



Ejercicios Búsqueda Heurística y de dos agentes
Grado en Robótica. Curso 2021-2022.
Departamento de Electrónica y Computación

1. Ejercicio 1

Aplicar el algoritmo de búsqueda A* sobre la representación del problema de los misioneros y los caníbales extraída de la anterior hoja de problemas. Especifica cuándo un nodo del árbol de búsqueda es expandido y cuándo es sólo generado. A modo de resumen, se describen los siguientes elementos:

1. Un estado es una terna (M_i, C_i, B, M_f, C_f) en la que:
 - $B \in [i, f]$ indica la posición de la barca, por lo que toma el valor i si está en el extremo inicial, o f si está en el final.
 - $M_i, C_i, M_f, C_f \in [0, \dots, 3]$ indican el número de misioneros y caníbales que quedan en el extremo inicial y final del río, respectivamente.
2. El estado inicial se representa como $(3, 3, i, 0, 0)$ y el final como $(0, 0, f, 3, 3)$
3. Se dispone de 5 operadores: Mover1C(x,y), Mover2C(x,y), Mover1M(x,y), Mover2M(x,y), y Mover1M1C(x,y)
4. La heurística a utilizar es $h_2(n)$

Emplear la siguiente versión de A* que evita tener estados repetidos en ABIERTA pero que requiere tener que recorrer ABIERTA y CERRADA por cada sucesor:

ABIERTA=I, CERRADA=Vacío, EXITO=Falso

Hasta que ABIERTA esta vacío O EXITO

Quitar el primer nodo de ABIERTA, N

SI N es Estado-final **ENTONCES** EXITO=Verdadero

SI NO Expandir N y meterlo en CERRADA, generando el conjunto S de sucesores de N

Para cada sucesor s en S

1. Si s no está ni en ABIERTA y CERRADA se inserta en orden en ABIERTA
2. Si está en ABIERTA y la función de evaluación $f()$ de s es mejor, se elimina el que ya estaba en ABIERTA y se introduce s en orden
3. Si está en CERRADA se ignora

SI EXITO Entonces Solución=camino desde N a I a través de los punteros

SI NO Solución=Fracaso

2. Ejercicio 2

Supongamos que tenemos un algoritmo de búsqueda de tipo el mejor primero que utiliza la siguiente función de evaluación: $f(n) = (2 - w) \times g(n) + wh(n)$:

- ¿Qué tipo de búsqueda hace cuando $w = 0$? ¿y cuando $w = 1$? ¿y para $w = 2$?
- ¿Para qué valores de w el algoritmo es admisible?

Ejercicio 3

HARE & HOUNDS (Figura 1) es un juego de estrategia para 2 jugadores: *la presa* y *el cazador*. *La presa* decide los movimientos de la liebre mientras que *el cazador*, decide los movimiento de los perros. *La presa* gana el juego si consigue llegar de un extremo al otro del tablero. En cambio, *el cazador* gana si logra atrapar a la liebre, es decir si se alcanza una situación del juego en la que la liebre no puede realizar ningún movimiento. En cada turno cada jugador puede mover sólo una ficha y siempre a una de las casillas adyacentes. Las fichas *del cazador* no pueden retroceder. El primer turno corresponde siempre *al cazador*. Se pide:

- Indica posibles representaciones de los estados.
- Describe los operadores del juego.
- ¿Qué funciones de evaluación se te ocurren para programar un jugador automático inteligente de HARE & HOUNDS.

En <http://www.appletonline.com/JavaApplets/HoundsAndHare/> puede encontrarse un *applet* para jugar a HARE & HOUNDS.

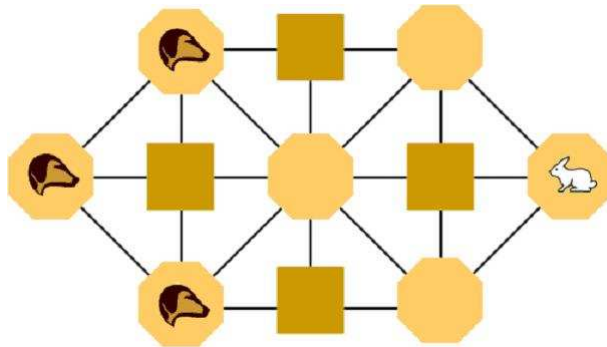


Figura 1: Posición inicial de la liebre y los perros en el juego de HARE & HOUNDS

Ejercicio 4

Se tiene un robot autónomo en una habitación cuadrada de 6x6 casillas (Figura 2). El robot es capaz de realizar desplazamientos verticales y horizontales y reconoce un obstáculo si esta en una de las casillas adyacentes. Suponiendo que el robot utiliza algoritmos de búsqueda para planificar sus movimientos contesta a las siguientes preguntas:

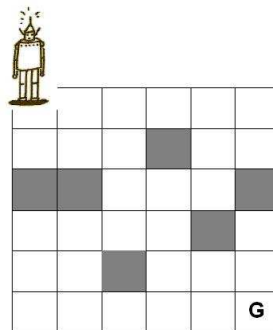


Figura 2: El Robot tiene que ir de una esquina de la habitación a la otra

- ¿Qué heurística podría utilizar el robot para guiarse desde su posición actual hasta el otro extremo de la habitación?

- ¿Qué camino tomaría el robot si su movimiento estuviese determinado por un algoritmo de búsqueda en escalada con la heurística del apartado anterior? Suponer que los estados sucesores se generan aplicando los operadores de desplazamiento en este orden: [derecha, abajo, izquierda, arriba]?
- Y siguiesen el orden:[arriba, izquierda, abajo, derecha] ¿Qué camino tomaría?
- Siguiendo el primer orden de operadores ¿Qué camino tomaría el robot si sus movimientos estuviesen guiados por un algoritmo de búsqueda en haz con $k=3$?

Ejercicio 5

Recorrer el grafo de la figura 3 utilizando los algoritmos de búsqueda heurística: *Escalada*, A^* e IDA^* , teniendo en cuenta los costes indicados en cada arco.

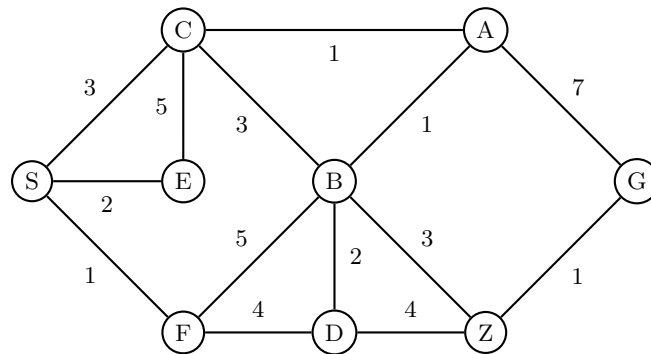


Figura 3: Espacio de estados

En todos los casos, indicar en qué orden se visitan los nodos, distinguiendo nodos generados de nodos expandidos. Tomar como estado inicial el nodo S y como único estado meta el nodo G.

Cada nodo del grafo tiene el valor heurístico descrito en la tabla 1

n	S	A	B	C	D	E	F	G	Z
$h(n)$	8	2	3	5	4	6	6	0	1

Tabla 1: Tabla de valores heurísticos

Ejercicio 6

En algunas aplicaciones reales, el espacio de problemas se suele formular con un único estado inicial, s , y un conjunto arbitrariamente grande de estados finales o meta $\Gamma : \{t_1, t_2, \dots, t_n\}$. Considerando los espacios de problemas formulados de esta manera, y considerando únicamente el caso de espacios de estados no dirigidos, se pide:

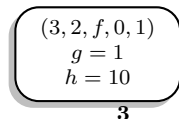
- Sin modificar la función heurística que se esté empleando, ¿qué modificaciones deben hacerse al algoritmo de búsqueda A^* para que encuentre la secuencia óptima de operadores que transforma un único estado inicial, s , en uno cualquiera de varios estados meta $\Gamma : \{t_1, t_2, \dots, t_n\}$?

Solución ejercicio 1

El algoritmo A^* , como todos los algoritmos de *el mejor primero*, se diferencia de sus predecesores, fundamentalmente, en lo siguiente:

- ① Los nodos se almacenan en una lista denominada ABIERTA por su valor $f(n)$ ordenada ascendentemente. De esta manera, el primer nodo de dicha lista es siempre el que tiene el menor valor $f(\cdot)$

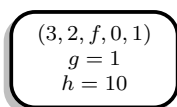
En las figuras que siguen, los nodos en la lista ABIERTA se muestran en el (sub)árbol de búsqueda en una caja sólida, sin relleno como en el siguiente ejemplo:



y debajo se indica, a su derecha, el orden que tiene el nodo en la lista ABIERTA —3 en el ejemplo.

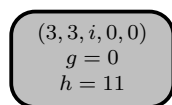
- ② Para evitar la *re-expansión* de nodos, el algoritmo mantiene una lista CERRADA con todos los nodos que han sido anteriormente expandidos.

A continuación, los nodos expandidos se mostrarán con una línea que los rodea más gruesa y sin ningún número debajo de ellos —puesto que los nodos en la lista CERRADA no tienen por qué estar ordenados:



- ③ El algoritmo no se detiene cuando genera eventualmente el nodo final sino cuando va a expandirlo. De acuerdo con el punto ①, esto significa que estando primero es el nodo con menor valor $f(\cdot)$ y, por lo tanto, la solución encontrada será óptima —si la función heurística $h(\cdot)$ es admisible.

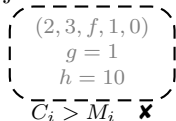
En la solución, los estados inicial y final se han distinguido gráficamente del resto con un nodo relleno. Por ejemplo, el estado inicial es:



Por otra parte, de entre los algoritmos de el mejor primero, el algoritmo A^* se distingue porque emplea la función de evaluación $f(n) = g(n) + h(n)$. En todos los nodos que se mostrarán a continuación, el valor de la función de evaluación se muestra explícitamente indicando el valor de sus componentes $g(n)$ y $h(n)$.

Por lo tanto, en términos generales el algoritmo A^* opera siempre de la misma manera: coge el primer nodo de la lista ABIERTA y, si no es el nodo final, lo expande (introduciéndolo entonces en la lista CERRADA) e inserta en la lista ABIERTA todos los sucesores que no estén en la lista CERRADA. Después de evaluar cada sucesor, ordena toda la lista ABIERTA y así, sucesivamente, hasta que se encuentra el nodo final.

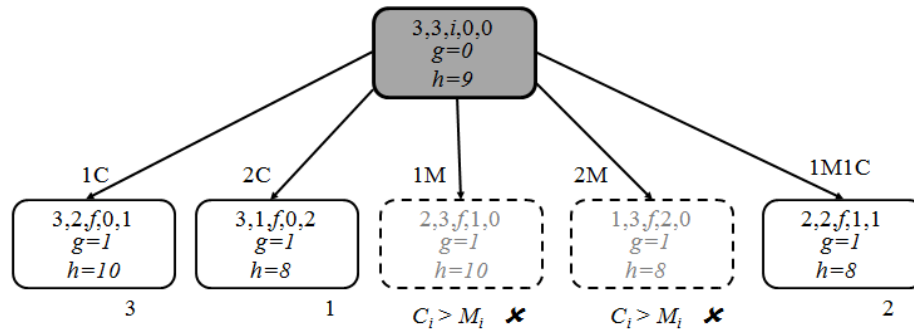
En lo sucesivo, los operadores se aplicarán, siempre, en el mismo orden en el que aparecen en el enunciado, y abreviados como sigue: $1C$, $2C$, $1M$, $2M$ y $1M1C$. Las veces que un operador no puede aplicarse, se indicará explícitamente, dibujando el nodo *imposible* atenuado y con líneas discontinuas:



En cualquier caso, es preciso observar que todos los operadores tienen el mismo coste, 1, por lo que el valor $g(n)$ de cualquier nodo n será siempre igual a la profundidad del árbol en el que aparece.

Iteración 1

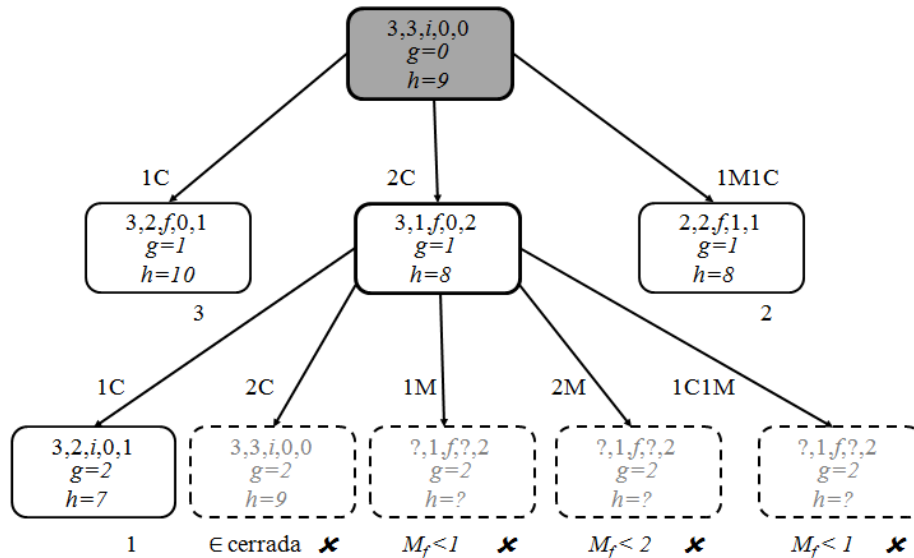
Inicialmente, se considera que el nodo raíz $(3, 3, i, 0, 0)$ ha sido generado e insertado en la lista ABIERTA que, por lo tanto, consiste en un único elemento. La siguiente figura muestra el caso de su expansión:



Como se ve, el nodo inicial es inmediatamente insertado en la lista CERRADA (rodeado entonces de una línea más gruesa) que ahora tiene un único elemento. Nótese que los nodos (2, 3, f, 1, 0) y (1, 3, f, 2, 0) son estados imposibles puesto que en ellos $C_i > M_i$. Por lo tanto, se han generado hasta tres nodos nuevos, que son ordenados ascendentemente en la lista ABIERTA según el índice que se muestra debajo de ellos ¹.

Iteración 2

El primer nodo de la lista abierta, (3, 1, f, 0, 2) no es el estado final. Por lo tanto, es expandido a continuación como se muestra en la siguiente figura:



Al expandirlo, se elimina de ABIERTA y se inserta en CERRADA. Obsérvese como uno de los nuevos sucesores, (3, 3, i, 0, 0) es desestimado inmediatamente por pertenecer a la lista CERRADA o, en otras palabras, porque ya ha sido expandido anteriormente. Asimismo, todos los operadores que pretendían desplazar misioneros desde la orilla final fallarán necesariamente puesto que allí $M_f = 0$.

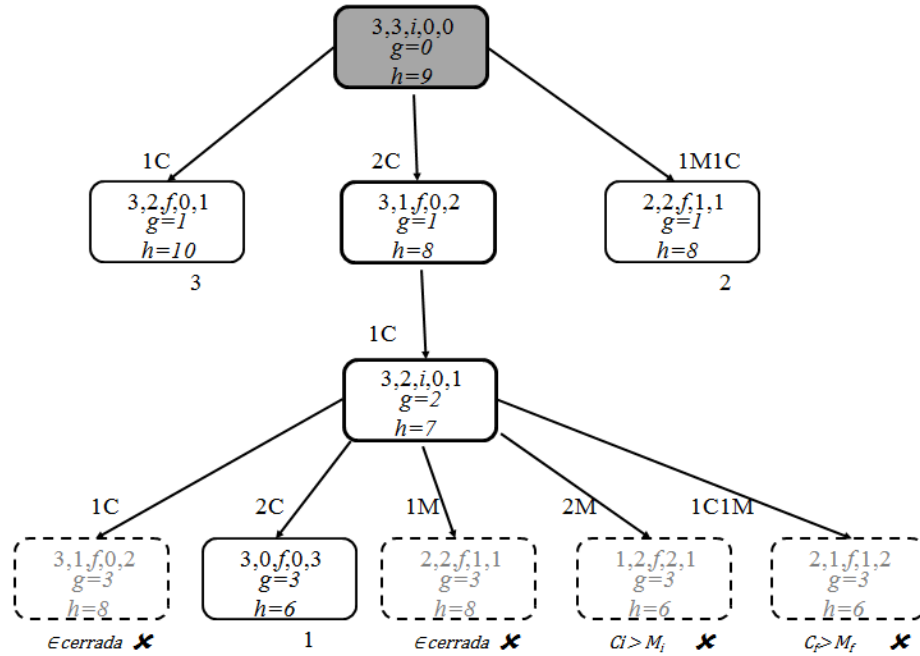
Los sucesores generados se insertan en orden en ABIERTA. Como en este caso existe un único sucesor, el nodo (3, 2, i, 0, 1), es éste el que se inserta en ABIERTA ². Por lo tanto, hay tres nodos en ABIERTA, y dos de ellos empatan en la función de evaluación $f(n)$: el nodo (3, 2, i, 0, 1) y el nodo (2, 2, f, 1, 1). Tanto uno como otro nodo tienen un $f(n) = 9$ resultante de sumar en cada uno de ellos la función de coste, $g(n)$, con la función heurística, $h(n)$. Como se comentaba anteriormente, en caso de empate se escoge siempre el primero más a la izquierda, por lo que el siguiente nodo a expandir será el nodo (3, 2, i, 0, 1).

¹Por convenio, en caso de empate del valor de la función de evaluación $f()$, entre dos o más nodos, se escogerá siempre el primero más a la izquierda.

²Es preciso advertir que si se hubiese generado un nuevo nodo que ya estuviese en ABIERTA pero con una evaluación $f(n)$ menor, el primero sería eliminado de la lista y, en su lugar, se introduciría el nuevo descendiente, puesto que tiene un coste inferior.

Iteración 3

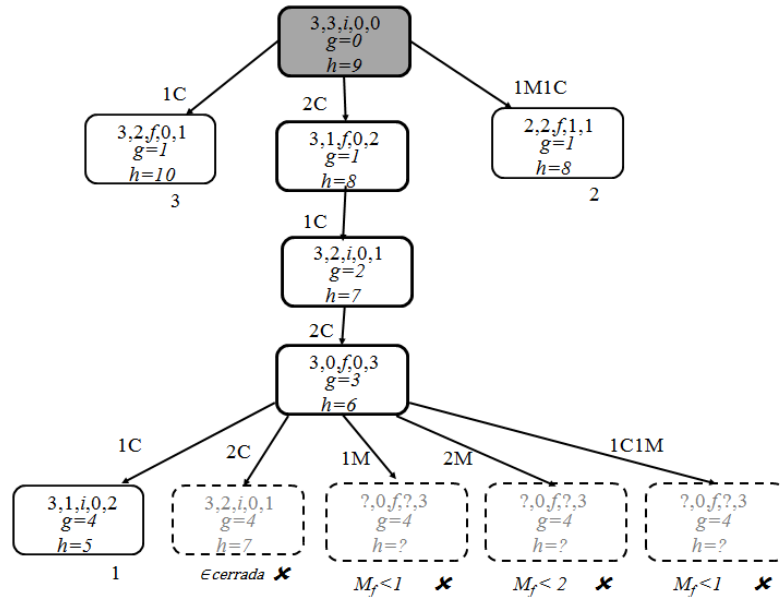
Puesto que el nodo $(3, 2, i, 0, 1)$ no es el estado final, se expande inmediatamente como se muestra a continuación:



Eliminando ahora los sucesores que son estados imposibles, los que ya estaban en CERRADA, y los que no mejoran las alternativas que ya hay en ABIERTA, tenemos que la expansión genera un único nodo, el $(3, 0, f, 0, 3)$, que es insertado en orden en ABIERTA. Nuevamente hay un empate de $f(n)$ de dos nodos que están en ABIERTA: el nodo recién generado $(3, 0, f, 0, 3)$ y el nodo $(2, 2, f, 1, 1)$. Ambos empatan con una $f(n) = 9$, pero nuevamente, por convenio, se escoge el nodo que está más a la izquierda para expandirlo. Por lo tanto, el nuevo nodo a expandir será el nodo $(3, 0, f, 0, 3)$.

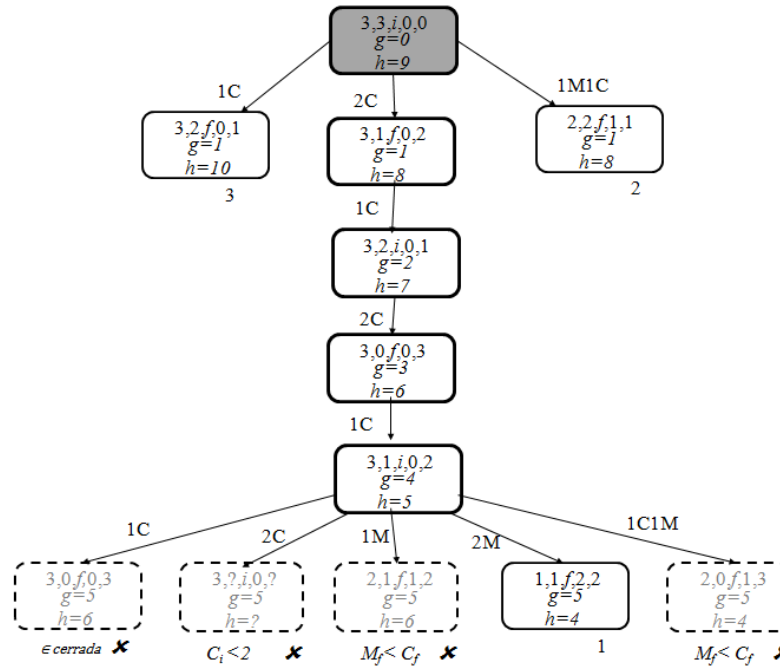
Iteración 4

En esta iteración se expande el nodo $(3, 0, f, 0, 3)$ que se elimina de ABIERTA y se inserta en CERRADA. La expansión de este nodo genera un único sucesor válido, puesto que el resto o bien se encontraban ya en CERRADA o bien incumplían alguna de las precondiciones de los operadores. El nodo $(3, 1, i, 0, 2)$ se inserta en orden en ABIERTA. Por lo tanto, los nodos ordenados en ABIERTA son: $(3, 1, i, 0, 2)$, $(2, 2, f, 1, 1)$ y $(3, 2, f, 0, 1)$. Los dos primeros empatan con una $f(n) = 9$ pero otra vez se emplea el convenio de expandir primero el que se encuentre más a la izquierda. Por eso, en la siguiente iteración se expande el nodo $(3, 1, i, 0, 2)$.



Iteración 5

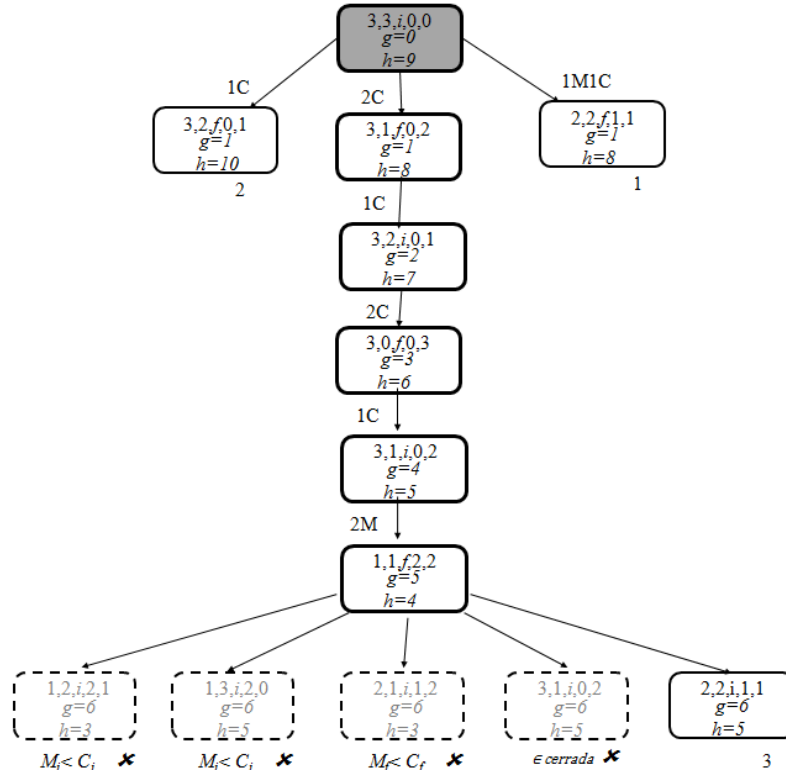
Se expande el nodo (3, 1, i, 0, 2) que se elimina de ABIERTA y se inserta en CERRADA. La expansión se muestra en la siguiente figura:



Como puede verse se genera un único sucesor, el nodo (1, 1, f, 2, 2) que se inserta en orden en ABIERTA. Por lo tanto, en ABIERTA hay tres nodos: (1, 1, f, 2, 2), (2, 2, f, 1, 1) y (3, 2, f, 0, 1), y ya han sido expandidos hasta cinco nodos, actualmente en CERRADA, a saber: (3, 3, i, 0, 0), (3, 1, f, 0, 2), (3, 2, i, 0, 1), (3, 0, f, 0, 3) y (3, 1, i, 0, 2). El siguiente nodo que se expandirá será el nodo (1, 1, f, 2, 2), porque aunque empata en valor $f(n)$ con el nodo (2, 2, f, 1, 1), es el nodo que se encuentra más a la izquierda.

Iteración 6

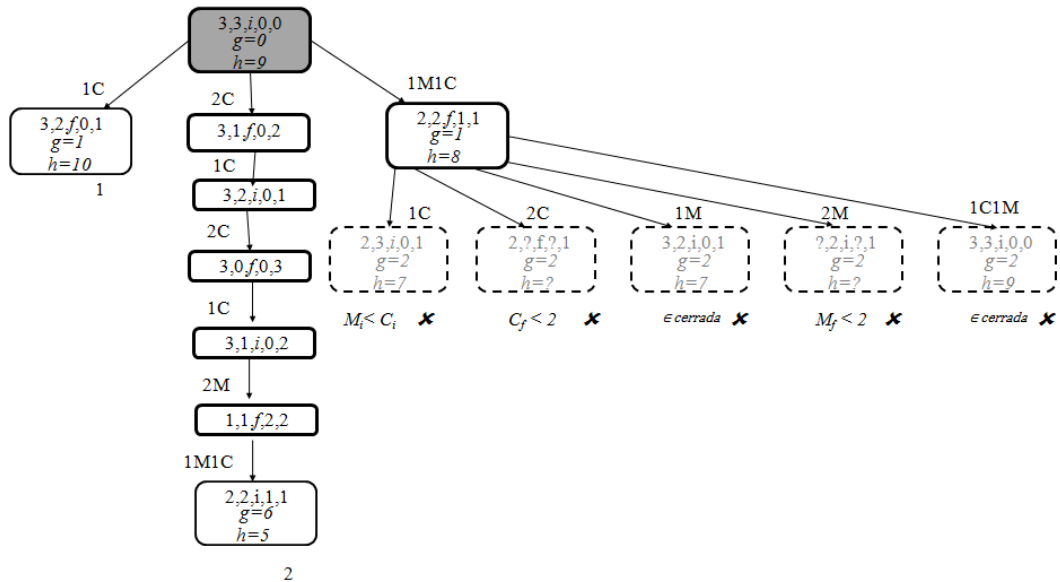
El nodo que se expande a continuación es el nodo (1, 1, f, 2, 2) como se muestra en la siguiente figura:



En este caso, la expansión del nodo genera un único sucesor válido, el nodo (2, 2, i, 1, 1) que insertamos en orden en ABIERTA. Es importante observar que este nodo no es como el nodo (2, 2, f, 1, 1), puesto que mientras que en este caso la barca se encuentra en la orilla inicial, en aquel caso está en la orilla final. El siguiente nodo que se expandirá será el nodo (2, 2, f, 1, 1) que es el nodo con una menor $f(n)$ en ABIERTA.

Iteración 7

A continuación se expande el nodo (2, 2, f, 1, 1) como se muestra en la siguiente figura:

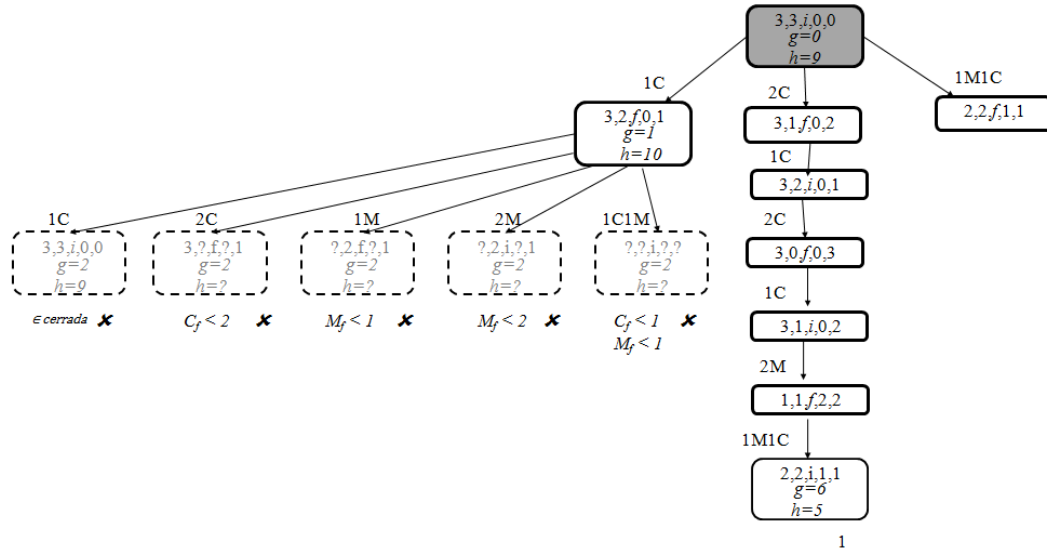


Resulta muy fácil observar que la expansión de este nodo no genera ningún sucesor útil puesto que o bien se incumple alguna precondition de los operadores, o el nodo ya se encontraba en CERRADA.

Obsérvese además la utilidad del mecanismo implementado por la lista CERRADA: efectivamente, desde el estado inicial, (3, 3, i, 0, 0), las trayectorias [2C, 1C] y [1M1C, 1M] son estrictamente equivalentes y, por ello, la segunda puede eliminarse, puesto que la primera ya fue considerada (esto es, expandida) en la tercera iteración.

Iteración 8

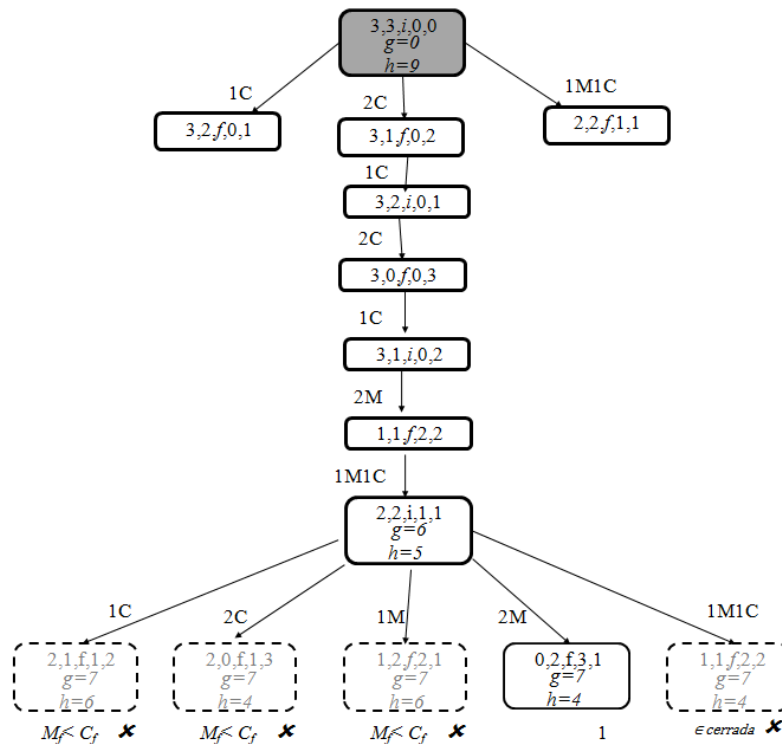
El primer nodo en ABIERTA es el nodo (3, 2, f, 0, 1) que es el nodo que se expande a continuación:



Como en el caso anterior, puede observarse que la expansión de este nodo no genera ningún sucesor válido.

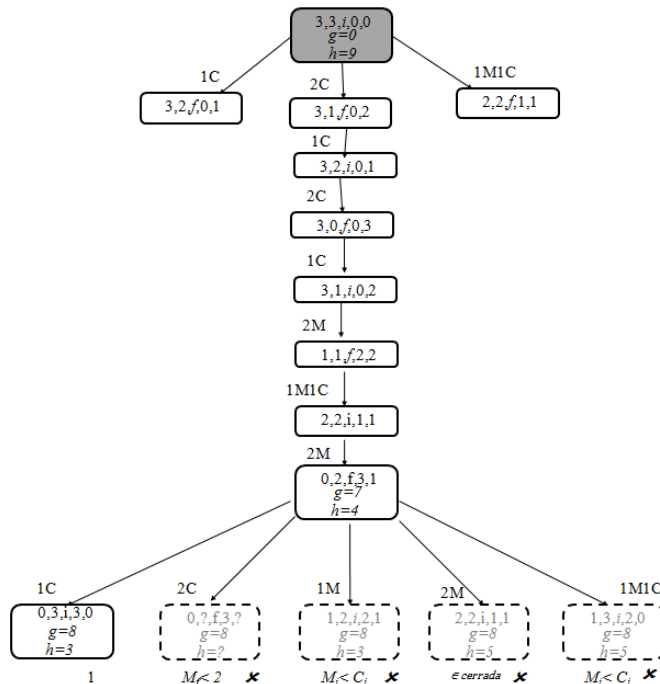
Iteración 9

La lista ABIERTA contiene un único nodo, por lo que se procede a su expansión. Puesto que no se trata del nodo final, el árbol de búsqueda resulta quedar como se muestra a continuación:



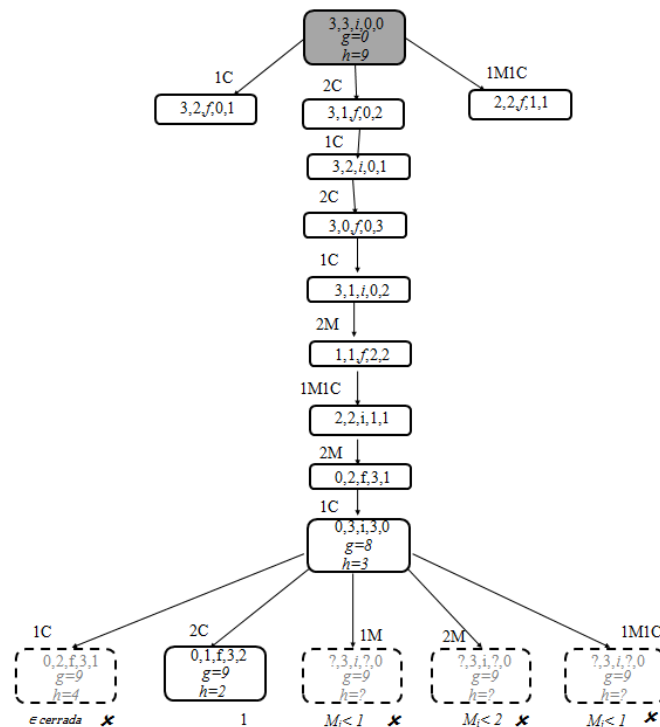
Iteración 10

Como ocurriera en la iteración anterior, ABIERTA tiene de nuevo un único nodo. Después de su expansión, el árbol de búsqueda resultante es:



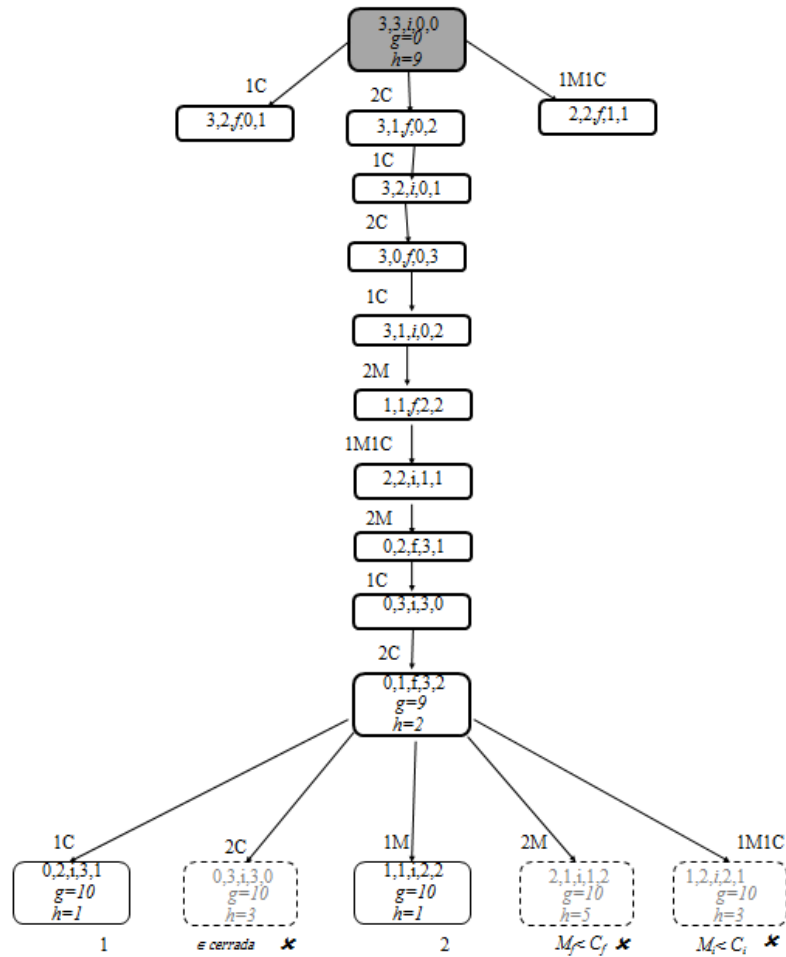
Iteración 11

Una vez más, el número de elementos en ABIERTA es uno. Por lo tanto, después de la expansión de la única alternativa disponible, resulta:



Iteración 12

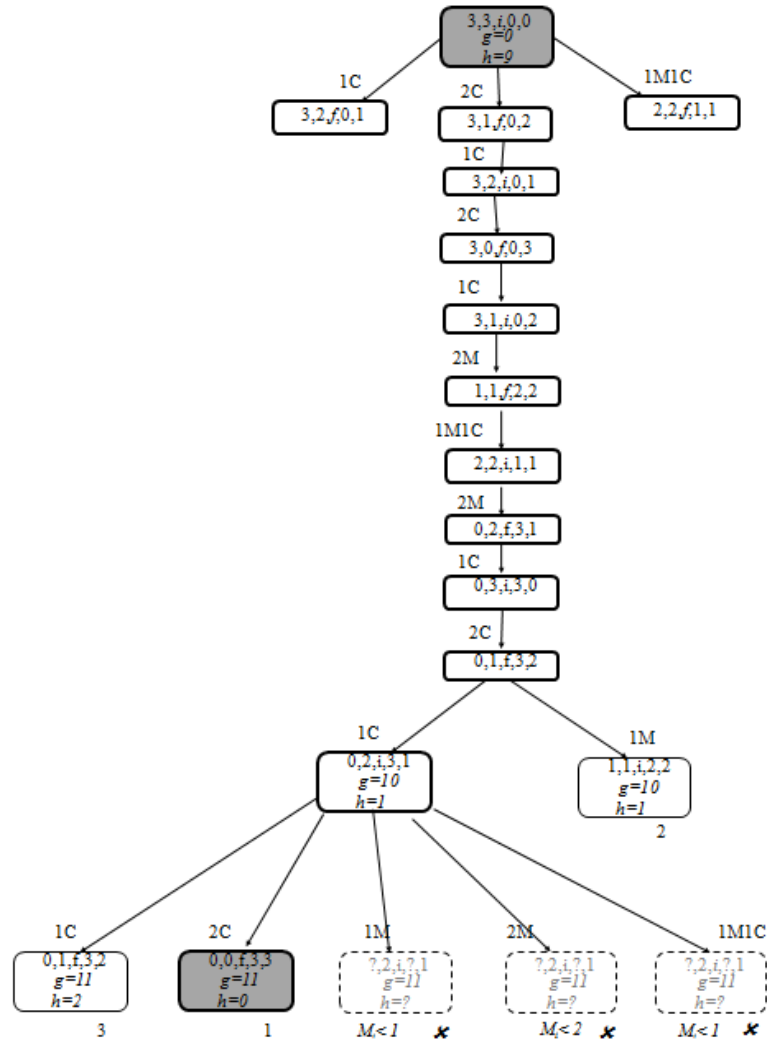
Nuevamente, hay un único nodo que expandir, $(0, 1, f, 3, 2)$:



Iteración 13

En este momento, hay dos nodos en ABIERTA, ordenados tal y como se indica en la figura anterior. En realidad, es trivial comprobar que expandiendo cualesquiera de los dos nodos, se genera por fin el estado final. Sin embargo, el algoritmo A* no se detiene cuando genera una solución, esto es, un camino desde el nodo inicial hasta el nodo final.

El árbol resultante después de la expansión del primer nodo en ABIERTA es:



Nótese que el nodo final (0, 0, f, 3, 3) es el primero (y en realidad, el único) con $h = 0$. Puesto que las funciones heurísticas admisibles no pueden sobreestimar el esfuerzo hasta una solución, y desde el nodo final, el coste óptimo es necesariamente $h^* = 0$, se sigue que $h = 0$. Nótese también que el nodo (0, 2, i, 3, 1), que es el nodo padre del nodo final, tiene un valor heurístico de $h(n) = 1$, y que por lo tanto tampoco sobreestima el coste real de llegar a la meta, puesto que desde ese nodo sólo es necesario un desplazamiento de la barca para alcanzar el estado final. Lo mismo se puede aplicar al resto de nodos del árbol de búsqueda.

Iteración 14

Sin embargo, no será hasta la decimocuarta iteración cuando por fin se acabe la búsqueda. Puesto que el nodo (0, 0, f, 3, 3) es, de todos los nodos en abierta, el que tiene el menor valor $f(n)$, puede garantizarse que no hay ninguna otra alternativa (o nodo en ABIERTA aguardando a ser expandido) que pueda encontrar una solución menor, siempre que la función heurística $h(n)$ sea admisible!, como ocurre en este caso.

Por lo tanto, cuando el algoritmo A* se dispone a expandir el primer nodo en abierta, advierte que se trata del estado final y termina con la solución óptima siguiente: 2C, 1C, 2C, 1C, 2M, 1M1C, 2M, 1C, 2C, 1C, 2C.

Solución ejercicio 2

Éste es un problema excelente para el estudio de la relación entre $g(n)$ y $h(n)$, así como de su impacto en $f(n)$ cuando es usada por la clase de algoritmos denominada de *el mejor primero*.

Los algoritmos de *el mejor primero* emplean un valor $f(n)$ para ordenar ascendentemente los nodos en ABIERTA. Esto es, en cada iteración, escogen el primer nodo (que tendrá el menor valor de $f(\cdot)$) en ABIERTA para ser expandido. Además, puesto que estos algoritmos no se detienen hasta que por fin proceden a expandir el estado final, un resultado bien conocido de esta clase de algoritmos es que expanden todos los nodos con $f(n) \leq C^*$, donde C^* es el coste obtenido en la resolución del estado inicial, s —y que será óptimo o no, si la función heurística, $h(n)$ empleada es admisible o no según se discute a continuación.

Por lo tanto, la elección de $f(n)$ es fundamental, no sólo a propósito del rendimiento del algoritmo (ya sea en el número de nodos expandidos o el tiempo que se tarda en encontrar la solución), sino incluso a propósito de sus propiedades de admisibilidad, esto es, si encontrará o no soluciones óptimas.

Considerando ahora la expresión general para $f(n)$ siguiente: $f(n) = g(n) + h(n)$, donde $g(n)$ es el **coste conocido** para llegar desde el nodo inicial, s , hasta el nodo n y $h(n)$ es una **estimación** del coste para llegar hasta el estado final, t , se consideran los siguientes casos:

- ① La función heurística empleada, $h(n)$, es admisible, esto es, no sobreestima el esfuerzo para llegar hasta el nodo final, t , desde el nodo n . En otras palabras: $h(n) \leq h^*(n)$, donde $h^*(n)$ es el coste óptimo para llegar hasta el estado final desde el nodo n .

Cuando la función heurística es admisible, se sabe que los algoritmos de *el mejor primero* encuentran soluciones óptimas.

- ❶ Un caso especial de admisibilidad de la función heurística es $h(n) = 0 \forall n$ ¡puesto que el coste para alcanzar el estado final será siempre mayor o igual que 0!

A propósito de este caso, cuando un algoritmo no emplea funciones heurísticas, $h(n)$, con el propósito de estimar el esfuerzo para llegar a la solución, a partir del nodo n , y en su lugar emplea únicamente una medida del esfuerzo realizado para llegar hasta el nodo n desde el nodo inicial (esto es, si $f(n) = g(n)$), se sabe que el algoritmo de *el mejor primero* resultante es Dijkstra.

Este algoritmo es capaz de encontrar soluciones óptimas al estado inicial, s , pero lo hace calculando **las soluciones óptimas de todos los estados, n , alcanzables desde el estado inicial con un coste menor o igual que el óptimo, $h^*(s)$.**

Esta conclusión es fácil de entender si se comprende: primero, que $h(n)$ es admisible, como se ha mostrado, de modo que C^* será el coste óptimo para resolver el estado inicial, $h^*(s)$; segundo, que los algoritmos de *el mejor primero* expanden todos los nodos con $f(n) = g(n) \leq C^* = h^*(s)$, como se discutía al principio.

- ❷ El caso general de admisibilidad de la función heurística, consiste en incrementar el valor de $f(n)$ en una cantidad $h(n)$, tal que $h(n) \leq h^*(n)$. De hecho, esta es la expresión que emplean los algoritmos A^* e IDA^* y, en términos generales, prácticamente todos los algoritmos admisibles de *el mejor primero*.

A diferencia del algoritmo de Dijkstra, pero razonando de la misma manera que entonces, estos algoritmos expanden todos los nodos con $f(n) \leq C^*$, pero ahorrando una cantidad considerable de expansiones: todas aquellas para las que $g(n) + h(n)$ excede el coste óptimo del estado inicial.

- ❸ El último caso digno de mención es cuando el cálculo de $f(n)$ se resume al cómputo de $h(n)$, esto es, cuando no se considera el coste conocido $g(n)$: $f(n) = h(n)$.

Como antes, el algoritmo de *el mejor primero* resultante, expandirá nodos con $f(n) = h(n) \leq C^*$. Ahora bien, resulta fácil observar que las soluciones encontradas no tienen por qué ser óptimas en absoluto. Considérense dos nodos n_1 y n_2 con $h(n_1) = h(n_2)$ pero tal que $g(n_1) < g(n_2)$, de modo que, en realidad, el nodo n_1 es preferible al nodo n_2 —puesto que es más barato llegar hasta él y la estimación para llegar hasta t desde n_1 es igual que desde n_2 , más caro de alcanzar desde el nodo inicial, s . Sin embargo, puesto que este algoritmo no considera los costes $g(n)$ podría igualmente expandir el nodo n_2 antes que n_1 .

- ❹ La función heurística empleada, $h(n)$ no es admisible y, por lo tanto, sí sobreestima el esfuerzo para llegar hasta el nodo final, t , desde el nodo n o, equivalentemente: $h(n) > h^*(n)$.

Un resultado muy importante en Inteligencia Artificial es que si se expresa la sobreestimación como un factor ϵ , esto es, si se dice que $f(n) = g(n) + (1 + \epsilon)h^*(n)$, el algoritmo de *el mejor primero* resultante no excederá el coste óptimo del estado inicial, $h^*(s)$, en $(1 + \epsilon)$ o, en otras palabras, que $C^* \leq (1 + \epsilon)h^*(s)$.

Después de la discusión de estos casos, sólo será preciso relacionar los valores w propuestos en el enunciado con los casos conocidos.

Puesto que en el enunciado no se decía nada sobre la admisibilidad o no de la función heurística, se estudiarán ambos casos. Primero, si la función heurística $h(\cdot)$ es admisible:

- ① Con $w = 0$, se tiene $f(n) = 2g(n)$, que se distingue del caso estudiado $f(n) = g(n)$ en el factor, 2.

Sin embargo, es fácil advertir que los nodos ordenados por $f(n) = g(n)$ no cambiarían en absoluto su posición en ABIERTA con $f(n) = 2g(n)$, puesto que si un valor $f(n_1)$ es menor o igual que cualquier otro $f(n_2)$, necesariamente se sigue que $2f(n_1) \leq 2f(n_2)$.

Por lo tanto, con $w = 0$, el algoritmo resultante es Dijkstra, tal y como se discutió en el punto ❶.

- ② Con $w = 1$, se tiene $f(n) = g(n) + h(n)$, exactamente la misma función de evaluación empleada por los algoritmos A^* e IDA*, discutida en el punto ❷
- ③ Con $w = 2$, resulta la función de evaluación $f(n) = 2h(n)$, que se asimila inmediatamente al punto ❸, razonando sobre la ordenación de la lista abierta como en el caso $w = 0$.
- ④ De todos los casos mencionados, nótese que sólo ❶ y ❷ devolvían soluciones óptimas si $h(\cdot)$ es admisible, tal y como se explicaba allí.

Por lo tanto, siempre que la función heurística empleada sea admisible, los valores de w para los que el algoritmo resultante es admisible son $w \in [0, 1]^3$.

Por último, en caso de que la función heurística $h(\cdot)$ no sea admisible:

- ① Si $w = 0$, se tiene $f(n) = 2g(n)$, como en el caso $w = 0$ anterior. Por lo tanto, puesto que $f(n)$ no usa los valores de $h(\cdot)$, se siguen aquí las mismas conclusiones que en el caso $w = 0$ con funciones heurísticas admisibles.
- ② Si $w = 1$ resultará, como antes $f(n) = g(n) + h(n)$. Por otra parte, si $h(n)$ es inadmisble, existirá algún $\epsilon > 0$, tal que $h(n) \leq (1 + \epsilon)h^*(n)$, de modo que, tal y como se vió anteriormente, el coste C^* calculado por el algoritmo de *el mejor primero* resultante no excederá el óptimo en una cantidad igual a $(1 + \epsilon)$.
- ③ Si $w = 2$, la función de evaluación resultante, $f(n) = 2h(n)$ es inherentemente inadmisble y, si en el caso de funciones heurísticas admisibles ya se había demostrado que el algoritmo podría encontrar soluciones no óptimas, el mismo caso resulta mucho más evidente ahora. Más aún, las soluciones encontradas por un algoritmo de *el mejor primero* como éste serán típicamente no óptimas.
- ④ Por último, repasando todas las consideraciones realizadas para estos casos, resulta trivial observar que no existe ningún $w > 0$ (caso ❶) para el que resulte un algoritmo admisible que usa una función heurística inadmisble.

Solución ejercicio 3

Como de costumbre, hay varias alternativas para representar cada estado. La más fácil consiste, simplemente, en emplear un vector de posiciones, cada una de las cuales tiene asignada una posición del tablero, cuyo contenido podría ser: p , l o \square , representando el vacío. Así, por ejemplo, ordenando las casillas del tablero por filas primero, y columnas después, el estado inicial se describiría computacionalmente como sigue: $\langle p, \square, \square, p, \square, \square, \square, l, p, \square, \square \rangle$. Existen muchísimos otros formalismos que podrían emplearse, pero este, de hecho, sería perfectamente válido.

Para la descripción de los operadores es preciso distinguir entre los movimientos de los perros y los de las liebres. Para ello, se definen dos operadores diferentes: **MoverPerro**(x) y **MoverLiebre**(x) que reciben como argumento la posición desde la que hay que moverlos. Obviamente, que en esa posición haya un perro o una liebre, según el caso, es una *precondición* ineludible. Pero, además, es preciso considerar los movimientos legales representados con arcos en el tablero. Por ejemplo, con un perro (p) en la posición 1 (como ocurre, de hecho, en la situación inicial), las únicas jugadas legales en **MoverPerro**(1) deberían ser $\{2, 5, 6\}$, dado por supuesto que esas posiciones estén libres —esto es, que ni siquiera esté la liebre. Aunque sea poco elegante, la forma más eficiente de implementar estos

³Sólo se ha demostrado que con $w = 0$ y $w = 1$, el algoritmo de *el mejor primero* resultante es admisible si la función heurística que lo guía lo es también. Sin embargo, las discusiones presentadas en esta solución deberían bastar para entender *intuitivamente* que la misma conclusión se sigue para cualquier valor $w \in [0, 1]$.

operadores es considerando de forma específica, casilla por casilla, las posiciones libres adyacentes y devolviendo un conjunto que contiene aquellas que, además, están desocupadas.

Para el diseño de una función de evaluación es necesario identificar la expresión que pretende minimizarse o maximizarse. Por ejemplo, en ajedrez el material podría servir como función de evaluación para un jugador *voraz* (pero muy poco inteligente, desde luego). En este caso, pueden considerarse diferentes medidas de lo cerca o lejos que están ambos jugadores de acabar ganando la partida. Por lo tanto, son funciones de evaluación plausibles las siguientes:

- ① Podría entenderse que cuanto más cerca esté la liebre del extremo izquierda (casilla 4) del tablero, más cerca está de ganar sin ninguna otra consideración. En ese caso, los perros querrían maximizar **la distancia de la liebre a su destino**, mientras que la liebre querría minimizarla:

$$f_1(x) = \text{Distancia}(l, 4)$$

- ② Alternativamente, podría entenderse que, puesto que los perros quieren restringir los movimientos de la liebre completamente, **el número de movimientos factibles de la liebre** representaría esta función de evaluación.

$$f_2(x) = |\text{Movimientos}(l)|$$

- ③ Asimismo, podría entenderse que cuantas más formas tenga la liebre para llegar a su destino, más posibilidades tendrá entonces de ganar la partida —puesto que los perros deberían cerrar entonces más “*agujeros*”. Este concepto se representaría fielmente con el uso de una función de evaluación como la siguiente:

$$f_3(x) = |\text{Caminos}(l)|$$

donde *Caminos* es una función que emplea un algoritmo de búsqueda de un agente para contar todas las formas de llegar hasta el destino. ¿Serías capaz de hacerlo? ¿Qué algoritmo elegirías?

Por supuesto, también es posible combinar linealmente (o no) estas *características* (*features*, en la bibliografía especializada) para concederles diferentes importancias. Así, son funciones plausibles (pero de un propósito dudoso en esta ocasión) $f_1(x) + f_2(x) + 3f_3(x)$ o $\frac{f_1(x)}{f_2(x)}$.

En definitiva, cuanto mayor sea la comprensión del juego, mejor resultan las estimaciones de la función de evaluación. En este caso, aún es posible profundizar un poco más en el juego para advertir lo siguiente:

- ① Los perros ganan sólo si encierran a la liebre. Puesto que son tres, sólo hay unas pocas casillas con grado 3: 2, 8 y 10. En el resto de las posiciones resulta claro que la liebre nunca quedará acorralada.
- ② Por lo tanto, los perros deben forzar a la liebre a ocupar una de estas casillas y, a continuación, ocupar la última casilla adyacente a la ocupada recientemente por la liebre, para poder ganar.
- ③ Teniendo en cuenta que los perros no pueden capturar en esta versión a la liebre, las reglas anteriores (junto con esta última consideración) se reduce a considerar la *paridad/imparidad* de las alternativas para la liebre y los perros, esto es:
 - ❶ Si la liebre dispone de movimientos de longitud *par/impar* para llegar a las casillas 2, 8 ó 10.
 - ❷ Si los perros disponen de una cantidad de movimientos *par/impar* para encerrar a la liebre en las mismas casillas 2, 8 ó 10.

Obviamente, si ahora:

- ❶ Es el turno de la liebre. Para ganar la liebre, necesariamente debe dirigirse a la casilla 2, 8 ó 10 a la que pueda llegar en un número *par/impar* de movimientos a sabiendas de que los perros no disponen de una combinación de la misma paridad. Si fuera así, entonces, necesariamente, la liebre perdería si ya está encerrada en una región (por muy grande que sea).
- ❷ Es el turno de los perros. Aplican las consideraciones estrictamente contrarias que en el caso anterior.

De hecho, considerando ahora esta función de evaluación es fácil asignar en cualquier posición alguno de los valores $\{-1, 0, +1\}$ si la liebre pierde, empata o gana, respectivamente. Más aún, es fácil, muy fácil, demostrar (jugando) que el valor teórico de este juego (o, en otras palabras, el valor *minimax*) del árbol de búsqueda completamente desarrollado es 0, esto es, tablas.

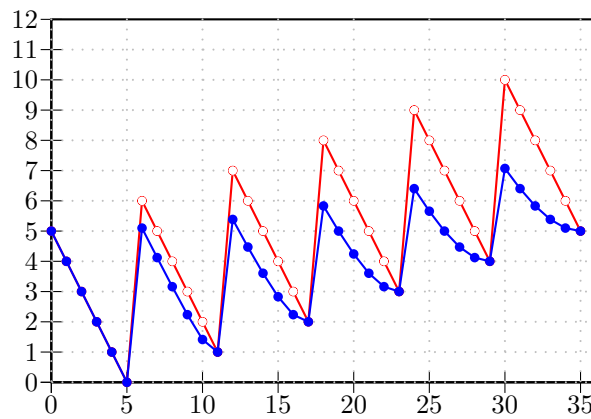


Figura 4: Distancia Euclídea vs. Distancia de Manhattan en mallas

Solución ejercicio 4

Para la obtención de una función heurística, lo mejor es emplear el modelo de *relajación de restricciones* que consiste en simplificar los problemas reales considerando la violación *selectiva* de algunas restricciones, de modo que, en el problema resultante (denominado “*problema relajado*”) sea posible calcular el coste óptimo de las soluciones. El valor heurístico, $h(n)$, de cualquier problema real, n , será entonces el coste óptimo de su versión relajada.

En este caso, una restricción susceptible de ser relajada son los obstáculos. Por lo tanto, relajando esta restricción, resulta un problema mucho más sencillo donde, simplemente, ¡no hay obstáculos! En un problema simplificado de este modo, el coste óptimo para llegar desde cualquier casilla $P(x, y)$ hasta el estado final, $(5, 0)$ (considerando que el origen de coordenadas, $(0, 0)$, está en la casilla inferior izquierda) es la **distancia de Manhattan**, definida como la suma de la diferencia de filas y columnas entre ambas posiciones. Formalmente: $d_m(P(x, y)) = |x - 5| + |y - 0| = |x - 5| + y$.

Sin embargo, podría relajarse, además, el hecho de que sólo es posible transitar a las posiciones horizontal o verticalmente adyacentes, después de haber relajado la presencia de los obstáculos. En este caso, el coste óptimo del nuevo problema simplificado es la **distancia euclídea** o **aérea** entre $P(x, y)$ y el estado final, $(5, 0)$ definida de la siguiente manera: $d_e(P(x, y)) = \sqrt{(x - 5)^2 + (y - 0)^2} = \sqrt{(x - 5)^2 + y^2}$.

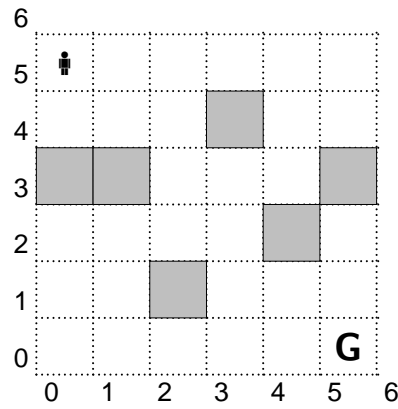
Puesto que se dispone de más de una función heurística, debe elegirse entre una de ellas. Típicamente, cuando una función heurística resulta de más relajaciones que otra, la primera suele ofrecer (evidentemente) peores estimaciones que la segunda. En este caso, la distancia euclídea, d_e , ha resultado de relajar aún más restricciones que las consideradas para la obtención de la distancia de Manhattan, d_m , por lo que esta última debería ser la elegida. De hecho, resulta muy fácil demostrar analíticamente que $d_m \geq d_e$ o, en otras palabras, que d_m **es más informada** que d_e . La figura ?? muestra con puntos sólidos los valores de la distancia euclídea y con puntos huecos, los valores de la distancia de Manhattan⁴. Como puede verse, la distancia de Manhattan es siempre mayor o igual que la euclídea, por lo que será la que se usará en el resto de este ejercicio.


Antes de resolver los siguientes apartados, será bueno recordar que el algoritmo de búsqueda en *escalada* (*hill-climbing*, en inglés) simplemente expande el sucesor con la mejor evaluación heurística, de entre los nodos recién generados. De hecho, se dice que este algoritmo implementa una *política irrevocable* porque escoge inmediatamente un sucesor desestimando todos los demás sin hacer ninguna otra consideración que el valor de la función heurística de cada sucesor.

Aunque este algoritmo se implementa frecuentemente sin mantener una lista de todos los nodos visitados, en los siguientes casos, sí se considera que el algoritmo conoce siempre el camino que le ha llevado hasta él desde el nodo raíz. Por este motivo, en vez de mostrar un árbol de búsqueda explícitamente, se describirá el desarrollo del algoritmo de búsqueda dibujando sobre el tablero las casillas visitadas y los valores heurísticos de cada sucesor.

Considerando, primero, que los sucesores de cualquier casilla se generan siempre en el orden: {derecha (\rightarrow), abajo (\downarrow), izquierda (\leftarrow), arriba (\uparrow)} y que el robot está inicialmente en $(0, 5)$, el estado inicial se describe simplemente por:

⁴En esta gráfica, las abscisas son el punto $(i * 6 + j)$ donde i es la fila y j es la columna



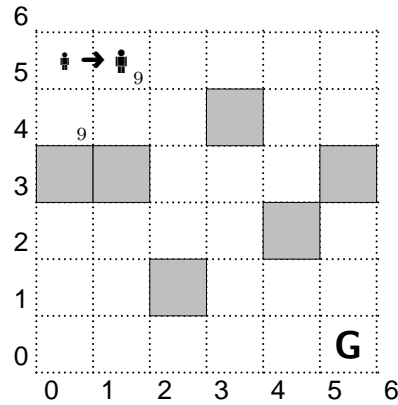
donde la posición del robot se indica con el símbolo .

Después de expandir el estado inicial, se originan sólo dos estados factibles (puesto que moviendo el robot en cualquiera de las direcciones \leftarrow o \uparrow , se exceden los límites de la habitación, generando entonces estados infactibles) cuyas evaluaciones heurísticas son iguales, $h(P(1, 4)) = h(P(2, 5)) = 9$, puesto que, según la distancia de Manhattan:

$$d_m(P(0, 4)) = |0 - 5| + |4 - 0| = 5 + 4 = 9$$

$$d_m(P(1, 5)) = |1 - 5| + |5 - 0| = 4 + 5 = 9$$

Puesto que el algoritmo de búsqueda en *escalada* únicamente considera los valores de las funciones heurísticas, y éstas son iguales para los únicos sucesores del estado inicial, la selección se resuelve escogiendo el primer nodo generado que, en la ordenación que actualmente se está empleando (\rightarrow , \downarrow , \leftarrow y \uparrow) será \rightarrow , de modo que el robot pasa a la posición (1, 5):



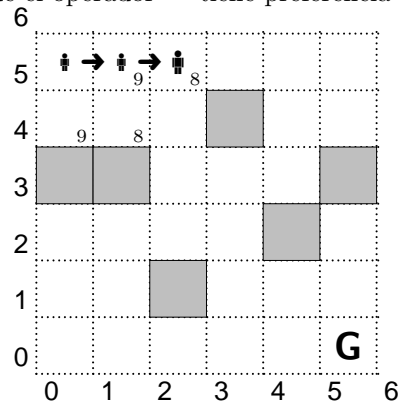
donde los nuevos sucesores contienen sus valores heurísticos en la esquina inferior derecha.

Ahora, la expansión del estado actual, (1, 5) genera hasta tres sucesores (después de deshechar la dirección \uparrow que llevaría al robot fuera de la habitación) de los que (0, 5) (al que se llega gracias a la aplicación del operador \leftarrow) es eliminado, puesto que pertenece al camino desde la raíz hasta el estado actual, lo que se indica explícitamente en el gráfico anterior con el mismo símbolo empleado para el robot pero con una grafía más pequeña. Los nuevos estados, (2, 5) y (1, 4) una vez más, tendrán la misma evaluación, 8:

$$d_m(P(1, 4)) = |1 - 5| + |4 - 0| = 4 + 4 = 8$$

$$d_m(P(2, 5)) = |2 - 5| + |5 - 0| = 3 + 5 = 8$$

de modo que se escoge (2, 5), puesto que el operador \rightarrow tiene preferencia sobre \downarrow , de modo que resulta:

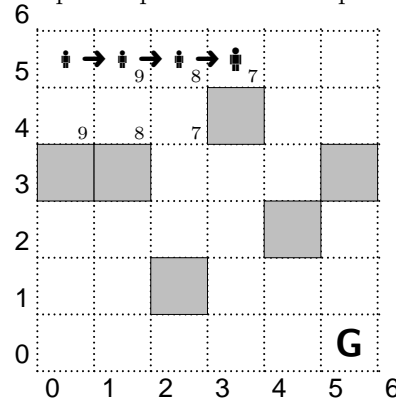


Nuevamente, el nodo $(2, 5)$ es expandido y hasta dos nuevos sucesores son generados, $(2, 4)$ y $(3, 5)$, cuyos valores heurísticos coinciden:

$$d_m(P(2,4)) = |2 - 5| + |4 - 0| = 3 + 4 = 7$$

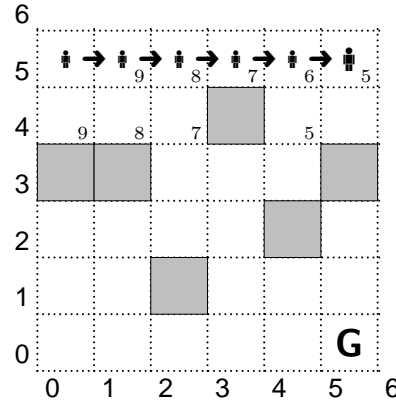
$$d_m(P(3,5)) = |3 - 5| + |5 - 0| = 2 + 5 = 7$$

y aunque, obviamente, la solución óptima desde el estado actual, $(2, 5)$ pasa por el sucesor $(2, 4)$, el algoritmo de búsqueda en *escalada* elegirá $(3, 5)$, puesto que el operador \rightarrow tiene preferencia sobre \downarrow , resultando el estado:

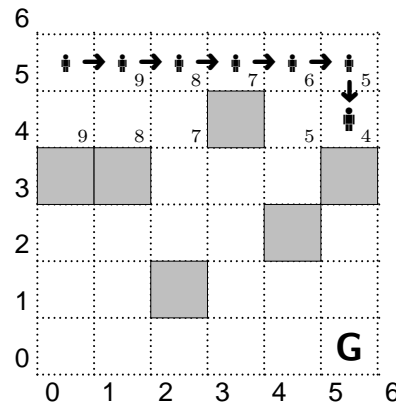


Resulta notable el hecho de que tan pronto como el algoritmo de búsqueda en *escalada* escoge en este punto, el sucesor $(3, 5)$, se olvida inmediatamente de $(2, 4)$. Se dice entonces que el nodo $(2, 4)$ ha sido completamente desechado y, de hecho, no volverá a ser considerado salvo que, por supuesto, pasemos de nuevo por el nodo, $(2, 5)$.

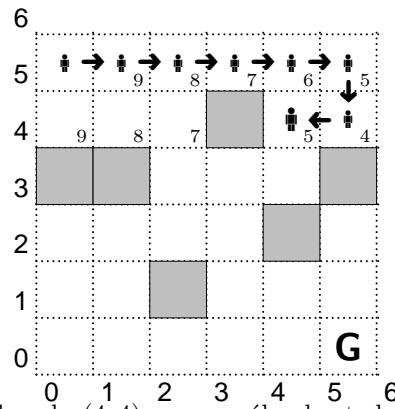
Es fácil ver que en dos pasos más, el algoritmo de búsqueda en *escalada* llegará exactamente a la posición siguiente:



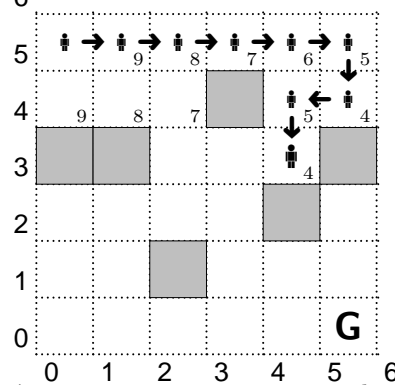
Llegados a este punto, la expansión del nodo actual, $(5, 5)$, sólo puede producir un estado útil, $(5, 4)$ puesto que, como hasta ahora, los operadores \rightarrow y \uparrow no conducían al robot a posiciones legales y la posición $(4, 5)$ está en la lista de posiciones visitadas desde el nodo raíz, $(0, 5)$ hasta la posición actual, $(5, 5)$. Circunstancialmente, la puntuación del único sucesor útil del estado actual tiene un valor heurístico (4) aún menor que el valor heurístico del nodo actual (5) pero debe advertirse que el algoritmo de búsqueda en *escalada* lo habría escogido igualmente, por ser la única alternativa:



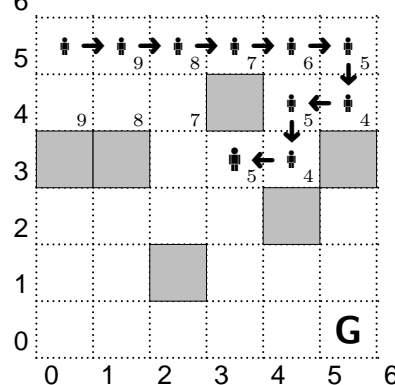
Como antes, la expansión del nodo actual, $(5, 4)$, sólo genera el estado $(4, 4)$. Aunque el valor heurístico de esta posición es $h(4, 4) = d_m(P(4, 4)) = 5$, mayor que el valor heurístico del nodo actual, $h(5, 4) = d_m(P(5, 4)) = 4$, este sucesor será irremediabilmente escogido por ser la única alternativa disponible:



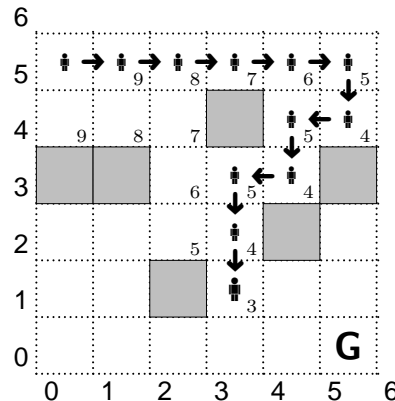
Nótese que, ahora, la expansión del nodo $(4,4)$ genera sólo el estado $(4,3)$ que, aunque tiene el mismo valor heurístico que la posición $(5,4)$ (esto es, $h(4,3) = h(5,4) = 4$), ésta última es desestimada por formar parte del camino desde la raíz hasta la posición actual. Con una única alternativa factible, el algoritmo de búsqueda en *escalada* llega ahora a la siguiente posición:



desde la cual sólo es posible llegar a un único nuevo sucesor con un valor heurístico igual a 5, el estado $(3,3)$:

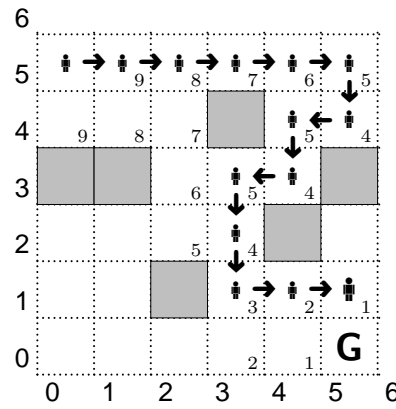


Desde esta posición es fácil ver que el algoritmo de búsqueda en *escalada* llegará en dos pasos a la siguiente posición:

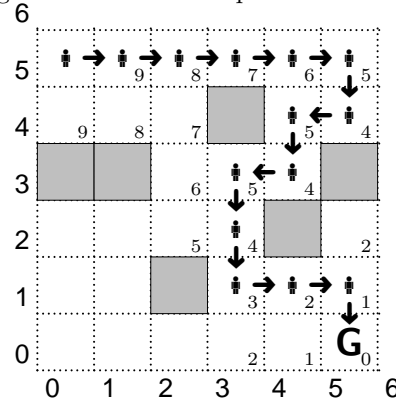


puesto que desde las posiciones $(3,3)$ y $(3,2)$, el operador \downarrow siempre conduce al sucesor con el mejor valor heurístico.

Como ocurriera al principio, los sucesores que ahora se generan corresponden únicamente a la aplicación de los operadores \rightarrow y \downarrow , de entre los cuales \rightarrow tiene preferencia, de modo que en sólo dos pasos se llega exactamente a la siguiente configuración:



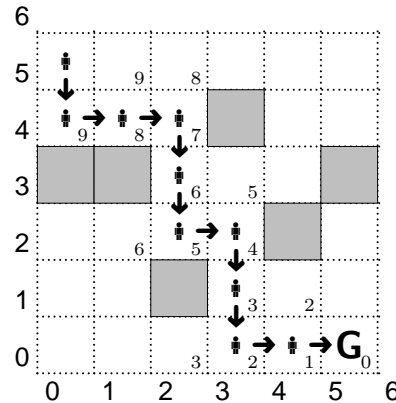
Por fin, después de la expansión del estado actual, $(5,1)$, se reconoce el estado final, $(5,0)$, con $h(5,0) = d_m(P(5,0)) = |5 - 5| + |0 - 0| = 0$. Puesto que el algoritmo de búsqueda en *escalada* se detiene cuando coincide *eventualmente* con el estado final, el algoritmo acaba en la posición:



con la solución $\langle \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \downarrow, \leftarrow, \downarrow, \leftarrow, \downarrow, \downarrow, \rightarrow, \rightarrow, \downarrow \rangle$

Si, en cambio, ahora se considera la ordenación de operadores siguiente $\{\uparrow, \leftarrow, \downarrow \text{ y } \rightarrow\}$, tal y como se sugería en el tercer apartado, resultará trivial comprobar que el camino explorado por el algoritmo de *escalada* para encontrar la solución final es el que se muestra en el cuadro ???. Con el objeto de abreviar la exposición se muestra, en cada caso, el estado que resulta antes de aplicar un operador distinto al último. Nótese, por ejemplo, cómo desde el estado $(2,2)$, el algoritmo escoge el sucesor $(3,2)$, en vez de $(1,2)$ puesto que, aunque \leftarrow tiene precedencia sobre \rightarrow , lo cierto es que $h(3,2) = 4 < h(1,2) = 6$. En otras palabras, es importante advertir que la precedencia de operadores se aplica en caso de empate entre las evaluaciones heurísticas.

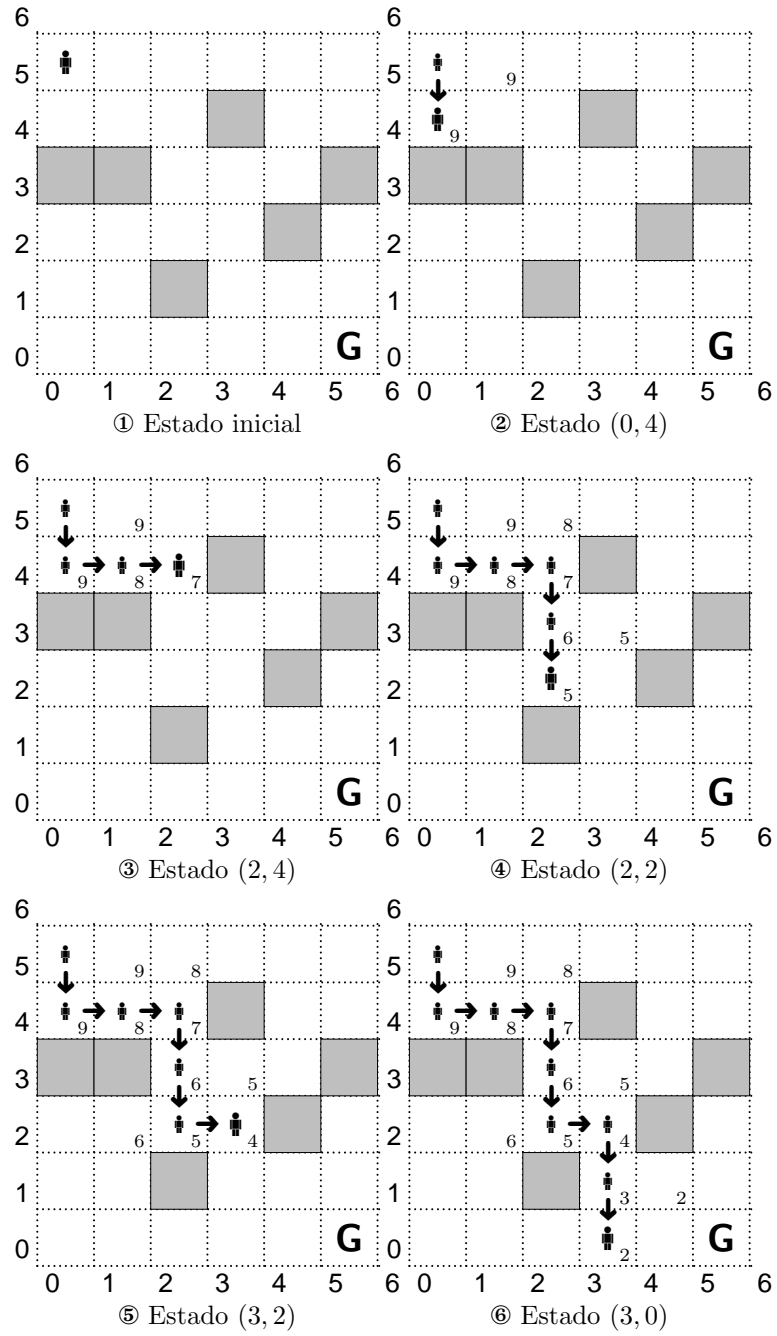
Por fin, desde el paso ⑥, es posible generar en sólo dos pasos el estado final siguiendo, simplemente, los sucesores con la menor puntuación heurística:



con la solución $\langle \downarrow, \rightarrow, \rightarrow, \downarrow, \downarrow, \rightarrow, \downarrow, \downarrow, \rightarrow, \rightarrow \rangle$ que, de hecho, es la solución óptima, con sólo 10 pasos, en vez de los 14 pasos de la solución anterior.

El orden de los operadores ha influido para la obtención de una solución mejor que la anterior

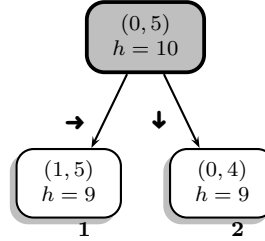
Y ahora, ¿será posible generar también una buena solución empleando el algoritmo de búsqueda en haz en vez de *escalada*? Generalmente, la respuesta es que sí, pero esto depende también del valor elegido para k . En el



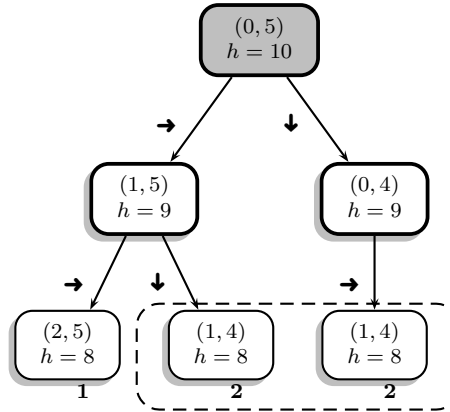
Cuadro 2: Algoritmo de búsqueda en *escalada* con la ordenación $\{\uparrow, \leftarrow, \downarrow \text{ y } \rightarrow\}$

cuarto apartado se sugiere intentar resolver el problema con la primera ordenación de nodos, $\{\rightarrow, \downarrow, \leftarrow, \uparrow\}$ (para la que el algoritmo de búsqueda en *escalada* no encontró la solución óptima), con $k = 3$, esto es, manteniendo constantemente las $k = 3$ mejores alternativas en una lista denominada ABIERTA.

En este caso, puesto que es preciso examinar simultáneamente hasta k alternativas diferentes, se usará un árbol de búsqueda para describir explícitamente el comportamiento de este algoritmo. En las descripciones que siguen, cada estado se representa como un par (x, y) que indica la posición actual del robot y que es relativo al origen de coordenadas arbitrariamente situado en la casilla inferior izquierda. Como antes, la función heurística empleada será la distancia de Manhattan, d_m . Por lo tanto, para encontrar el mejor camino desde el estado inicial, $(0, 5)$, hasta el estado final, $(5, 0)$, usando la distancia de Manhattan, se tiene que, después de la expansión del nodo inicial resultarán sólo dos descendientes:



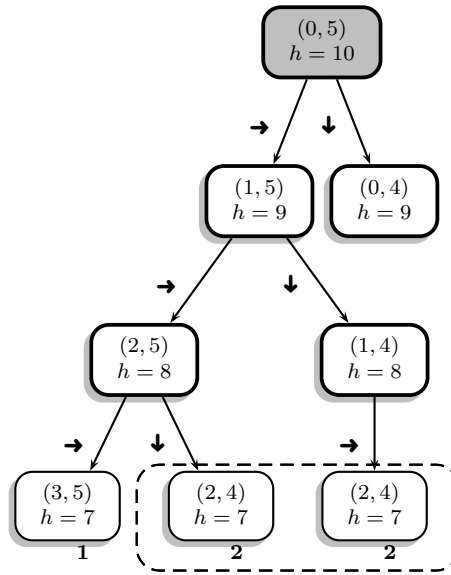
donde los sucesores han sido generados, efectivamente, con la ordenación sugerida: $\{\rightarrow, \downarrow, \leftarrow, \uparrow\}$. Después de insertar los nuevos sucesores, $(1, 5)$ y $(0, 4)$ en la lista ABIERTA y ordenarlos, se escogen los $k = 3$ mejores descendientes. Esto es, la lista de sucesores a expandir resulta inalterada puesto que sólo hay $2 < k = 3$ nodos que, a continuación, son expandidos simultáneamente como sigue:



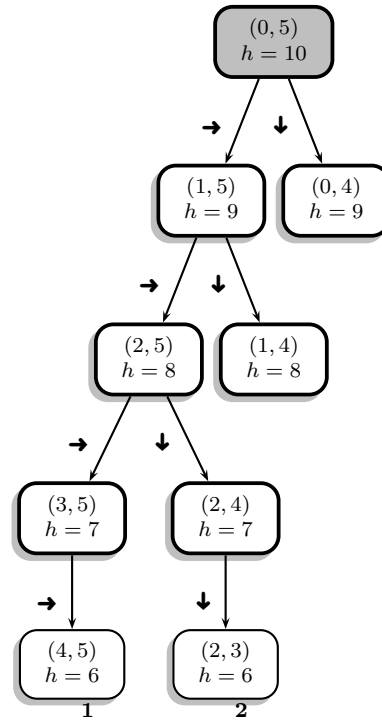
Por supuesto, al insertar los nuevos descendientes en la lista ABIERTA, se encuentra que el nodo $(1, 4)$ es generado por dos caminos diferentes: $\langle \rightarrow, \downarrow \rangle$ y $\langle \downarrow, \rightarrow \rangle$ pero, evidentemente, es insertado una única vez, puesto que siempre tienen la misma puntuación⁵ $h(1, 4)_{\langle \rightarrow, \downarrow \rangle} = h(1, 4)_{\langle \downarrow, \rightarrow \rangle} = 8$. Este hecho se marca explícitamente en todos los árboles de búsqueda mostrando los nodos repetidos (denominados en la bibliografía especializada como *transposiciones*) en una caja punteada.

Por lo tanto, una vez más, se tienen solo dos nodos útiles de modo que la lista de sucesores resulta inalterada con $k = 3$. Después de expandirlos resulta:

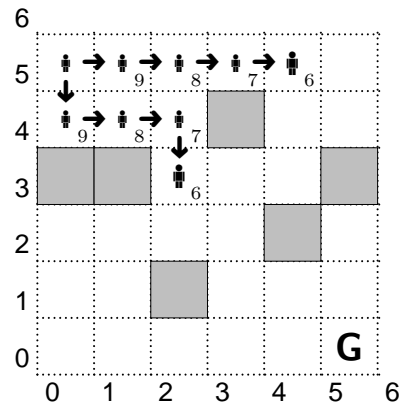
⁵Debe advertirse, en relación con otros algoritmos que empleen el valor de $g(n)$, como el A* o el IDA*, que en esos casos sí que puede ocurrir que sea preciso distinguir entre caminos que llevan hasta un mismo nodo, puesto que en un caso podrían encontrarse caminos más económicos que otros.



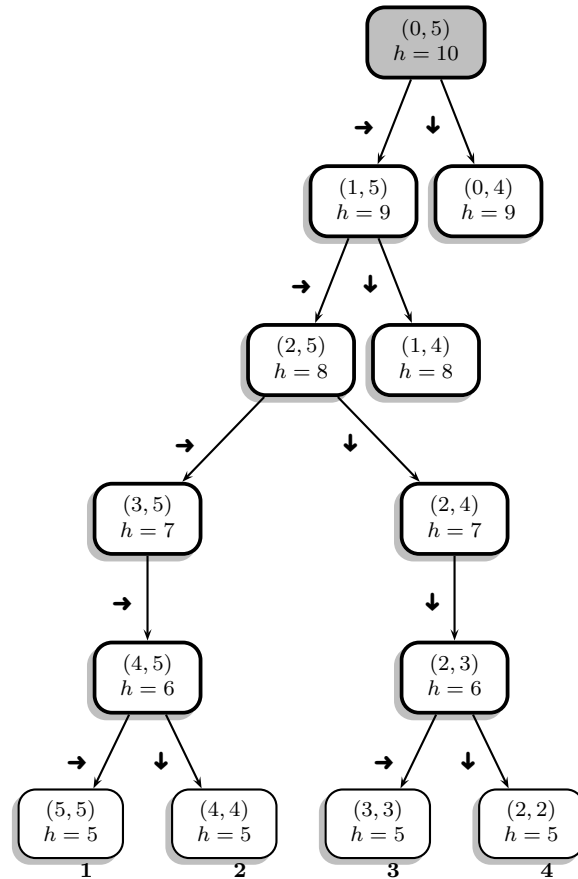
De nuevo, se genera el mismo nodo $(2, 4)$ por dos caminos diferentes que, sin embargo, no se distinguen en el cálculo de la nueva lista *abierta* que, por lo tanto, contendrá únicamente los nodos $(3, 5)$ y $(2, 4)$ con la misma puntuación $h(\cdot) = 7$. A continuación, vuelven a expandirse hasta los mejores $k = 3$ nodos en ABIERTA que, sin embargo, serán sólo dos:



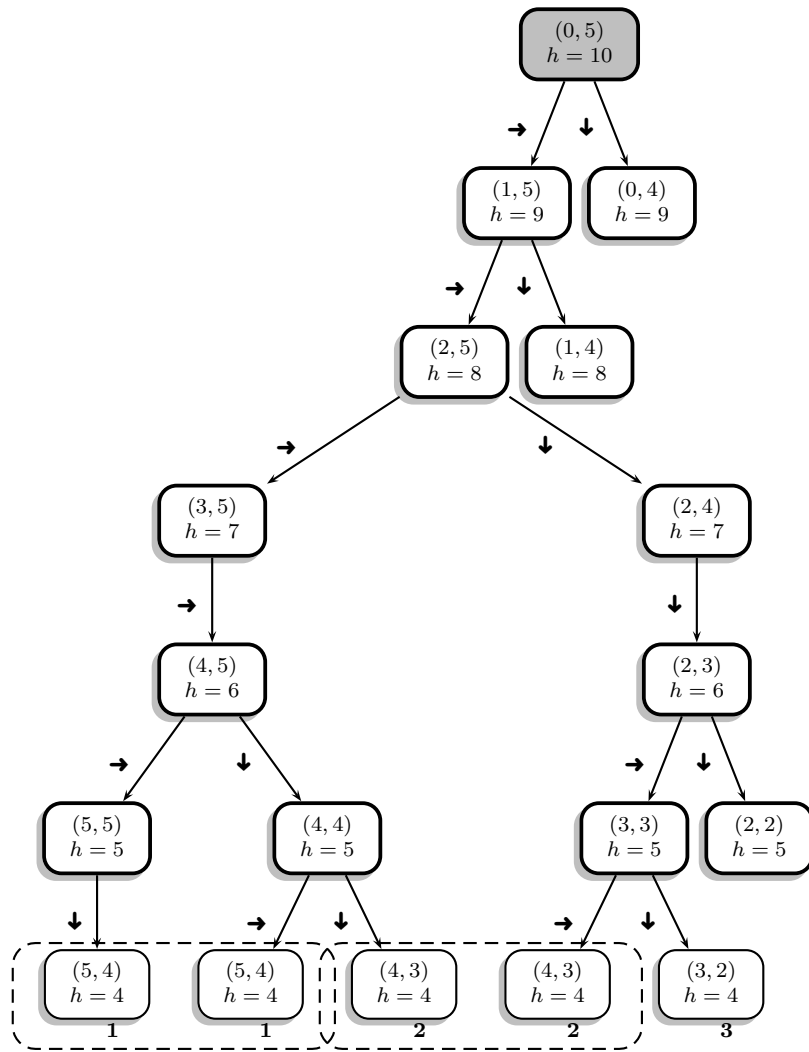
Este es un buen momento para advertir el efecto del algoritmo de búsqueda en haz. Comparándolo con el algoritmo de búsqueda en *escalada* que se empleó anteriormente, resulta ahora muy fácil advertir (incluso gráficamente), que el algoritmo de búsqueda en haz está manteniendo simultáneamente las dos mejores alternativas que hay, en la forma de caminos generados al mismo tiempo desde el nodo raíz. Ambas alternativas son, sobre la habitación, las siguientes:



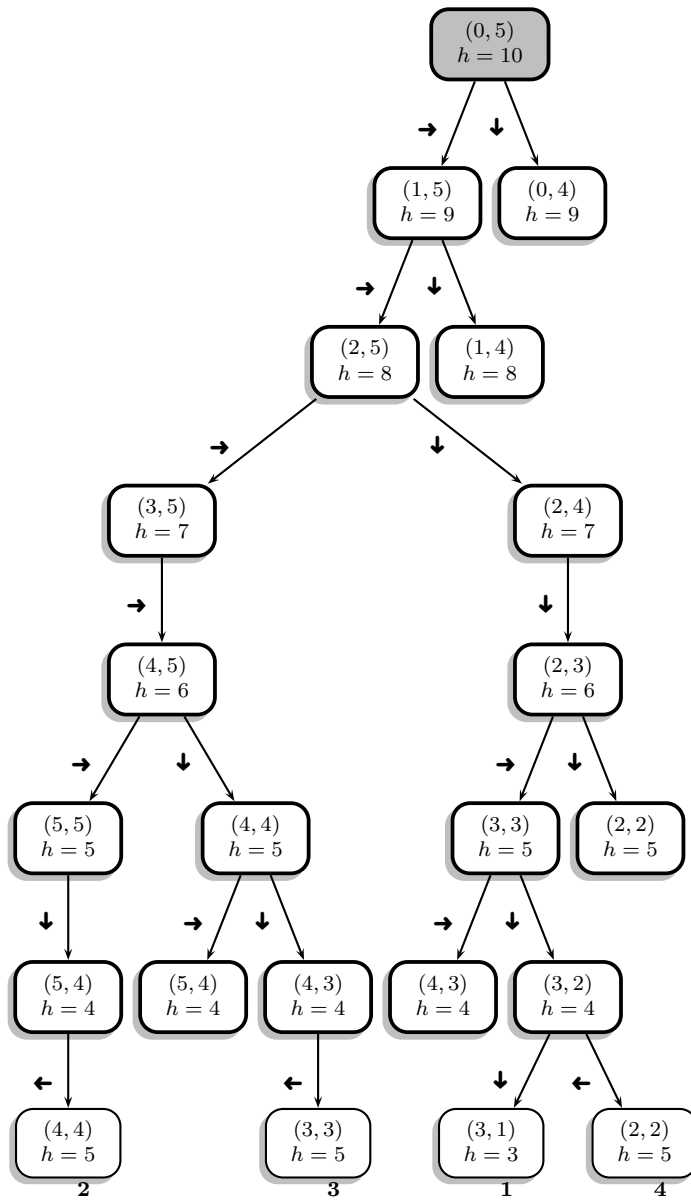
De nuevo, al expandir las $k = 2$ mejores alternativas se tiene:



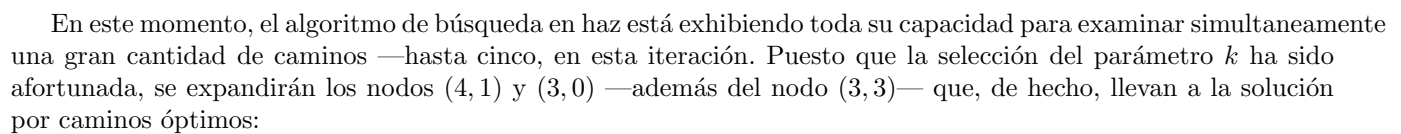
La ordenación en ABIERTA mostrada con los números debajo de cada nodo resulta de la ordenación de los operadores considerada en esta parte del ejercicio. Obsérvese que las dos primeras alternativas (esto es, (5,5) y (4,4)) se refieren a caminos que llevan a soluciones subóptimas —puesto que la solución óptima para llegar a (0,5) no pasa en absoluto por (4,5). Afortunadamente, usando $k = 3$ se dejará en ABIERTA uno de los sucesores que pasan por (2,3). Por lo tanto, después de ordenar los nodos en ABIERTA se expandirán todos menos el último, (2,2):

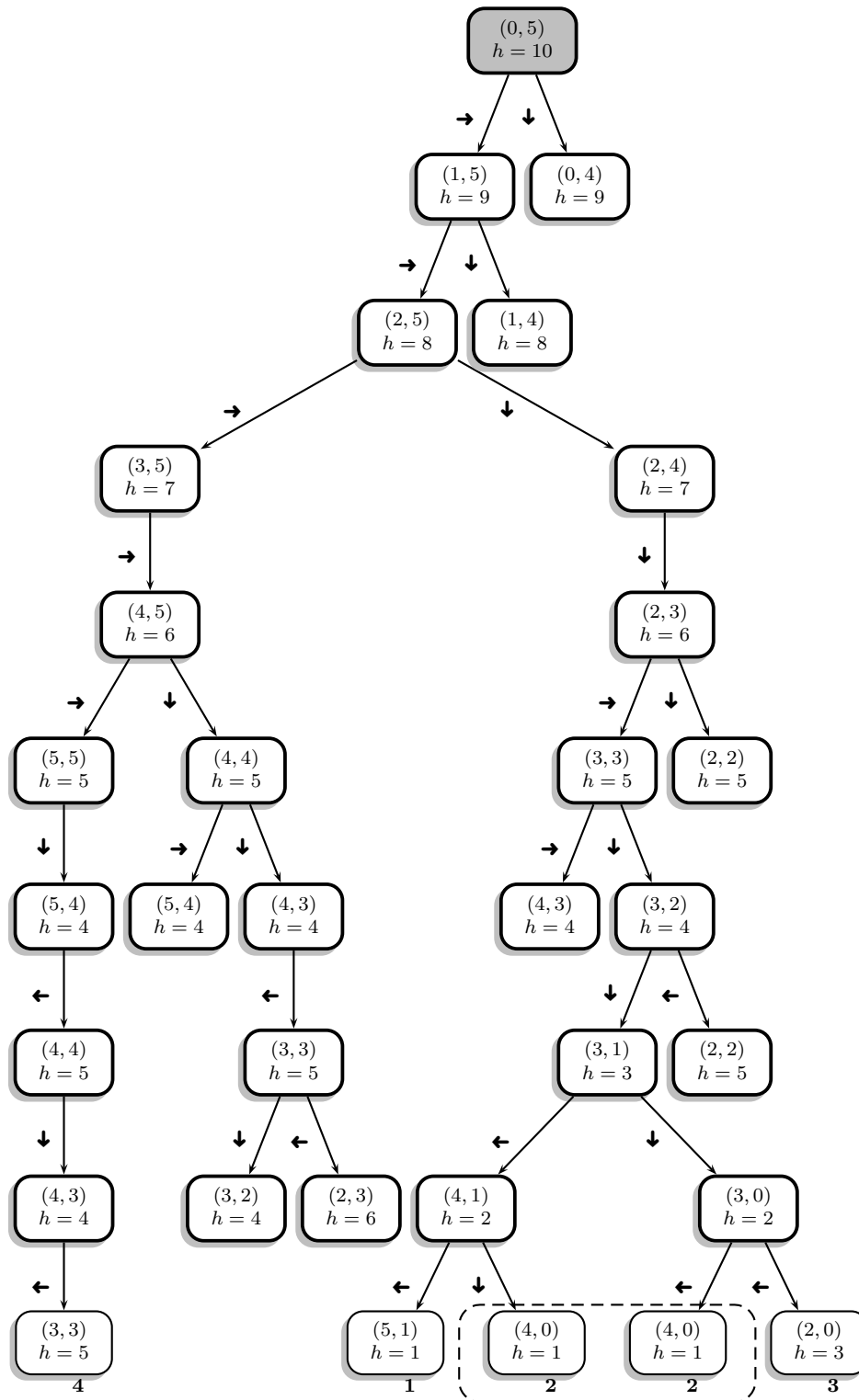


Como antes, el algoritmo de búsqueda en haz escogerá ahora arbitrariamente entre cualesquiera de los nodos repetidos para continuar a partir de ellos, siempre que estén entre los $k = 3$ mejores nodos. En este caso, todos los nodos recién generados serán expandidos puesto que se tienen, exactamente, hasta $k = 3$ nuevos sucesores distintos:



Debe advertirse que el algoritmo de búsqueda en haz está, de hecho, *revisitando* los mismos nodos en algunas ocasiones. Por ejemplo, puesto que se eligió el nodo (5, 4) más a la izquierda, su sucesor, (4, 4), es generado puesto que no estaba en el conjunto de antecesores de su padre, **aunque sí que lo estaba en el conjunto de antecesores del otro nodo** (5, 4). En cualquier caso, de nuevo vuelven a expandirse las $k = 3$ mejores alternativas, lo que deja fuera de toda consideración al nodo (2, 2)





Efectivamente, entre las $k = 3$ mejores alternativas, se encuentran cualesquiera de los caminos que superaron el punto (3, 2) —esto es, todos los nuevos nodos generados menos (3, 3) que sigue persistentemente buscando nuevos caminos por la mitad superior de la habitación. Expandiendo ahora tanto (5, 1) como (4, 0) se llega finalmente al estado meta. Puesto que el primer nodo en expandir será (5, 1), la solución encontrada por el algoritmo de búsqueda en haz **con el uso de una ordenación de nodos que no sirvió para encontrar la solución óptima con *escalada*** es: $\langle \rightarrow, \rightarrow, \downarrow, \downarrow, \rightarrow, \downarrow, \downarrow, \rightarrow, \rightarrow, \downarrow \rangle$

La expansión simultanea de un número determinado de nodos en abierta ha servido para encontrar la solución óptima

Solución ejercicio 5

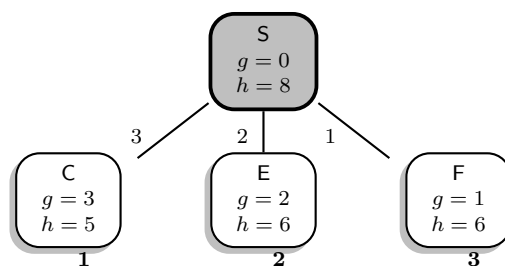
Este problema discute la aplicación de algoritmos de búsqueda al caso de la búsqueda en grafos con el uso de funciones heurísticas *admisibles*. ¿Serías capaz de explicar por qué la función heurística del problema es efectivamente admisible?

Escalada

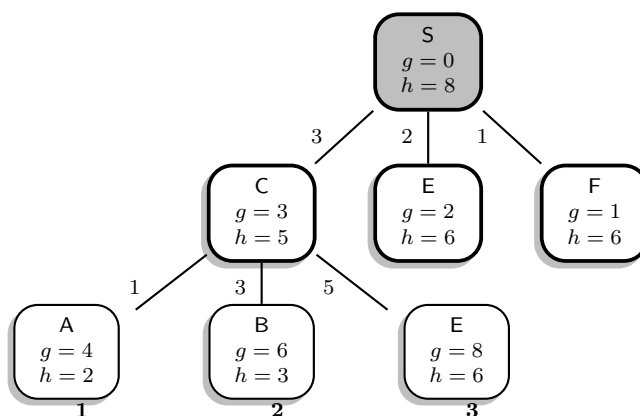
El algoritmo de búsqueda en *escalada* elegirá, en cada paso, el nodo más *prometedor* considerando para ello, únicamente, el valor de la función heurística $h(\cdot)$. Se dice que el algoritmo de búsqueda en *escalada* es un algoritmo que implementa una política irrevocable porque, después de cada expansión, escoge un único sucesor (resolviendo los empates en favor del nodo más a la izquierda) y prosigue por él, desechando inmediatamente el resto.

Para mostrar su comportamiento, considérese el grafo del enunciado. En las figuras que siguen, se ha considerado que los nodos se generan en orden lexicográfico. Naturalmente, cada arco del árbol de búsqueda desarrollado en cada caso estará etiquetado con el coste que tiene en el grafo del enunciado.

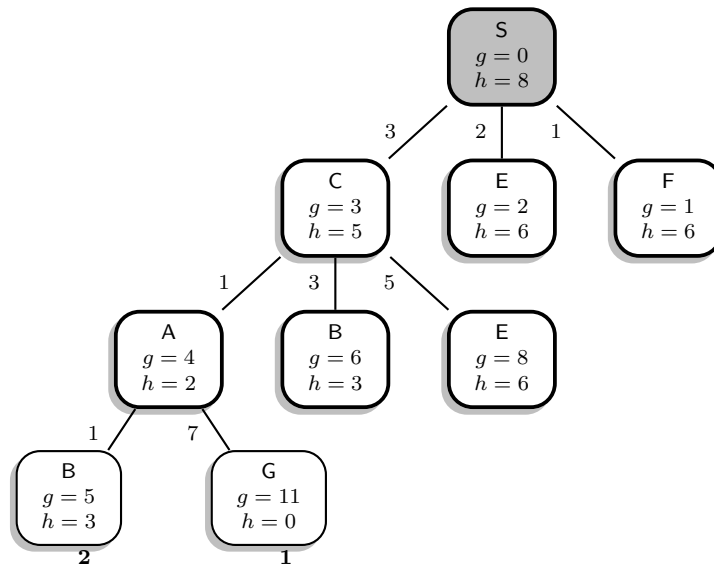
Después de la expansión del nodo inicial, S, se tiene:



de modo que, obviamente, el mejor nodo —*heurísticamente* hablando— será C, luego E y, por último, F, tal y como se indica en la figura anterior. Se procede, por lo tanto, a la expansión del nodo C resultando:



Obsérvese como el árbol de búsqueda ha encontrado una segunda forma de llegar al nodo E, a través del nodo C. Sin embargo, esta segunda manera es, efectivamente, más cara y, de hecho, el algoritmo no la considerará puesto que existe otra alternativa *heurísticamente* más barata: el nodo A.



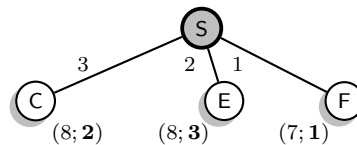
Por fin, se encuentra eventualmente la meta, el nodo G, y el algoritmo termina con la solución $\langle S, C, A, G \rangle$ con un coste igual a $g(G) = 11$ unidades.

A*

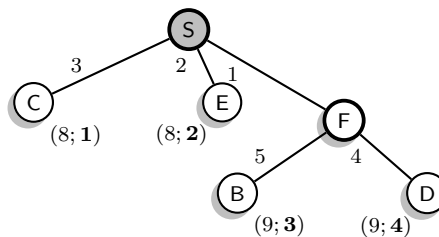
A diferencia del algoritmo de *escalada*, el algoritmo A* tendrá en cuenta el coste, $g(\cdot)$, para llegar a cada nodo y ordenará todas las alternativas disponibles en una lista ABIERTA ascendentemente por el valor $f(n) = g(n) + h(n)$. Puesto que el algoritmo se detiene cuando procede a expandir la meta (en vez de cuando la genera), puede asegurarse que el algoritmo A* terminará con la solución óptima si la función heurística que lo guía es admisible. Además, para evitar expandir el nodo varias veces, el algoritmo A* mantiene en una lista CERRADA todos los nodos expandidos hasta el momento. Así, basta con comprobar antes de la expansión de un nodo que éste no se encuentra en la lista CERRADA porque si es así, entonces ya ha sido visitado con anterioridad y puede ahorrarse esta expansión.

A propósito de esta cuestión, ¿puede asegurarse que el algoritmo de búsqueda en *escalada* nunca expandirá dos veces el mismo nodo?

En el caso del enunciado, después de expandir el nodo inicial, S queda⁶:

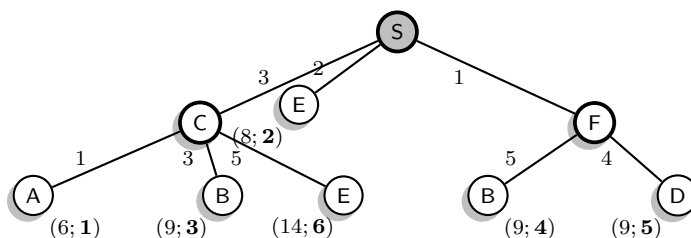


Nótese como, ahora, el nodo preferible resulta ser F, en vez de C, que fue el nodo escogido por *escalada*. El motivo es que el algoritmo A* tiene en cuenta, además, el esfuerzo para llegar a cada nodo, de modo que $f(C) = g(C) + h(C) = 8 > 7 = f(F) = g(F) + h(F)$. Por lo tanto, expandiendo el nodo F se tiene:



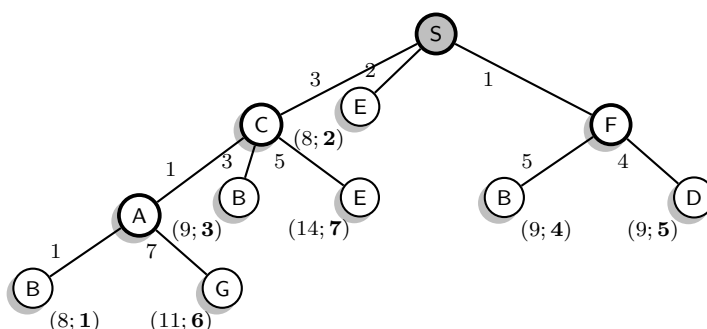
Resolviendo, como siempre, los empates en favor del nodo más a la izquierda, el mejor descendiente es el nodo C:

⁶Por motivos de espacio, los nodos se representan ahora, únicamente, con su letra y debajo se indican su valor de $f(\cdot)$ y el orden que ocupan en ABIERTA entre paréntesis



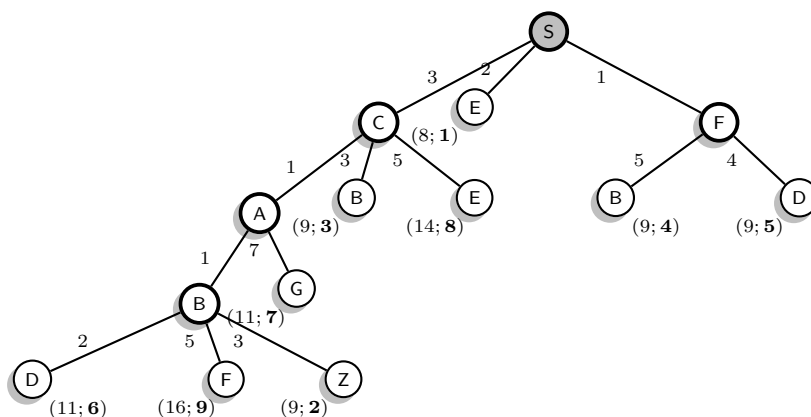
Algunos podrían pensar que un refinamiento de la codificación del algoritmo A* consiste en no introducir el nodo E en ABIERTA que se acaba de generar, puesto que tiene una evaluación $f(E) = 14$ unidades, notablemente superior que las 8 unidades con que había sido generado hace dos pasos. Sin embargo, ello nos obligaría a revisar la lista ABIERTA cada vez que se quiere introducir algo en ella. En su lugar, es más fácil insertar el nuevo nodo en ABIERTA sin comprobar si está o no repetido. Cuando se expanda la primera copia del nodo E (que necesariamente lo hará antes que ésta, puesto que tiene 8 unidades de coste, menor que las 14 de esta segunda), éste pasará a CERRADA, de modo que no se expandirá la segunda⁷.

Como quiera que ocurriera con el algoritmo de *escalada*, será nuevamente el nodo A el que primero genere la meta, G, puesto que $f(A)$ es la menor de las evaluaciones de la lista *abierta* que ahora se tiene cuyos contenidos ordenados son: {A, E, B, B, D, E}



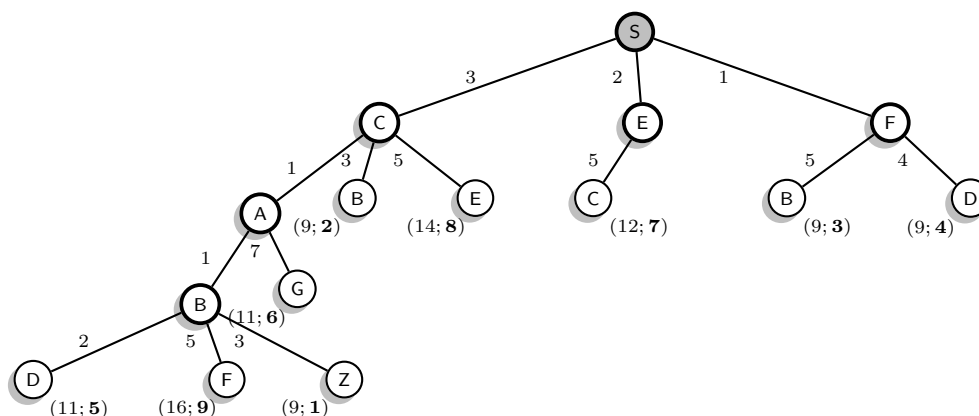
Resulta notable advertir ahora que el algoritmo A* ha descubierto que para llegar al nodo B, lo más barato es hacerlo por el camino $\langle S, C, A, B \rangle$ que por cualesquiera de las otras alternativas que, aunque consisten en menos nodos, tienen un coste $g(\cdot)$ mayor.

Nótese, además, que aunque se ha generado la meta, G, el algoritmo no se detiene puesto que aún existen varias oportunidades de llegar al mismo nodo final a través de otros caminos más baratos. Por ejemplo, ¿quién sabe si ahora expandiendo el primer nodo en ABIERTA, el nodo B, resulta que llegamos de un solo golpe al nodo G con un coste menor a 11 unidades? Sólo hay una forma de averiguarlo, expandiéndolo:



Puesto que todos los nodos recién generados superan el valor del nodo E al que se llega directamente desde la raíz, éste se convierte en el primero en *abierta* y, después de su expansión resulta:

⁷El mismo comentario aplica a la **segunda** copia que ahora se ha generado del nodo B, sólo que, en este caso, ¡la nueva es más barata que la anterior!

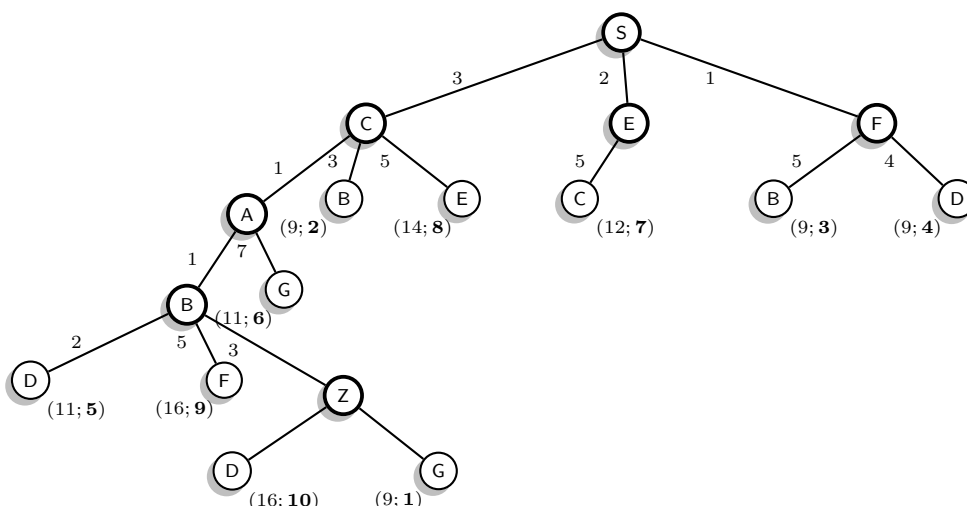


Convendría entender bien como se ha ordenado incrementalmente la lista ABIERTA y los efectos que origina su uso. Para entenderlo mejor se muestran a continuación los contenidos de ABIERTA en este mismo instante:

Posición	Contenido	$f(\cdot)$
①	$\langle S, C, A, B, Z \rangle$	9
②	$\langle S, C, B \rangle$	9
③	$\langle S, F, B \rangle$	9
④	$\langle S, F, D \rangle$	9
⑤	$\langle S, C, A, B, D \rangle$	11
⑥	$\langle S, C, A, G \rangle$	11
⑦	$\langle S, E, C \rangle$	12
⑧	$\langle S, C, E \rangle$	14
⑨	$\langle S, C, A, B, F \rangle$	16

Cómo puede verse, el primer nodo en ABIERTA tiene siempre la menor de las puntuaciones $f(\cdot)$ o, en otras palabras, es la mejor de las alternativas disponibles. Conviene recordar, asimismo, que los empates se resuelven siempre a favor del nodo más a la izquierda o, en otras palabras, en favor del nodo que se encontraría antes en un recorrido *en profundidad*. Sin embargo, la observación más importante es que la lista ABIERTA almacena **caminos** y no nodos (a pesar de que con frecuencia se diga así). Por ello, si se genera el mismo nodo más de una vez, éste es insertado igualmente. Obviamente las copias de cualquier nodo que ya ha sido expandido, no se expandirán nunca puesto que el nodo del que son copia ya estará entonces en la lista CERRADA.

Por lo tanto, ahora se expande el primer nodo, el nodo Z:

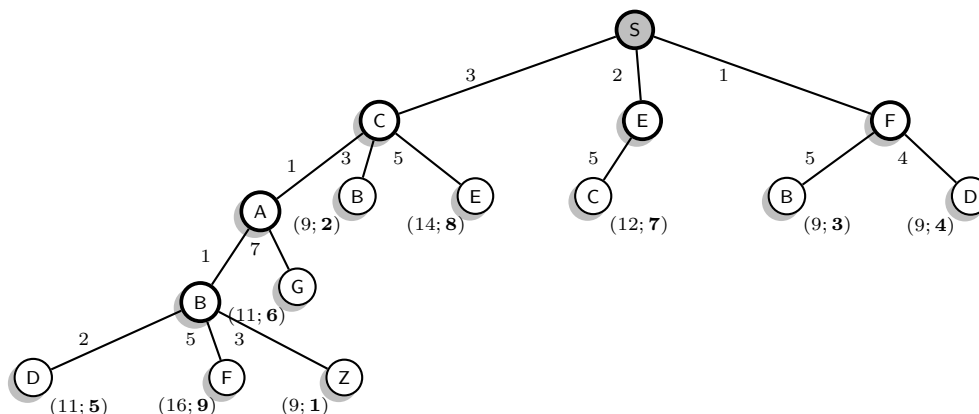


¡Por segunda vez se ha generado el nodo final, G! pero esta vez con un valor de $f(\cdot)$, 9, mejor que el que tenía antes, 11, de modo que el nuevo camino se sitúa en ABIERTA por delante y, de hecho, se coloca en primera posición. Por lo tanto, al escoger ahora el primer nodo en ABIERTA para su expansión, se advierte que se trata del nodo final o meta y, por lo tanto, se acaba la búsqueda con una solución *óptima*: $\langle S, C, A, B, Z, G \rangle$ con un coste de 9 unidades, mejor que la solución encontrada por el algoritmo de búsqueda en *escalada*.

IDA*

El algoritmo IDA*, a diferencia del A* expande en profundidad el árbol de búsqueda desde S hasta que el valor $f(\cdot)$ de sus nodos hoja exceden, por primera vez, un umbral η . Para garantizar la *admisibilidad* del algoritmo, el valor del umbral η se inicializa a $\eta_1 = h(S) = 8$.

Iteración 1: $\eta_1 = 8$



Puesto que este árbol se genera en profundidad, la ocupación de memoria será lineal y, en un sólo paso se ha llegado al mismo árbol de búsqueda que desarrolló el A* en 6 iteraciones —y, además, con ocupación de memoria exponencial.

Como quiera que en esta iteración no se ha encontrado la meta, G, será preciso desarrollar otra iteración pero con un umbral η distinto. Para garantizar, de nuevo, la *admisibilidad* del algoritmo, se toma como nuevo límite el mínimo exceso en $f(\cdot)$ cometido por todos los nodos hoja generados en esta iteración⁸:

$$\begin{aligned}\eta_2 &= \min\{f(Z), f(B), f(B), f(D), f(D), f(G), f(C), f(E), f(F)\} \\ &= \min\{9, 9, 9, 9, 11, 11, 12, 14, 16\} \\ &= 9\end{aligned}$$

Iteración 2: $\eta_2 = 9$

En esta iteración, el algoritmo IDA* generará, nuevamente (¡desde el principio!), el árbol de búsqueda que se obtiene al expandir todos los nodos cuyo valor $f(\cdot)$ no excede el nuevo umbral, $\eta_2 = 9$. De esta suerte, resulta obvio que se encontrarán todos los caminos que pueden desarrollarse con un coste menor o igual a 9 unidades —puesto que no expandirán los nodos con un valor $f(n) = g(n) + h(n)$ mayor.

Al hacerlo generará, exactamente, el mismo árbol de la última iteración del A*, puesto que **se detendrá en cuanto genera por primera vez el nodo final o meta, G**. Puesto que el nodo G tiene ahora un coste menor o igual que el umbral empleado en esta iteración, puede asegurarse que el camino encontrado es, entonces, óptimo: $\langle S, C, A, B, Z, G \rangle$, con un coste de 9 unidades.

Solución ejercicio 6

En vez de considerar la forma de alcanzar un único estado inicial, s , a partir de un número cualquiera de estados finales o meta, $\Gamma : \{t_1, t_2, \dots, t_n\}$, considérese primero el caso más simple: desde un único estado final. Por ejemplo, desde el t_1 .

Aplicando el algoritmo A*, basta con iniciar la lista ABIERTA con el nodo t_1 hasta que finalmente se encuentre el nodo inicial, s . Puesto que el enunciado advierte que sólo debe considerarse el caso de grafos no dirigidos, es obvio que puede realizarse la búsqueda *hacia atrás* de esta forma sin problemas —porque en los grafos no dirigidos es posible transitar desde un estado a otro adyacente en cualquier sentido.

Puesto que el algoritmo A* se detiene cuando va a expandir la solución (en este caso, el nodo inicial), se sabe que ésta tiene el coste óptimo, independientemente del resto de alternativas que aún queden pendientes de expandir en la lista ABIERTA. Por lo tanto, sería posible popular inicialmente la lista ABIERTA con un conjunto arbitrario de nodos meta, t_i y ejecutar normalmente el algoritmo A* a continuación: en cada paso se tomaría el primer nodo de

⁸Se advierte, nuevamente, que cada nodo que aparece en la siguiente expresión representa, en realidad, caminos diferentes que llevan hasta cada uno de ellos

ABIERTA, al que se habría llegado desde alguno de los nodos meta t_i , y se evaluaría tomando como valor heurístico la estimación hasta llegar al nodo inicial (de modo que, como requería el enunciado, no es necesario reformular la función heurística), y así sucesivamente, hasta que por fin se va a expandir el nodo s .

La solución al problema consiste, por lo tanto, en dos observaciones:

- Si el espacio de estados es un grafo no dirigido, el algoritmo A^* puede expandir también el árbol de búsqueda *hacia atrás*: desde una meta cualquiera, t_i hasta el nodo inicial, s .
- Si se popula la lista ABIERTA con un número arbitrario de nodos, el A^* termina con el camino óptimo (dado que la función heurística sea admisible) hasta la solución desde uno cualquiera de ellos.