3. Siempre se obtiene el mismo resultado al aplicar la misma acción al mismo estado (**sistema determinista**).

4. El estado en que se encuentra el problema únicamente cambia al aplicar una acción (**sistema estático**).
5. El objetivo de la planificación es conseguir que el problema se encuentre en uno de los estados establecidos como objetivos (**consecución de estados objetivos**).
6. Una solución al problema consiste de una secuencia consecutiva de acciones (**planes secuenciales**).
7. La aplicación de las acciones se considera instantánea, es decir, las acciones no tienen duración (**tiempo implícito**).
8. Mientras se planifica una solución no se produce ningún cambio en el problema (**planificación fuera de línea**).

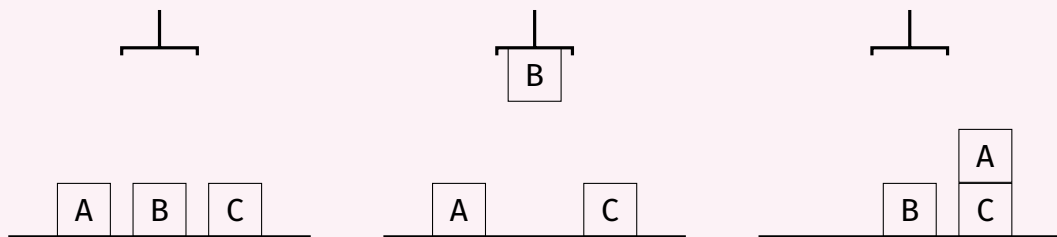
En este marco, un problema de planificación se puede entender como un sistema de transición de estados. Es decir, es una tupla  $P = (S, A, s_I, S_G)$  donde

- $S$  es un **conjunto finito de estados**.
- $A$  es un **conjunto finito de acciones** que hacen cambiar al sistema de un estado a otro.
- $s_I$  es el **estado inicial del problema**.
- $S_G$  es el **subconjunto de estados objetivo**, alguno de los cuales se pretende alcanzar.

### Ejemplo: El mundo de los bloques

Un dominio estándar en planificación automática es el mundo de los bloques: se tiene una mesa sobre la que hay un conjunto de bloques cúbicos; los bloques se pueden apilar, pero sobre cada bloque solo se puede colocar un bloque; se tiene también un brazo robótico que puede coger cualquier bloque, uno cada vez, y moverlo a otra posición, ya sea sobre la mesa o sobre otro bloque; el objetivo es, partiendo de una disposición inicial de los bloques, alcanzar alguna disposición específica (por ejemplo, todos los bloques apilados uno encima de otro en algún orden determinado).

Si consideramos que únicamente hay tres bloques, A, B y C, algunos posibles estados en los que podría encontrarse el problema serían:



Es fácil comprobar que el problema de comenzar con una cierta disposición de los bloques y conseguir colocar estos en una de entre varias disposiciones preestablecidas, es un problema de planificación clásica: hay una cantidad finita de

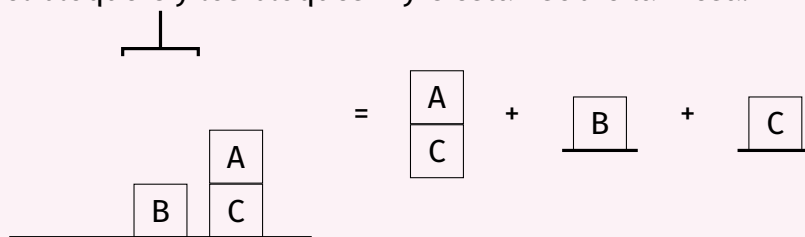
estados y de acciones, ya que, o bien el brazo robótico coge un bloque, o bien suelta un bloque sobre la mesa o sobre otro bloque; se conoce perfectamente toda la información que determina en qué estado se encuentra el problema, que es saber dónde se encuentra cada bloque: sobre la mesa, sobre otro bloque o cogido por el brazo robótico; el resultado de coger o soltar un bloque está completamente determinado; la disposición de los bloques solo se puede cambiar mediante las acciones del brazo robótico; el objetivo es llegar a una disposición de los bloques entre varias posibles, es decir, a ciertos estados del problema; esa disposición se alcanzará con una secuencia consecutiva de acciones del brazo robótico; no interesa el tiempo que se tarde en llevar a cabo cada acción, por lo que las podemos considerar instantáneas; además, la disposición de los bloques no cambia mientras se determina esa secuencia de acciones.

Es evidente que el mundo de los bloques es un dominio de planificación clásica desde un punto de vista abstracto. En la realidad no sería así (por ejemplo, colocar un bloque exactamente sobre otro sin que sobresalga requeriría una precisión extrema del brazo robótico), pero es una buena metodología tratar de resolver una versión simplificada del problema e ir luego incorporando las complejidades que aparecen en el mundo real.

Una característica que diferencia a los problemas de planificación automática de los sistemas de transición genéricos es el uso de una representación factorizada para los estados del problema. Esta es una representación estructurada que, en lugar de considerar los estados como un todo, los considera como una agregación de componentes simples.

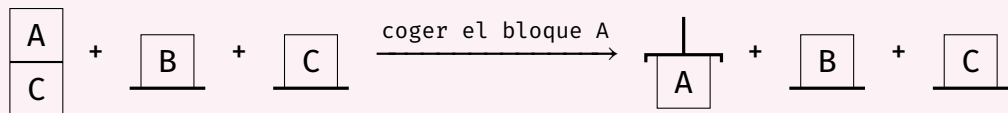
### Ejemplo: Representación factorizada del mundo de los bloques

En el mundo de los bloques podemos recuperar un estado completo simplemente a partir de la posición ocupada por cada bloque, es decir, si lo ha cogido el brazo robótico o, en caso contrario, sobre qué está colocado. Por ejemplo, el siguiente estado queda completamente determinado conociendo simplemente que el bloque A está sobre el bloque C y los bloques B y C están sobre la mesa:



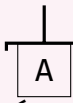
El uso de una representación factorizada proporciona varias ventajas:

1. Permite determinar el estado que resulta de aplicar una acción a un estado especificando únicamente qué componentes de este último cambian y cómo lo hacen. El resto de componentes permanecerán inalteradas.

**Ejemplo: Aplicación de una acción en el mundo de los bloques**

2. Permite detectar la necesidad de aplicar ciertas acciones antes siquiera de haber determinado qué acciones se aplicarán previamente.

**Ejemplo: Acción necesaria en el mundo de los bloques**

Si un estado objetivo contiene la componente , entonces puede deducirse que, para alcanzarlo, posiblemente en algún momento será necesario aplicar la regla coger el bloque A.

3. Permite definir heurísticas independientes del dominio, a partir del cómputo del número de acciones aplicables.

## Representación de problemas de planificación

Para representar problemas de planificación clásica se usará el formalismo STRIPS (*STanford Research Institute Problem Solver*). En este formalismo, un problema de planificación viene especificado por

1. Un conjunto finito de hechos (también llamados proposiciones), representados por símbolos de predicados aplicados a símbolos de constantes.
2. Un conjunto finito de acciones. Para cada una de ellas es necesario proporcionar su nombre, sus precondiciones (hechos que deben cumplirse para que se pueda aplicar la acción), su lista de adición (hechos que se cumplen tras aplicar la acción) y su lista de borrado (hechos que dejan de cumplirse tras aplicar la acción).
3. Un estado inicial (hechos que se cumplen inicialmente).
4. Un objetivo (hechos que se quiere que se cumplan).

El conjunto de posibles hechos y acciones conforman el dominio del problema.

Los estados de un problema, incluido el inicial, se representan mediante conjuntos de hechos. Se asume la hipótesis del mundo cerrado, es decir, se asume que los hechos no incluidos en un estado no se cumplen.

Una acción será aplicable a un estado si este contiene todas sus precondiciones. El resultado es un estado en el que se han eliminado todos los hechos de la lista de borrado de la acción y se han añadido todos los hechos de la lista de adición.

Los estados objetivos son aquellos que contienen todos y cada uno de los hechos objetivos.

Una solución del problema será una secuencia de acciones que, al aplicarlas una detrás de otra partiendo del estado inicial, da como resultado un estado objetivo.

A continuación proporcionamos como ejemplos el desarrollo de la representación STRIPS de dos problemas de planificación.

## El problema de la rueda pinchada

El problema de la rueda pinchada consiste en determinar los pasos a realizar para cambiar una rueda pinchada por una rueda de repuesto que se encuentra en el maletero, guardando finalmente la rueda pinchada en el maletero, para poder continuar el viaje.

Para determinar completamente el estado en que se encuentra el problema, basta conocer dónde se encuentran la rueda pinchada y la rueda de repuesto. Usaremos entonces en esta representación el conjunto de hechos

EN(RUEDA-REPUESTO, MALETERO), EN(RUEDA-PINCHADA, MALETERO),  
EN(RUEDA-REPUESTO, SUELO), EN(RUEDA-PINCHADA, SUELO),  
EN(RUEDA-REPUESTO, EJE), EN(RUEDA-PINCHADA, EJE)

Para poder aplicar ciertas acciones, también será necesario determinar si una cierta localización no está ocupada por una rueda, por lo que también incluimos los hechos

NO-EN(RUEDA-REPUESTO, MALETERO), NO-EN(RUEDA-PINCHADA, MALETERO),  
NO-EN(RUEDA-REPUESTO, SUELO), NO-EN(RUEDA-PINCHADA, SUELO),  
NO-EN(RUEDA-REPUESTO, EJE), NO-EN(RUEDA-PINCHADA, EJE)

Al definir las posibles acciones, debemos asegurarnos de establecer que cada una de las dos ruedas esté siempre en una única localización y de evitar que las dos ruedas estén a la vez en el eje o en el maletero (las dos en el suelo sí se admitiría).

Analizando el problema de manera abstracta, se llega a la conclusión de que para cambiar de sitio la rueda pinchada las posibles acciones a realizar serían

- QUITAR-PINCHADA: esta acción quitaría la rueda pinchada y la dejaría en el suelo. Por lo tanto, el requisito para poder aplicarla es EN(RUEDA-PINCHADA, EJE). El efecto de la acción sería añadir EN(RUEDA-PINCHADA, SUELO), pero también se debe eliminar EN(RUEDA-PINCHADA, EJE) del estado, para no representar que la rueda pinchada está a la vez en el eje y en el suelo. Además, también se debe eliminar NO-EN(RUEDA-PINCHADA, SUELO), para no representar que la rueda pinchada está y no está a la vez en el suelo, y se debe añadir NO-EN(RUEDA-PINCHADA, EJE), para no representar que la rueda pinchada ni está ni no está en el eje.
- GUARDAR-PINCHADA: esta acción guardaría la rueda pinchada en el maletero. Para poder aplicar esta acción, la rueda pinchada debe estar en el suelo, EN(RUEDA-PINCHADA, SUELO), pero también el maletero debe estar vacío, NO-EN(RUEDA-REPUESTO, MALETERO). El efecto de la acción sería añadir EN(RUEDA-PINCHADA, MALETERO), pero también se debe eliminar EN(RUEDA-PINCHADA, SUELO) del estado, para no representar que la rueda pinchada está a la vez en el maletero y en el suelo. Igual que para la acción anterior, también se debe eliminar NO-EN(RUEDA-PINCHADA, MALETERO) y añadir NO-EN(RUEDA-PINCHADA, SUELO).

y, de manera análoga, para cambiar de sitio la rueda de repuesto las posibles acciones a realizar serían

- SACAR-REPUESTO, aplicable cuando  $EN(RUEDA-REPUESTO, MALETERO)$  y que añade  $EN(RUEDA-REPUESTO, SUELO)$  y  $NO-EN(RUEDA-REPUESTO, MALETERO)$  y elimina  $EN(RUEDA-REPUESTO, MALETERO)$  y  $NO-EN(RUEDA-REPUESTO, SUELO)$  al aplicarla.
- PONER-REPUESTO, aplicable cuando  $EN(RUEDA-REPUESTO, SUELO)$  y también  $NO-EN(RUEDA-PINCHADA, EJE)$  y que añade  $EN(RUEDA-REPUESTO, EJE)$  y  $NO-EN(RUEDA-REPUESTO, SUELO)$  y elimina  $EN(RUEDA-REPUESTO, SUELO)$  y  $NO-EN(RUEDA-REPUESTO, EJE)$  al aplicarla.

De esta manera tendríamos definido el dominio de planificación. Para definir el problema concreto habría que especificar además un estado inicial, que en este ejemplo sería aquel con la rueda pinchada en el eje y la de repuesto en el maletero, y un objetivo, que en este ejemplo es que la rueda de repuesto esté en el eje y la pinchada en el maletero.

En resumen, una representación STRIPS del problema de la rueda pinchada sería:

### Ejemplo: El problema de la rueda pinchada

#### Hechos:

$EN(RUEDA-REPUESTO, MALETERO)$	$NO-EN(RUEDA-REPUESTO, MALETERO)$
$EN(RUEDA-PINCHADA, MALETERO)$	$NO-EN(RUEDA-PINCHADA, MALETERO)$
$EN(RUEDA-REPUESTO, SUELO)$	$NO-EN(RUEDA-REPUESTO, SUELO)$
$EN(RUEDA-PINCHADA, SUELO)$	$NO-EN(RUEDA-PINCHADA, SUELO)$
$EN(RUEDA-REPUESTO, EJE)$	$NO-EN(RUEDA-REPUESTO, EJE)$
$EN(RUEDA-PINCHADA, EJE)$	$NO-EN(RUEDA-PINCHADA, EJE)$

#### Acciones:

- QUITAR-PINCHADA:
  - Precondiciones:  $EN(RUEDA-PINCHADA, EJE)$
  - Lista de adición:  $EN(RUEDA-PINCHADA, SUELO)$   
 $NO-EN(RUEDA-PINCHADA, EJE)$
  - Lista de borrado:  $EN(RUEDA-PINCHADA, EJE)$   
 $NO-EN(RUEDA-PINCHADA, SUELO)$
- GUARDAR-PINCHADA:
  - Precondiciones:  $EN(RUEDA-PINCHADA, SUELO)$   
 $NO-EN(RUEDA-REPUESTO, MALETERO)$
  - Lista de adición:  $EN(RUEDA-PINCHADA, MALETERO)$   
 $NO-EN(RUEDA-PINCHADA, SUELO)$
  - Lista de borrado:  $EN(RUEDA-PINCHADA, SUELO)$   
 $NO-EN(RUEDA-PINCHADA, MALETERO)$

### ■ SACAR-REPUESTO:

- Precondiciones: EN(RUEDA-REPUESTO, MALETERO)
- Lista de adición: EN(RUEDA-REPUESTO, SUELO)  
NO-EN(RUEDA-REPUESTO, MALETERO)
- Lista de borrado: EN(RUEDA-REPUESTO, MALETERO)  
NO-EN(RUEDA-REPUESTO, SUELO)

### ■ PONER-REPUESTO:

- Precondiciones: EN(RUEDA-REPUESTO, SUELO)  
NO-EN(RUEDA-PINCHADA, EJE)
- Lista de adición: EN(RUEDA-REPUESTO, EJE)  
NO-EN(RUEDA-REPUESTO, SUELO)
- Lista de borrado: EN(RUEDA-REPUESTO, SUELO)  
NO-EN(RUEDA-REPUESTO, EJE)

#### Estado inicial:

EN(RUEDA-REPUESTO, MALETERO) NO-EN(RUEDA-PINCHADA, MALETERO)  
 NO-EN(RUEDA-REPUESTO, SUELO) NO-EN(RUEDA-PINCHADA, SUELO)  
 NO-EN(RUEDA-REPUESTO, EJE) EN(RUEDA-PINCHADA, EJE)

#### Objetivo:

EN(RUEDA-REPUESTO, EJE) EN(RUEDA-PINCHADA, MALETERO)

## El mundo de los bloques

A continuación se explica el desarrollo de una posible manera de representar el dominio de planificación del mundo de los bloques.

Para determinar el estado en que se encuentra el problema necesitamos conocer dónde se encuentra cada bloque. Usaremos entonces los predicados

- SOBRELAMESA: predicado unario que se usará para representar que un bloque está sobre la mesa.
- SOBRE: predicado binario que se usará para representar que un bloque está sobre otro bloque.
- AGARRADO: predicado unario que se usará para representar que un bloque está cogido por el brazo robótico.

de tal forma que, si asumimos que los bloques son A, B, C, ..., el conjunto de hechos es

$$\begin{aligned} &\{\text{SOBRELAMESA}(x) \mid x = A, B, C, \dots\} \cup \\ &\{\text{SOBRE}(x, y) \mid x, y = A, B, C, \dots\} \cup \\ &\{\text{AGARRADO}(x) \mid x = A, B, C, \dots\} \end{aligned}$$

Analizando el problema de manera abstracta, se llega a la conclusión de que las acciones posibles son que el brazo coja o que suelte un bloque. Sin embargo, el hecho de que usemos predicados distintos para representar si un bloque se encuentra sobre la mesa o sobre otro bloque nos obliga a desdoblar las acciones en función de esas dos posibilidades. Ejemplos de acciones serían las siguientes:

- AGARRAR(A): con esta acción el brazo robótico cogería el bloque A cuando este está sobre la mesa. Esta debe ser, por tanto, una precondition de la acción. Hay dos precondiciones adicionales que deben cumplirse y que nos obligan a introducir nuevos predicados, ya que con los actuales es complejo recuperar la información que se necesita:
  - En primer lugar, puesto que el brazo robótico solo puede coger un bloque cada vez, para poder coger el bloque A no debe tener ningún bloque cogido. En lugar de expresar esto añadiendo NO-AGARRADO(A), NO-AGARRADO(B), ..., como precondiciones, es más simple incorporar el hecho BRAZOLIBRE( ) a la representación.
  - En segundo lugar, el brazo robótico no podrá coger el bloque A si hay algún bloque encima de él. Igual que antes, en lugar de añadir como precondiciones NO-SOBRE(B, A), NO-SOBRE(C, A), ..., es más simple incorporar el hecho DESPEJADO(A) a la representación.

El efecto de la acción sería que el bloque A pasaría de estar sobre la mesa a estar agarrado por el brazo robótico, pero también hay que eliminar DESPEJADO(A), pues dejaría de tener sentido en la nueva situación, y BRAZOLIBRE( ), ya que ahora el brazo robótico tendría cogido un bloque.

De manera análoga se definirían las acciones AGARRAR(B), AGARRAR(C), etc.

- BAJAR(A): con esta acción el brazo robótico dejaría el bloque A sobre la mesa. La precondition para poder aplicar esta acción es, evidentemente, que el brazo robótico tenga agarrado el bloque A.

Los efectos de la acción serían que el brazo robótico dejaría de tener agarrado el bloque A, pasando este a estar sobre la mesa y sin ningún bloque encima.

De manera análoga se definirían las acciones BAJAR(B), BAJAR(C), etc.

- DESAPILAR(A, B): con esta acción el brazo robótico cogería el bloque A cuando este se encuentra sobre el bloque B. Esta, junto con que sobre el bloque A no haya ningún bloque y que el brazo robótico no tenga agarrado ningún bloque, serían las precondiciones.

Los efectos directos de la acción serían que el bloque A pasaría de estar sobre el bloque B a estar agarrado por el brazo robótico, que ya no estaría libre. Pero también el bloque B pasaría a estar despejado, porque ya no tendría el bloque A encima. Para este último, sin embargo, ya no tendría sentido decir que está despejado.

De manera análoga se definirían las acciones DESAPILAR(A, C), DESAPILAR(B, A), etc.



- **APILAR(A, B)**: con esta acción el brazo robótico dejaría el bloque A sobre el bloque B. Una precondition para poder aplicar esta acción es, evidentemente, que el brazo robótico tenga agarrado el bloque A, pero también que el bloque B no tenga ya otro bloque encima.

Los efectos directos de la acción serían que el brazo robótico dejaría de tener agarrado el bloque A, pasando este a estar sobre el bloque B y sin ningún bloque encima. El bloque B dejaría de estar despejado, ya que ahora tendría el bloque A encima.

De manera análoga, se definirían las acciones **APILAR(A, C)**, **APILAR(B, A)**, etc.

Obsérvese que si el dominio del problema incluyera  $n$  bloques, entonces habría que definir  $2n + 2n^2$  acciones en total. Es posible, sin embargo, simplificar la representación del dominio de planificación mediante el uso de esquemas de acciones: cada uno de ellos definirá un conjunto de acciones empleando variables para representar subconjuntos de objetos.

Así, podríamos representar el dominio del mundo de los bloques como sigue:

### Ejemplo: El mundo de los bloques

*Hechos:*

$\{ \text{SOBRELAMESA}(x) \mid x = A, B, C, \dots \} \cup$   
 $\{ \text{SOBRE}(x, y) \mid x, y = A, B, C, \dots \} \cup$   
 $\{ \text{AGARRADO}(x) \mid x = A, B, C, \dots \} \cup$   
 $\{ \text{BRAZOLIBRE}() \} \cup$   
 $\{ \text{DESPEJADO}(x) \mid x = A, B, C, \dots \}$

*Esquemas de acciones:*

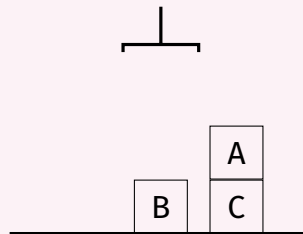
- **AGARRAR( $x$ )**, para  $x = A, B, C, \dots$ :
  - Precondiciones: **SOBRELAMESA( $x$ )**  
**BRAZOLIBRE()**  
**DESPEJADO( $x$ )**
  - Lista de adición: **AGARRADO( $x$ )**
  - Lista de borrado: **SOBRELAMESA( $x$ )**  
**DESPEJADO( $x$ )**  
**BRAZOLIBRE()**
- **BAJAR( $x$ )**, para  $x = A, B, C, \dots$ :
  - Precondiciones: **AGARRADO( $x$ )**
  - Lista de adición: **SOBRELAMESA( $x$ )**  
**DESPEJADO( $x$ )**  
**BRAZOLIBRE()**
  - Lista de borrado: **AGARRADO( $x$ )**
- **DESAPILAR( $x, y$ )**, para  $x, y = A, B, C, \dots$ :

- Precondiciones: SOBRE( $x, y$ )  
DESPEJADO( $x$ )  
BRAZOLIBRE( )
- Lista de adición: AGARRADO( $x$ )  
DESPEJADO( $y$ )
- Lista de borrado: SOBRE( $x, y$ )  
DESPEJADO( $x$ )  
BRAZOLIBRE( )
- APILAR( $x, y$ ), para  $x, y = A, B, C, \dots$ :
  - Precondiciones: AGARRADO( $x$ )  
DESPEJADO( $y$ )
  - Lista de adición: SOBRE( $x, y$ )  
DESPEJADO( $x$ )  
BRAZOLIBRE( )
  - Lista de borrado: AGARRADO( $x$ )  
DESPEJADO( $y$ )

Para definir un problema concreto en este dominio hay que especificar un estado inicial y un conjunto de estados objetivos.

### Ejemplo: Estado inicial y objetivo para el mundo de los bloques

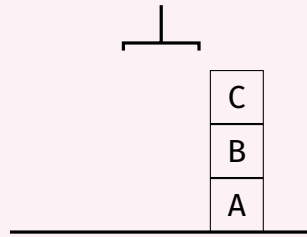
En un dominio del mundo de los bloques que solo contuviera los bloques A, B y C, para establecer como estado inicial el estado



bastaría usar la siguiente representación:

SOBRE(A, C)  
DESPEJADO(A)  
SOBRELAMESA(B)  
DESPEJADO(B)  
SOBRELAMESA(C)  
BRAZOLIBRE( )

Para establecer como único estado objetivo el estado



bastaría usar la siguiente representación para el objetivo:

SOBRELAMESA(A)  
 SOBRE(B, A)  
 SOBRE(C, B)

Por el contrario, para establecer como estados objetivos los dos estados



se podría usar la siguiente representación para el objetivo:

SOBRELAMESA(A)  
 SOBRE(C, A)

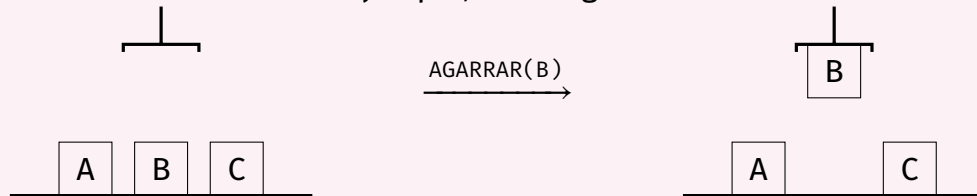
PDDL (*Planning Domain Definition Language*) es un lenguaje que implementa en la práctica el formalismo STRIPS, especificando una sintaxis estricta para la escritura de problemas de planificación en ficheros de texto llano. Este lenguaje se ha convertido en el formato de entrada *de facto* usado por todas las implementaciones de planificadores. PDDL ha evolucionado a lo largo de su historia, ampliando el formalismo STRIPS con diversas extensiones, como la posibilidad de usar precondiciones negadas y cuantificadas, efectos condicionales, variables numéricas, etc.

## Resolución de problemas de planificación

Una de las posibles maneras de abordar la resolución de un problema de planificación clásica, es decir, de encontrar un plan solución del problema, es mediante una búsqueda en el espacio de estados del problema. Esta consiste en aplicar un algoritmo de búsqueda de caminos en el grafo dirigido cuyos vértices son los posibles estados del problema y cuyos arcos conectan estados para los que existe una acción que transforma uno de ellos en el otro. Una solución al problema se obtendría a partir de cualquier camino desde el estado inicial hasta algún estado objetivo.

### Ejemplo: Grafo del mundo de los bloques

En el mundo de los bloques con tres bloques A, B y C habría que buscar, en un grafo con 22 vértices (uno por cada posible estado del problema), un camino desde el estado que representa la disposición inicial de los bloques hasta un estado que represente alguna de las disposiciones finales deseadas. En ese grafo, dos estados estarían conectados si alguna acción del brazo robótico (coger o soltar un bloque) transformara uno en otro. Por ejemplo, en ese grafo nos encontraríamos el arco:



### Algoritmos de búsqueda de caminos en grafos

Es habitual que, para resolver un problema de planificación, los algoritmos de búsqueda construyan el grafo asociado de manera implícita. El esquema general es disponer de una estructura de nodos cerrados, que son aquellos nodos del grafo ya expandidos, y una estructura de nodos abiertos, que son aquellos nodos del grafo ya generados, pero aún no expandidos. Comenzando por el nodo correspondiente al estado inicial, el comportamiento genérico de estos algoritmos consiste en iterar el proceso de seleccionar un nodo abierto, expandirlo generando sus nodos sucesores y cerrarlo una vez expandido, construyendo lo que se conoce como árbol de búsqueda. Las distintas maneras de seleccionar el nodo abierto y, al cerrar este, actualizar las estructuras de nodos abiertos y cerrados, dan lugar a distintos algoritmos de búsqueda.

Cuando la estructura de nodos abiertos es una pila (es decir, una estructura LIFO —*Last In, First Out*—, en la que se selecciona el último nodo introducido) se obtiene un algoritmo de búsqueda en profundidad. Cuando la estructura de nodos abiertos es una cola (es decir, una estructura FIFO —*First In, First Out*—, en la que se selecciona el primer nodo introducido) se obtiene un algoritmo de búsqueda en anchura.

- 1 Cerrados  $\leftarrow \emptyset$
- 2 Abiertos  $\leftarrow \{s_I\}$
- 3 **Mientras** Abiertos  $\neq \emptyset$  **hacer**
- 4     (Búsqueda en profundidad) Extraer el último nodo  $s$  de Abiertos  
       (Búsqueda en anchura) Extraer el primer nodo  $s$  de Abiertos
- 5     Insertar  $s$  en Cerrados
- 6     **Si**  $s \in S_G$  **entonces**
- 7         **Devolver** el camino (plan) de  $s_I$  a  $s$
- 8     **Para cada**  $a$  acción aplicable a  $s$  **hacer**
- 9          $s' \leftarrow$  sucesor de  $s$  obtenido al aplicar  $a$
- 10        **Si**  $s'$  no está ni en Abiertos ni en Cerrados **entonces**
- 11           Padre( $s'$ )  $\leftarrow s$
- 12           Insertar  $s'$  en Abiertos
- 13 **Devolver** Sin solución

Cuando las acciones del problema de planificación (y, por tanto, los arcos del grafo asociado) tienen asociado un coste  $c_a(s, s')$  (no negativo), el algoritmo de Dijkstra calcula una plan óptimo, es decir, un camino en el grafo desde el nodo inicial hasta un nodo objetivo, con el menor coste posible. Este algoritmo se obtiene usando como estructura de nodos abiertos una cola de prioridades en la que a cada nodo  $s$  se le asocia un valor  $f(s)$  que acota superiormente el mínimo coste de un camino desde el nodo inicial  $s_I$  hasta  $s$  y que es disminuido sucesivamente hasta que coincide con ese coste mínimo.

```

1   $f(s) \leftarrow \begin{cases} 0 & \text{para } s = s_I \\ +\infty & \text{para } s \neq s_I \end{cases}$ 
2  Cerrados  $\leftarrow \emptyset$ 
3  Abiertos  $\leftarrow \{s_I\}$ 
4  Mientras Abiertos  $\neq \emptyset$  hacer
5      Extraer el nodo  $s$  de Abiertos con menor valor  $f(s)$ 
6      Insertar  $s$  en Cerrados
7      Si  $s \in S_G$  entonces
8          Devolver el camino (plan) de  $s_I$  a  $s$ 
9      Para cada  $a$  acción aplicable a  $s$  hacer
10          $s' \leftarrow$  sucesor de  $s$  obtenido al aplicar  $a$ 
11         Si  $s'$  está en Abiertos y  $f(s) + c_a(s, s') < f(s')$  entonces
12             Padre( $s'$ )  $\leftarrow s$ 
13              $f(s') \leftarrow f(s) + c_a(s, s')$ 
14         Si no si  $s'$  no está en Abiertos ni en Cerrados entonces
15             Padre( $s'$ )  $\leftarrow s$ 
16              $f(s') \leftarrow f(s) + c_a(s, s')$ 
17             Insertar  $s'$  en Abiertos
18 Devolver Sin solución

```

Una función heurística es una función que para cada estado del problema estima el coste de un plan óptimo desde ese estado hasta el objetivo. El algoritmo  $A^*$  es capaz de incorporar esa información para guiar la búsqueda, haciéndola más eficiente. Este algoritmo se obtiene usando como estructura de nodos abiertos una cola de prioridades en la que a cada nodo  $s$  se le asocia el valor  $g(s) + h(s)$ , donde  $g(s)$  es el valor (conocido) del camino óptimo actual desde el nodo inicial  $s_I$  hasta  $s$  y  $h(s)$  es el coste mínimo (estimado) desde  $s$  hasta el objetivo.

```

1   $g(s) \leftarrow \begin{cases} 0 & \text{para } s = s_I \\ +\infty & \text{para } s \neq s_I \end{cases}$ 
2   $f(s) \leftarrow \begin{cases} h(s) & \text{para } s = s_I \\ +\infty & \text{para } s \neq s_I \end{cases}$ 
3  Cerrados  $\leftarrow \emptyset$ 
4  Abiertos  $\leftarrow \{s_I\}$ 
5  Mientras Abiertos  $\neq \emptyset$  hacer
6      Extraer el nodo  $s$  de Abiertos con menor valor  $f(s)$ 
7      Insertar  $s$  en Cerrados
8      Si  $s \in S_G$  entonces
9          Devolver el camino (plan) de  $s_I$  a  $s$ 
10     Para cada  $a$  acción aplicable a  $s$  hacer
11          $s' \leftarrow$  sucesor de  $s$  obtenido al aplicar  $a$ 
12         Si  $s'$  está en Abiertos y  $g(s) + c_a(s, s') < g(s')$  entonces
13             Padre( $s'$ )  $\leftarrow s$ 
14              $g(s') \leftarrow g(s) + c_a(s, s')$ 
15              $f(s') \leftarrow g(s') + h(s')$ 
16         Si no si  $s'$  está en Cerrados y  $g(s) + c_a(s, s') < g(s')$  entonces
17             Padre( $s'$ )  $\leftarrow s$ 
18              $g(s') \leftarrow g(s) + c_a(s, s')$ 
19              $f(s') \leftarrow g(s') + h(s')$ 
20             Eliminar  $s'$  de Cerrados
21             Insertar  $s'$  en Abiertos
22         Si no si  $s'$  no está en Abiertos ni en Cerrados entonces
23             Padre( $s'$ )  $\leftarrow s$ 
24              $g(s') \leftarrow g(s) + c_a(s, s')$ 
25              $f(s') \leftarrow g(s') + h(s')$ 
26             Insertar  $s'$  en Abiertos
27 Devolver Sin solución

```

Una función heurística  $h$  es:

- **Segura**, si  $h(s) = +\infty$  implica que no hay camino desde  $s$  hasta el objetivo. Es decir, una heurística segura es aquella que, para cualquier estado, si asume que es el estado inicial y estima que el problema no tiene solución, entonces no está equivocada.
- **Consciente del objetivo**, si  $h(s) = 0$  para cualquier estado  $s \in S_G$ . Es decir, una heurística consciente del objetivo no estima un coste positivo de alcanzar el objetivo desde un estado objetivo.
- **Admisible**, si para cualquier estado el coste mínimo estimado desde ese estado hasta el objetivo es menor o igual que el coste mínimo real. Es decir, una heurística admisible es aquella que siempre subestima el coste de llegar hasta el objetivo.
- **Consistente**, si para cualesquiera estado  $s$ , acción  $a$  aplicable a  $s$  y estado resultante  $s'$  se tiene que  $h(s) \leq h(s') + c_a(s, s')$ . Es decir, una heurística consistente es aquella

que, al aplicar una acción, no disminuye su estimación de llegar al objetivo en mayor medida que el coste de aplicar la acción.

Una heurística admisible es una heurística segura y consciente del objetivo. Una heurística consistente y consciente del objetivo es una heurística admisible.

Si usa una heurística admisible, entonces el algoritmo  $A^*$  es completo (si el problema tiene solución, entonces el algoritmo encuentra un plan solución) y óptimo (si el problema tiene solución, entonces el plan proporcionado por el algoritmo tiene coste mínimo).

## Heurísticas independientes del dominio

La cantidad de estados de un problema de planificación clásica es, en general, demasiado elevado para que se pueda resolver mediante una búsqueda ciega, como pueden ser una búsqueda en anchura o profundidad.

### Ejemplo: Cantidad de posibles estados en el mundo de los bloques

La siguiente tabla muestra cómo crece de forma exponencial la cantidad de posibles estados existentes en el dominio del mundo de los bloques en función del número de bloques:

Bloques	Estados	Bloques	Estados
1	2	6	7057
2	5	7	65 990
3	22	8	695 417
4	125	9	8 145 730
5	866	10	104 906 621

Es necesario recurrir entonces a algoritmos como  $A^*$ , en los que se guía la búsqueda mediante alguna función heurística. Una vía para definir estas funciones heurísticas de forma «automática» es mediante el uso de la técnica de relajación del problema, que consiste en eliminar restricciones del problema de tal manera que sea entonces posible determinar el coste exacto de una solución óptima.

En el caso concreto de los problemas de planificación, un método habitual de relajación del problema es la relajación del borrado: se descartan las listas de borrado de las acciones, asumiendo así que su ejecución solo puede dar lugar a que se cumplan nuevos hechos, pero no a que haya hechos que dejen de cumplirse. Se elimina así la restricción de que en un estado no puedan cumplirse simultáneamente hechos incompatibles (como, por ejemplo, que tanto la rueda de repuesto como la pinchada estén ambas en el maletero o en el eje).

Dado un problema de planificación  $P = (S, A, s_I, S_G)$ :

- Si  $a \in A$  es una acción con precondiciones  $\text{pre}(a)$ , lista de adición  $\text{add}(a)$  y lista de borrado  $\text{del}(a)$ , entonces la correspondiente acción relajada es la acción  $a^+$  con las mismas precondiciones y lista de adición que  $a$ , pero con el conjunto vacío como lista de borrado.

- El problema de planificación relajado correspondiente a  $P$  es  $P^+ = (S, A^+, s_I, S_G)$ , donde  $A^+$  es el conjunto de las acciones relajadas obtenidas a partir de todas las acciones en  $A$ .
- Si  $s \in S$  es un estado del problema, entonces  $P_s = (S, A, s, S_G)$  es el problema que se obtiene a partir de  $P$  estableciendo  $s$  como estado inicial. Un plan relajado para  $s$  es un plan solución de  $P_s^+$ . Un plan relajado para  $P$  es un plan relajado para  $s_I$ .
- Si  $\langle a_1, \dots, a_n \rangle$  es un plan solución de  $P_s$ , entonces  $\langle a_1^+, \dots, a_n^+ \rangle$  es un plan relajado para  $s$ . Por lo tanto, si  $P_s$  es resoluble, entonces  $P_s^+$  también lo es.

### Ejemplo: Plan relajado para el problema del camión y el paquete

Consideremos el problema de planificación en el que un camión debe recoger un paquete en una determinada localización y dejarlo en otra localización. Suponiendo cuatro localizaciones distintas A, B, C y D, una posible representación del problema usando el formalismo STRIPS sería:

*Hechos:*

$$\begin{aligned} &\{ \text{CAMIÓN-EN}(x), \text{PAQUETE-EN}(x) \mid x = A, B, C, D \} \cup \\ &\{ \text{PAQUETE-EN}(\text{CAMIÓN}) \} \cup \\ &\{ \text{CONECTADAS}(A, B), \text{CONECTADAS}(B, A), \text{CONECTADAS}(B, C), \\ &\quad \text{CONECTADAS}(C, B), \text{CONECTADAS}(C, D), \text{CONECTADAS}(D, C) \} \end{aligned}$$

Nota: los hechos que especifican qué localizaciones están conectadas son fijos y están incluidos en todos los estados, por lo que en lo que sigue no se indicarán explícitamente.

*Esquemas de acciones:*

- $\text{IR}(x, y)$ , para  $x, y = A, B, C, D$ :
  - Precondiciones:  $\text{CONECTADAS}(x, y)$   
 $\text{CAMIÓN-EN}(x)$
  - Lista de adición:  $\text{CAMIÓN-EN}(y)$
  - Lista de borrado:  $\text{CAMIÓN-EN}(x)$
- $\text{CARGAR-PAQUETE}(x)$ , para  $x = A, B, C, D$ :
  - Precondiciones:  $\text{CAMIÓN-EN}(x)$   
 $\text{PAQUETE-EN}(x)$
  - Lista de adición:  $\text{PAQUETE-EN}(\text{CAMIÓN})$
  - Lista de borrado:  $\text{PAQUETE-EN}(x)$
- $\text{DESCARGAR-PAQUETE}(x)$ , para  $x = A, B, C, D$ :
  - Precondiciones:  $\text{CAMIÓN-EN}(x)$   
 $\text{PAQUETE-EN}(\text{CAMIÓN})$



- Lista de adición:  $\text{PAQUETE-EN}(x)$
- Lista de borrado:  $\text{PAQUETE-EN}(\text{CAMIÓN})$

*Estado inicial:*

$\text{CAMIÓN-EN}(A)$   
 $\text{PAQUETE-EN}(C)$

*Objetivo:*

$\text{CAMIÓN-EN}(A)$   
 $\text{PAQUETE-EN}(D)$

Las acciones relajadas son:

*Esquemas de acciones relajadas:*

- $\text{IR}(x, y)^+$ , para  $x, y = A, B, C, D$ :
  - Precondiciones:  $\text{CONECTADAS}(x, y)$   
 $\text{CAMIÓN-EN}(x)$
  - Lista de adición:  $\text{CAMIÓN-EN}(y)$
- $\text{CARGAR-PAQUETE}(x)^+$ , para  $x = A, B, C, D$ :
  - Precondiciones:  $\text{CAMIÓN-EN}(x)$   
 $\text{PAQUETE-EN}(x)$
  - Lista de adición:  $\text{PAQUETE-EN}(\text{CAMIÓN})$
- $\text{DESCARGAR-PAQUETE}(x)^+$ , para  $x = A, B, C, D$ :
  - Precondiciones:  $\text{CAMIÓN-EN}(x)$   
 $\text{PAQUETE-EN}(\text{CAMIÓN})$
  - Lista de adición:  $\text{PAQUETE-EN}(x)$

Un plan relajado para el problema sería

$\langle \text{IR}(A, B)^+, \text{IR}(B, C)^+, \text{CARGAR}(C)^+, \text{IR}(C, D)^+, \text{DESCARGAR}(D)^+ \rangle$

que daría lugar a la siguiente secuencia de estados del problema relajado:

- Estado inicial:  $\text{CAMIÓN-EN}(A)$   $\text{PAQUETE-EN}(C)$
- Tras aplicar la acción  $\text{IR}(A, B)^+$ :  $\text{CAMIÓN-EN}(A)$   $\text{PAQUETE-EN}(C)$   
 $\text{CAMIÓN-EN}(B)$
- Tras aplicar la acción  $\text{IR}(B, C)^+$ :  $\text{CAMIÓN-EN}(A)$   $\text{PAQUETE-EN}(C)$   
 $\text{CAMIÓN-EN}(B)$   
 $\text{CAMIÓN-EN}(C)$

- Tras aplicar la acción CARGAR(C)<sup>+</sup>: CAMIÓN-EN(A) PAQUETE-EN(C)  
CAMIÓN-EN(B) PAQUETE-EN(CAMIÓN)  
CAMIÓN-EN(C)
- Tras aplicar la acción IR(C, D)<sup>+</sup>: CAMIÓN-EN(A) PAQUETE-EN(C)  
CAMIÓN-EN(B) PAQUETE-EN(CAMIÓN)  
CAMIÓN-EN(C)  
CAMIÓN-EN(D)
- Tras aplicar la acción DESCARGAR(D)<sup>+</sup>: CAMIÓN-EN(A) PAQUETE-EN(C)  
CAMIÓN-EN(B) PAQUETE-EN(CAMIÓN)  
CAMIÓN-EN(C) PAQUETE-EN(D)  
CAMIÓN-EN(D)

Obsérvese como en el problema relajado no es necesario aplicar acciones para que el camión vuelva desde D hasta A, ya que los estados conservan la información de todas las localizaciones por las que ha pasado.

Un plan relajado para  $s$  resuelve el problema  $P_s$  partiendo del estado  $s$  y simplemente añadiendo cada vez nuevos hechos que se cumplen, sin tener en cuenta las posibles interacciones entre las acciones. Esta consideración nos lleva al siguiente algoritmo voraz de cálculo de planes relajados para  $s$ :

```

1   $s^+ \leftarrow s$ 
2   $\alpha^+ \leftarrow \langle \rangle$ 
3  Mientras  $s_G \not\subseteq s^+$  hacer
4      Si existe  $a \in A$  tal que  $\text{pre}(a) \subseteq s^+$  y  $\text{add}(a) \not\subseteq s^+$  entonces
5          Elegir una tal acción  $a$ 
6          Añadir  $\text{add}(a)$  a  $s^+$ 
7          Añadir  $a^+$  al final de  $\alpha^+$ 
8      Si no entonces
9          Devolver no existe plan relajado para  $s$ 
10 Devolver  $\alpha^+$ 

```

#### Ejemplo: Cálculo de plan relajado para el problema del camión y el paquete

- Inicialización:

$$s^+ \leftarrow \text{CAMIÓN-EN(A) PAQUETE-EN(C)}$$

$$\alpha^+ \leftarrow \langle \rangle$$

- Acciones elegibles: IR(A, B)

Se elige la acción IR(A, B):

$$s^+ \leftarrow \text{CAMIÓN-EN(A)} \quad \text{CAMIÓN-EN(B)} \quad \text{PAQUETE-EN(C)}$$

$$\alpha^+ \leftarrow \langle \text{IR(A, B)}^+ \rangle$$

- Acciones elegibles: IR(B, C) (la acción IR(B, A) también es aplicable, pero no proporciona hechos nuevos)

Se elige la acción IR(B, C):

$$s^+ \leftarrow \text{CAMIÓN-EN(A)} \quad \text{CAMIÓN-EN(B)} \quad \text{CAMIÓN-EN(C)}$$

$$\text{PAQUETE-EN(C)}$$

$$\alpha^+ \leftarrow \langle \text{IR(A, B)}^+, \text{IR(B, C)}^+ \rangle$$

- Acciones elegibles: IR(C, D), CARGAR(C)

Se elige la acción CARGAR(C):

$$s^+ \leftarrow \text{CAMIÓN-EN(A)} \quad \text{CAMIÓN-EN(B)} \quad \text{CAMIÓN-EN(C)}$$

$$\text{PAQUETE-EN(C)} \quad \text{PAQUETE-EN(CAMIÓN)}$$

$$\alpha^+ \leftarrow \langle \text{IR(A, B)}^+, \text{IR(B, C)}^+, \text{CARGAR(C)}^+ \rangle$$

- Acciones elegibles: IR(C, D), DESCARGAR(A), DESCARGAR(B)

Se elige la acción DESCARGAR(A):

$$s^+ \leftarrow \text{CAMIÓN-EN(A)} \quad \text{CAMIÓN-EN(B)} \quad \text{CAMIÓN-EN(C)}$$

$$\text{PAQUETE-EN(C)} \quad \text{PAQUETE-EN(CAMIÓN)} \quad \text{PAQUETE-EN(A)}$$

$$\alpha^+ \leftarrow \langle \text{IR(A, B)}^+, \text{IR(B, C)}^+, \text{CARGAR(C)}^+, \text{DESCARGAR(A)}^+ \rangle$$

- Acciones elegibles: IR(C, D), DESCARGAR(B)

Se elige la acción IR(C, D):

$$s^+ \leftarrow \text{CAMIÓN-EN(A)} \quad \text{CAMIÓN-EN(B)} \quad \text{CAMIÓN-EN(C)}$$

$$\text{PAQUETE-EN(C)} \quad \text{PAQUETE-EN(CAMIÓN)} \quad \text{PAQUETE-EN(A)}$$

$$\text{CAMIÓN-EN(D)}$$

$$\alpha^+ \leftarrow \langle \text{IR(A, B)}^+, \text{IR(B, C)}^+, \text{CARGAR(C)}^+, \text{DESCARGAR(A)}^+, \text{IR(C, D)}^+ \rangle$$

- Acciones elegibles: DESCARGAR(B), DESCARGAR(D)

Se elige la acción DESCARGAR(D):

$$s^+ \leftarrow \text{CAMIÓN-EN(A)} \quad \text{CAMIÓN-EN(B)} \quad \text{CAMIÓN-EN(C)}$$

$$\text{PAQUETE-EN(C)} \quad \text{PAQUETE-EN(CAMIÓN)} \quad \text{PAQUETE-EN(A)}$$

$$\text{CAMIÓN-EN(D)} \quad \text{PAQUETE-EN(D)}$$

$$\alpha^+ \leftarrow \langle \text{IR(A, B)}^+, \text{IR(B, C)}^+, \text{CARGAR(C)}^+, \text{DESCARGAR(A)}^+, \text{IR(C, D)}^+, \text{DESCARGAR(D)}^+ \rangle$$

Se podría definir entonces la siguiente función heurística:

$$h(s) = \begin{cases} n & \text{si } \langle a_1^+, \dots, a_n^+ \rangle \text{ es un plan relajado para } s, \\ +\infty & \text{si no existe plan relajado para } s. \end{cases}$$

Sin embargo, esta heurística no es adecuada:

- El algoritmo de obtención de un plan relajado para  $s$  puede seleccionar acciones irrelevantes (como la acción DESCARGAR(A) en el ejemplo anterior), sobreestimando de esta forma la cantidad de acciones necesarias para alcanzar el objetivo.
- La heurística puede proporcionar estimaciones muy distintas para estados similares, e incluso para un mismo estado al calcularla repetidamente para ese estado.

Es por ello que la función heurística que realmente debe considerarse es

$$h^+(s) = \begin{cases} n & \text{si } \langle a_1^+, \dots, a_n^+ \rangle \text{ es un plan relajado óptimo para } s, \\ +\infty & \text{si no existe plan relajado para } s. \end{cases}$$

donde un plan relajado óptimo para  $s$  es un plan relajado para  $s$  con la menor cantidad posible de acciones.

$h^+$  es una heurística segura, consciente del objetivo y consistente (y, por tanto, admisible), pero su cálculo es computacionalmente costoso. Esto hace que, en la práctica, se consideren funciones heurísticas que aproximen  $h^+$  de una u otra manera.

Una de las maneras más habituales de realizar esa aproximación es considerar independencia entre los objetivos a alcanzar:

- La heurística  $h^{\max}$  asume que, para lograr un conjunto de objetivos, es suficiente con lograr el objetivo más costoso. De esta forma,  $h^{\max}(s) = h^{\max}(s, S_G)$ , donde

$$h^{\max}(s, G) = \begin{cases} 0 & \text{si } G = \{g\} \text{ y } g \in s, \\ \min(1 + h^{\max}(s, \text{pre}(a)) \mid a \in A, g \in \text{add}(a)) & \text{si } G = \{g\} \text{ y } g \notin s, \\ \max(h^{\max}(s, \{g\}) \mid g \in G) & \text{si } |G| > 1. \end{cases}$$

- La heurística  $h^{\text{add}}$  asume que, para lograr un conjunto de objetivos, se debe lograr cada uno de los objetivos por separado. De esta forma,  $h^{\text{add}}(s) = h^{\text{add}}(s, S_G)$ , donde

$$h^{\text{add}}(s, G) = \begin{cases} 0 & \text{si } G = \{g\} \text{ y } g \in s, \\ \min(1 + h^{\text{add}}(s, \text{pre}(a)) \mid a \in A, g \in \text{add}(a)) & \text{si } G = \{g\} \text{ y } g \notin s, \\ \sum_{g \in G} h^{\text{add}}(s, \{g\}) & \text{si } |G| > 1. \end{cases}$$

Para el cálculo de  $h^{\max}(s)$  y  $h^{\text{add}}(s)$  se puede utilizar el siguiente algoritmo de programación dinámica:

```

1  Para cada hecho  $g \in S$  hacer
2      Inicializar  $T_0^s(g)$  como  $\begin{cases} 0 & \text{si } g \in s, \\ +\infty & \text{si } g \notin s. \end{cases}$ 
3   $i \leftarrow 0$ 
4  Repetir
5       $i \leftarrow i + 1$ 
6      Para cada hecho  $g \in S$  hacer
7          Si  $\min(1 + T_{i-1}^s(\text{pre}(a)) \mid a \in A, g \in \text{add}(a)) < T_{i-1}^s(g)$  entonces
8               $T_i^s(g) \leftarrow \min(1 + T_{i-1}^s(\text{pre}(a)) \mid a \in A, g \in \text{add}(a))$ 
9      Si  $T_i^s = T_{i-1}^s$  entonces
10         Devolver  $T_i^s(S_G)$ 

```

donde

$$T_i^s(G) = \begin{cases} \max(T_i^s(g) \mid g \in G) & \text{para } h^{\max}, \\ \sum_{g \in G} T_i^s(g) & \text{para } h^{\text{add}}. \end{cases}$$

### Ejemplo: Cálculo de $h^{\max}(s_1)$ para el problema del camión y el paquete

La siguiente tabla muestra, para la heurística  $h^{\max}$  la evolución de la función  $T^{s_1}$  a lo largo de las iteraciones del algoritmo:

	$T_0^{s_1}$	$T_1^{s_1}$	$T_2^{s_1}$	$T_3^{s_1}$	$T_4^{s_1}$	$T_5^{s_1}$
CAMIÓN-EN(A)	0	0	0	0	0	0
CAMIÓN-EN(B)	$+\infty$	1	1	1	1	1
CAMIÓN-EN(C)	$+\infty$	$+\infty$	2	2	2	2
CAMIÓN-EN(D)	$+\infty$	$+\infty$	$+\infty$	3	3	3
PAQUETE-EN(A)	$+\infty$	$+\infty$	$+\infty$	$+\infty$	4	4
PAQUETE-EN(B)	$+\infty$	$+\infty$	$+\infty$	$+\infty$	4	4
PAQUETE-EN(C)	0	0	0	0	0	0
PAQUETE-EN(D)	$+\infty$	$+\infty$	$+\infty$	$+\infty$	4	4
PAQUETE-EN(CAMIÓN)	$+\infty$	$+\infty$	$+\infty$	3	3	3
CONECTADAS( $x, y$ )	0	0	0	0	0	0

Por ejemplo, para calcular  $T_1^{s_1}(\text{CAMIÓN-EN(B)})$  el algoritmo:

- Determina que las acciones que tienen a CAMIÓN-EN(B) en la lista de adición son IR(A, B) e IR(C, B).
- Las precondiciones de IR(A, B) son CONECTADAS(A, B) y CAMIÓN-EN(A), luego calcula

$$\begin{aligned}
 T_0^{s_1}(\text{pre}(\text{IR(A, B)})) &= \max(T_0^{s_1}(\text{CONECTADAS(A, B)}), T_0^{s_1}(\text{CAMIÓN-EN(A)})) \\
 &= \max(0, 0) \\
 &= 0
 \end{aligned}$$

- Las precondiciones de  $IR(C, B)$  son  $CONECTADAS(C, B)$  y  $CAMIÓN-EN(C)$ , luego calcula

$$\begin{aligned} T_0^{SI}(\text{pre}(IR(C, B))) &= \max(T_0^{SI}(CONECTADAS(C, B)), T_0^{SI}(CAMIÓN-EN(C))) \\ &= \max(0, +\infty) \\ &= +\infty \end{aligned}$$

- Realiza la comparación

$$\begin{aligned} \min(1 + T_0^{SI}(\text{pre}(IR(A, B))), 1 + T_0^{SI}(\text{pre}(IR(C, B)))) &= \min(1 + 0, 1 + \infty) \\ &= 1 \\ &< +\infty \\ &= T_0^{SI}(CAMIÓN-EN(B)) \end{aligned}$$

- Establece  $T_1^{SI}(CAMIÓN-EN(B)) \leftarrow 1$

Finalmente, una vez la función  $T^{SI}$  es estable, se tiene que

$$h^{\max}(s_I) = T^{SI}(S_G) = \max(T^{SI}(CAMIÓN-EN(A)), T^{SI}(PAQUETE-EN(D))) = 4$$

$h^{\max}$  es una heurística segura y consciente del objetivo. Se verifica que  $h^{\max} \leq h^+$ , luego es admisible. Puede, sin embargo, subestimar en demasía el coste real de alcanzar el objetivo.

#### Ejemplo: $h^{\max}$ puede ser demasiado optimista

Si en lugar de transportar un único paquete el problema consistiera en transportar 101 paquetes de la localización C a la localización D,  $h^{\max}(s_I)$  seguiría valiendo 4, ya que el transporte de cada paquete tiene ese coste y la heurística toma el máximo de los costes individuales.

**Ejemplo: Cálculo de  $h^{\text{add}}(s_I)$  para el problema del camión y el paquete**

La siguiente tabla muestra, para la heurística  $h^{\text{add}}$  la evolución de la función  $T^{s_I}$  a lo largo de las iteraciones del algoritmo:

	$T_0^{s_I}$	$T_1^{s_I}$	$T_2^{s_I}$	$T_3^{s_I}$	$T_4^{s_I}$	$T_5^{s_I}$
CAMIÓN-EN(A)	0	0	0	0	0	0
CAMIÓN-EN(B)	$+\infty$	1	1	1	1	1
CAMIÓN-EN(C)	$+\infty$	$+\infty$	2	2	2	2
CAMIÓN-EN(D)	$+\infty$	$+\infty$	$+\infty$	3	3	3
PAQUETE-EN(A)	$+\infty$	$+\infty$	$+\infty$	$+\infty$	4	4
PAQUETE-EN(B)	$+\infty$	$+\infty$	$+\infty$	$+\infty$	5	5
PAQUETE-EN(C)	0	0	0	0	0	0
PAQUETE-EN(D)	$+\infty$	$+\infty$	$+\infty$	$+\infty$	7	7
PAQUETE-EN(CAMIÓN)	$+\infty$	$+\infty$	$+\infty$	3	3	3
CONECTADAS(x, y)	0	0	0	0	0	0

Por ejemplo, para calcular  $T_1^{s_I}(\text{CAMIÓN-EN(B)})$  el algoritmo:

- Determina que las acciones que tienen a CAMIÓN-EN(B) en la lista de adición son IR(A, B) e IR(C, B).
- Las precondiciones de IR(A, B) son CONECTADAS(A, B) y CAMIÓN-EN(A), luego calcula

$$\begin{aligned} T_0^{s_I}(\text{pre}(\text{IR(A, B)})) &= T_0^{s_I}(\text{CONECTADAS(A, B)}) + T_0^{s_I}(\text{CAMIÓN-EN(A)}) \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

- Las precondiciones de IR(C, B) son CONECTADAS(C, B) y CAMIÓN-EN(C), luego calcula

$$\begin{aligned} T_0^{s_I}(\text{pre}(\text{IR(C, B)})) &= T_0^{s_I}(\text{CONECTADAS(C, B)}) + T_0^{s_I}(\text{CAMIÓN-EN(C)}) \\ &= 0 + \infty \\ &= +\infty \end{aligned}$$

- Realiza la comparación

$$\begin{aligned} \min(1 + T_0^{s_I}(\text{pre}(\text{IR(A, B)})), 1 + T_0^{s_I}(\text{pre}(\text{IR(C, B)}))) &= \min(1 + 0, 1 + \infty) \\ &= 1 \\ &< +\infty \\ &= T_0^{s_I}(\text{CAMIÓN-EN(B)}) \end{aligned}$$

- Establece  $T_1^{s_I}(\text{CAMIÓN-EN(B)}) \leftarrow 1$

Finalmente, una vez la función  $T^{s_I}$  es estable, se tiene que

$$h^{\text{add}}(s_I) = T^{s_I}(S_G) = T^{s_I}(\text{CAMIÓN-EN(A)}) + T^{s_I}(\text{PAQUETE-EN(D)}) = 7$$

$h^{\text{add}}$  es una heurística segura y consciente del objetivo. Se verifica que  $h^{\text{add}} \geq h^+$  y es, en general, no admisible. Puede, además, sobreestimar en demasía el coste real de alcanzar el objetivo, ya que ignora los subplanes compartidos entre los objetivos individuales.

**Ejemplo:  $h^{\text{add}}$  puede ser demasiado pesimista**

Si en lugar de transportar un único paquete el problema consistiera en transportar 101 paquetes de la localización C a la localización D,  $h^{\text{add}}(s_I)$  valdría 707, ya que cuenta una y otra vez, para cada paquete, el coste del movimiento del camión.

Es importante entender que los algoritmos de búsqueda de un plan solución trabajan con la especificación original del problema. Es decir, los nodos del árbol de búsqueda son estados del problema y los sucesores se obtienen aplicando las acciones del problema. Es únicamente a la hora de calcular la heurística de un estado cuando se hace uso del problema relajado.

Las funciones heurísticas  $h^+$ ,  $h^{\text{máx}}$  y  $h^{\text{add}}$  se han definido considerando que las acciones tienen coste unitario y, por tanto, un plan óptimo es un plan de longitud mínima. No es difícil generalizar sus definiciones a acciones con coste arbitrario.