



Memoria

Práctica 2

**Curso Heurística y Optimización
Grado en Ingeniería Informática
Curso 2021-22**

<i>Javier Chico García</i>	<i>100429138</i>	<i>Grupo 82</i>	<i>100429138@alumnos.uc3m.es</i>
<i>Mario García Rodríguez</i>	<i>100428982</i>	<i>Grupo 82</i>	<i>100428982@alumnos.uc3m.es</i>

Índice

Introducción	3
Parte 1	3
Modelización del problema	3
Implementación del modelo	4
Resolución y análisis de los casos de prueba	5
Parte 2	6
Modelización del problema	6
Conjunto de estados	6
Conjunto de operadores	7
Estado inicial	7
Estado final o meta	7
Implementación de A*	8
Generación de estados sucesores	8
Heurísticas	9
Resolución de los casos de prueba	11
Análisis de los casos de prueba	14
Conclusiones generales de la práctica	14

Introducción

En el presente documento se describen los modelos realizados para las partes 1 y 2 de los enunciados de esta práctica, así como las decisiones tomadas que influyeron en el desarrollo final del modelado y el desarrollo del código. Además se incluirá un análisis de los resultados obtenidos para la parte 1 y una explicación acerca del razonamiento utilizado para la implementación del algoritmo para la parte 2.

El documento finalizará con unas conclusiones generales que reflejarán las reflexiones finales acerca del contenido y los problemas encontrados.

Parte 1

En esta sección de la práctica se lleva a cabo el desarrollo de la parte 1 de la práctica. En ella, se nos encomienda la tarea de modelizar, con CSP, un problema de gestión de colocación y reparto de contenedores de un buque.

Modelización del problema

El problema presentado se resolverá utilizando CSP y por ello lo definiremos como (X, D, C) dónde:

$$X = \{x_i\}_{i=1}^n \text{ dónde } x_i \in \mathbb{N}, \text{ id de cada contenedor}$$

$$D = \{D_S, D_R\}$$

s_i : pila d_i : profundidad

$D_S = \{v_1, v_2, v_3, \dots, v_n\}$ celdas del mapa de carga cuyos valores son N ó E , dónde $v_i = (s_i, d_i)$

$D_R = \{v_1, v_2, v_3, \dots, v_n\}$ celdas del mapa de carga cuyos valores son E , dónde $v_i = (s_i, d_i)$

C = conjunto de restricciones del problema:

1-. Dos variables no pueden tener el mismo valor.

$$R_{ij} = \{(v_i, v_j) \mid \forall i, j \in X; \forall v_i, v_j \in D \ i \neq j, v_i \neq v_j\}$$

2-. Un contenedor siempre debe tener a otro debajo que lo soporte o una casilla X en su defecto

d_{max} : profundidad máxima de una pila sin contar las casillas X

$$R_{ij} = \{(v_i, v_j) \mid \forall i, j \in X; \forall v_i, v_j \in D \ i \neq j, d_i + 1 = d_j, s_i = s_j \text{ ó } d_i + 1 = d_{max}\}$$

3-. Un contenedor debe tener debajo contenedores con un índice de puerto igual o mayor al suyo

P_i = Puerto del contenedor i – ésimo

$$R_{i...n} = \{(v_i, v_{i+1}, \dots, v_{i-n}) \mid \forall i, j \in X; \forall v_i, v_j \in D \ i \neq j, p_i = p_j, P_i > P_j\}$$

Implementación del modelo

El modelo será implementado utilizando el lenguaje de programación *python* con el añadido de las librerías *sys*, utilizada para recoger los datos de entrada y *python_constraint*, utilizada para modelar el problema *CSP* en sí, añadiendo los conjuntos definidos previamente.

Para empezar, creamos una clase *Problem*. En su método *init* será donde inicializamos todos los atributos necesarios para resolver nuestro problema. Recogemos los dos archivos de entrada y los guardamos en atributos de nuestra clase problema. Nuestras variables serán los identificadores de cada contenedor que, irán de 1 a n y las guardaremos como otro atributo: *vars* en nuestra clase. Además guardaremos el resto de la información (tipo del contenedor y el puerto destinado) en otro atributo: *content*.

Gracias a esto, podemos restringir los dominios de cada variable haciendo uso del mapa obtenido. Ahora, los contenedores refrigerados sólo pueden tomar valores de casillas energéticas y los contenedores estándar pueden tomar valores de casillas energéticas y casillas normales. Con estos dos dominios definidos, añadimos las variables a nuestro problema con sus respectivos dominios.

Continuamos añadiendo las *constraints* o restricciones necesarias para resolver el problema. Creamos métodos para cada una de estas restricciones con los posibles valores de cada variable como parámetro.

Los métodos implementados para establecer estas restricciones al problema son los siguientes:


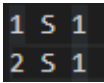
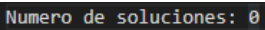
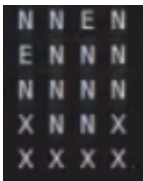
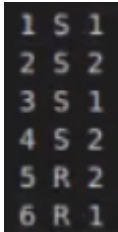
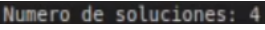
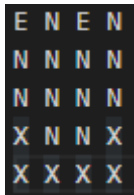
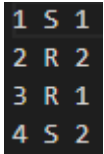
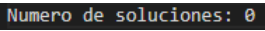
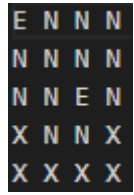
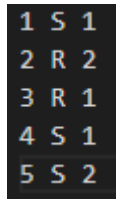
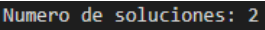
- *base()*: Recibe los valores de todas las variables y se iterará sobre ellos. Si el contenedor- i no tiene otro contenedor debajo o está en la profundidad máxima (encima de una X) ese valor del contenedor no es válido, el método *base()* devolverá *False*. Si hay algún valor posible, se devolverá *True*.
- *ports_order()*: Recibe los valores de todas las variables y se iterará sobre ellos. Si el contenedor- i tiene un puerto destinado menor que el puerto destinado que el contenedor- j y la profundidad del contenedor- i es mayor que la del contenedor- j , este valor del contenedor no es válido, el método *ports_order()* devolverá *False*. Si hay algún valor posible, se devolverá *True*.

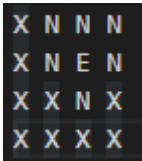
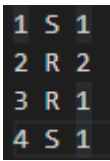
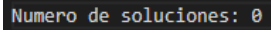
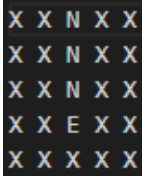
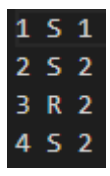
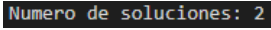
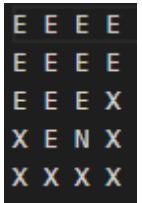
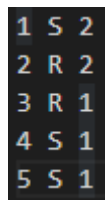
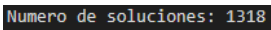
Resolución y análisis de los casos de prueba

Para la resolución de nuestro problema, ejecutaremos el problema pasándole un archivo que contenga el mapa y otro para los contenedores válidos. Nuestro script creará un objeto `a = Problema()` y hará uso de su método `write_solution()` para obtener el archivo que relata las posibles soluciones del problema.

En cuanto a la parte del análisis, creamos un setup de contenedores y mapas que construyan todos nuestros casos de prueba. Además, nuestro fichero `CSP-calls.sh` contiene comandos que ejecutarán el programa con los diferentes archivos de los contenedores y los mapas. Con esto, podemos analizar los resultados que obtenemos de la ejecución de estos comandos para los siguientes casos de prueba:

Cabe destacar que todos estos casos de prueba están definidos en la carpeta de tests de la parte-1 de nuestra entrega y que el identificador del caso de prueba incluido en los casos de prueba definidos es el mismo que el incluido en cada caso de prueba del directorio CSP-tests. Hemos decidido que estos casos de prueba son justos y necesarios para comprobar el correcto funcionamiento de nuestra implementación.

Caso a probar	Mapa	Cont	Output	Conclusiones
1-. Más contenedores que casillas en el mapa.				Indica correctamente que no hay solución
2-. Mapa más grande, más contenedores y casillas electrificadas				Se obtienen soluciones que son correctas amparadas por nuestra corrección manual
3. Mapa anterior irresoluble por la falta de contenedores para llegar a casillas electrificadas				Se comprueba que la validación de contenedores no volando tiene prioridad frente a los eléctricos
4-. Problema con pocas soluciones.				Ambas soluciones son correctas, comprobadas manualmente

5-. Problema irresoluble contenedores normales destinados al puerto 2				Es correcto, no existe solución, puesto que el contenedor 2 no tiene posibilidad de posicionarse debajo de uno destinado a P1
Problema con bodega columna				Las soluciones son correctas para este mapa atípico lleno de posiciones inválidas X
Problema complejo con muchas soluciones				Permite obtener todas las soluciones en menos de un minuto

Realizados todos estos análisis de los casos de prueba, podemos observar que nuestra implementación encuentra soluciones en un tiempo significativamente pequeño mientras no haya más de una decena de contenedores y el mapa sea similar o más pequeño al que da el propio enunciado. Además, concluimos que esta es capaz también de proporcionar una solución a casos extremos y específicos, como los mapas columna y, que incluso reconoce aquellos problemas que no tienen solución.

Con esto, podemos dar por concluida nuestra parte 1 de la práctica al haber conseguido implementar una solución al problema planteado con CSP correctamente.

Parte 2

Modelización del problema

Conjunto de estados

Los estados en este problema están definidos por una tupla conformada por las características de posición y ubicación en la carga de cada container y B , que representa la posición del barco que ocupará el último elemento de esta tupla.

$$((P, M), (P, M), \dots, (P, M), B)$$

B : posición del barco, tomará distintos valores:

- 0, si se encuentra en la bahía
- 1, si se encuentra en el puerto 1
- 2, si se encuentra en el puerto 2

P : posición del contenedor, tomará distintos valores:

- 0, si el contenedor se encuentra en la bahía
- 1, si el contenedor se encuentra en el puerto 1
- 2, si el contenedor se encuentra en el puerto 2
- 3, si el contenedor se encuentra en el barco

M : ubicación del contenedor dentro del mapa de carga del barco, si no está en el barco tomará valor *None*, esta ubicación se representará con una tupla del tipo

$$M = (S, D)$$

S : pila en la que está ubicado el contenedor

D : profundidad a la que está ubicado el contenedor

Conjunto de operadores

Para obtener los operadores, extraemos del problema las posibles acciones que se pueden realizar.

Cabe destacar que, tal y como dice el enunciado en la siguiente frase: Además, hay que considerar los viajes del barco, que puede navegar del puerto inicial al puerto 1, y del puerto 1 al puerto 2. Decidimos que el barco solo puede hacer los viajes estipulados en el enunciado, de la bahía al puerto 1 y del puerto 1 al puerto 2 y basamos así nuestro conjunto de operadores y nuestra implementación.

Operador	Precondiciones	Efectos
Cargar(x, m)	$P_x = b$	$P_x = 3, M_x = m$
Descargar(x, p)	$b \neq 0, P_x = 3$	$P_x = b, M_x = None$
Navegar(p)	$b \neq 2$	$b = b + 1$

Estado inicial

El estado inicial será aquel en el que el barco se encuentra en la bahía y todos los contenedores también. La representación de este estado con nuestra tupla general será:

$$(P, (S, D), P, (S, D), P, (S, D), \dots, P, (S, D), B)$$

$$(0, None, 0, None, 0, None, \dots, 0, None, 0)$$

Estado final o meta

El estado final será aquel que cumpla la condición de que todos los contenedores se encuentren en el puerto al que debían ser destinados. Queda definido implícitamente el estado meta de esta forma.

Implementación de A*

La implementación de A* se basa en un bucle que se está ejecutando siempre que tenga nodos que expandir. Se compone de tres métodos o fases principales:

- Expandir el primer nodo de la lista de abiertos (con mejor valor de $f() = g + h$) y añadirla a la de cerrados (expandidos).
- Comprobar si el nodo expandido cumple las condiciones de ser solución.
- Generar los sucesores del nodo expandido, añadiéndolos a la lista de abiertos.

Con esto, nuestra implementación irá escogiendo los mejores nodos conforme a si tienen menos valor de $f()$ para alcanzar una solución óptima.

Generación de estados sucesores

Para la generación de estados sucesores, nuestra implementación ha requerido de los operadores detallados anteriormente. Con ellos, hemos podido crear un método que, para cada nodo, comprobará cada precondition de cada uno de los operadores para poder generar los estados o nodos sucesores.

Este método es *get_children(node)*, que recibe el nodo de ASTAR de la lista de abiertos y hará las siguientes comprobaciones con sus respectivos efectos:

- *cargar(contenedor, nodo)*: este método recibe el contenedor como parámetro para cargarlo en el barco y el nodo padre, para que genere todos los posibles estados al cargar el contenedor en las posibles posiciones libres del barco.
La precondition a cumplir es que el contenedor esté en la misma posición del barco. Si es así, el método *cargar* obtendrá las posibles posiciones del contenedor en el barco y generará sus respectivos estados.
- *descargar(contenedor, nodo)*: este método recibe el contenedor como parámetro para cargarlo en el barco y el nodo padre para generar el estado en el que se descarga este contenedor en el puerto donde está situado el barco.
Las precondiciones a cumplir son: que el contenedor se encuentre en la posición 3, equivalente a estar en el barco; que el contenedor se encuentre con menos profundidad que ningún otro contenedor en la misma pila que este; y que el barco no se encuentre en la bahía. Si todas estas precondiciones se cumplen, el método *descargar()* generará el nodo sucesor al nodo padre *nodo* donde ya se ha descargado el contenedor en cuestión.
- *navegar(nodo)*: este método recibe el nodo padre para generar su nodo sucesor, que el barco se encuentre ahora en el siguiente puerto. La precondition necesaria para efectuar este

operador es la de que el barco no se encuentre en la posición del segundo puerto, es decir, que en el estado del nodo, el valor del barco es distinto de 2.

Heurísticas

h1: no_colocados

Con el objetivo de establecer unas heurísticas para resolver nuestro problema óptimamente con A*, en una primera aproximación, nos hemos apoyado en el problema del *8-puzzle*, al cual se le puede establecer una heurística admisible al comparar para cada nodo cuántas piezas están bien colocadas en el puzzle.

Al hacer la comparación de este problema con el de cargar y descargar contenedores en diferentes puertos, podemos obtener una heurística que contabiliza cuántos contenedores no se encuentran en su destino final.

Esta heurística se calculará cada vez que se cree un nuevo nodo estado y se sumará al coste real g para obtener nuestro deseado valor f para la elección de mejor nodo en el algoritmo A*.

Por tanto, habiendo dicho todo esto, podemos definir nuestra heurística como:

CN: contenedores que no están en su puerto destino y que no estén en el barco

CB: contenedores en el barco pero no en su puerto destino

$$h_1 = 25 \cdot CN + 15 \cdot CB$$

Ejemplo

Si este fuese el contenido de *cont_test*:

1 S 2
2 R 1

(0, None , 0, None , 0)

En este nodo, el valor de h_1 es de 50, ya que ninguno de los dos contenedores se encuentra en su puerto destino, cualquiera que este sea.

(1, None , 3, (0, 1) , 0)

Mientras que este nodo, tendría una h_1 de 15, ya que el número de contenedores que no están en su puerto destino es de 1, y es el contenedor número 2 que está en el barco. Este nodo tiene una heurística menor ya que está más cerca de la solución.

Estos factores multiplicativos que se encuentran en la heurística h_1 se obtienen de calcular cuánto le costará realmente al problema: llevar a un contenedor que no está en su puerto destino a su puerto destino 25, 10 de cargar el contenedor en el barco y 15 de descargarlo sin sobrestimar el coste real; y descargar un contenedor que está en el barco 15.

h2: recorrido

Para hallar una heurística más admisible procedemos al método de obtención de heurísticas mediante la relajación de las precondiciones de los operadores.

Revisión de los operadores del problema no relajado

Operador	Precondiciones	Efectos
Cargar(x, m)	$P_x = b$	$P_x = 3, M_x = m$
Descargar(x, p)	$b \neq 0, P_x = 3$	$P_x = b, M_x = None$
Navegar(p)	$b \neq 2$	$b = b + 1$

Relajamos el problema eliminando la precondición de que el barco esté en la misma posición que el container para cargarlo ($P_x = b$).

Resolver el problema de esta forma es trivial y se puede ver con este ejemplo:

<i>Cont:</i>	<i>Map:</i>
1S1	N N
2S2	N N

acción = 0 (contador que ayudará a determinar la heurística)

1.- El barco navega al puerto 1	acción +=1
2.- El barco carga el contenedor 1 en (0,1)	acción +=1
3.- El barco descarga el contenedor en el puerto 1	acción +=1
4.- El barco navega al puerto 2	acción +=1
5.- El barco carga el contenedor 2 en (0,1)	acción +=1
6.- El barco descarga el contenedor en el puerto 2	acción +=1

Tras esto queda claro que las acciones son 6, y si le damos sentido a este número podemos determinar que podemos dividir la heurística en 3 partes:

$$h = 2(\text{acciones cargar}) + 2(\text{acciones descargar}) + 2(\text{acciones navegar})$$

De aquí deducimos que la heurística tendrá que tener en cuenta estas acciones, y si la acabamos puliendo llegamos a la heurística final

CP: contenedores por cargar en ese lugar que no estén destinados a ese puerto

CD: contenedores por descargar de la bodega del barco

PR: puertos restantes, es decir $2 - b$, siendo b la posición del barco

$$h_2 = 2 \cdot 12,5 \cdot CP + 15 \cdot CD + PR \cdot 3500$$

Esta función heurística tiene un factor multiplicativo de 2 en CP para diferenciar aquellos estados donde acabo de cargar un contenedor de un puerto y estoy más cerca de la solución aunque ahora se contabilice en contenedores por descargar. Los otros factores multiplicativos aumentan la relevancia general de la heurística sin sobreestimar el coste.


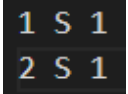
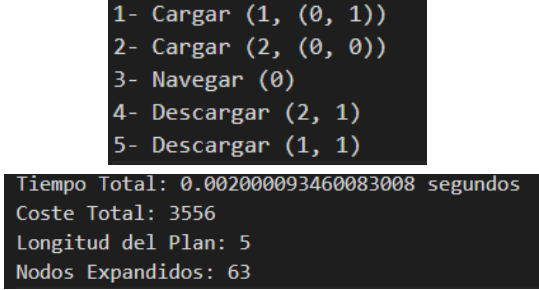
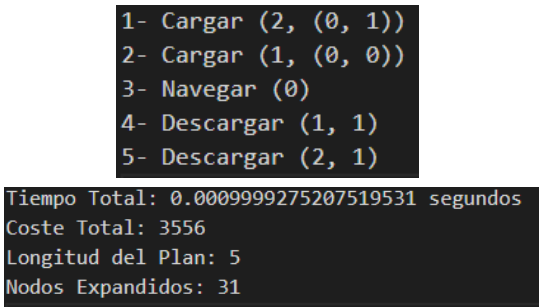
Ejemplo con 2 contenedores que tienen distinto puerto destino en un mapa genérico 2x2

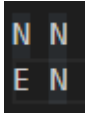
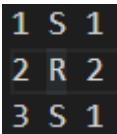
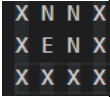
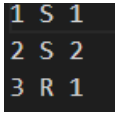
(0, None , 0, None , 0)	$h = 2 \cdot 12,5 \cdot 2 + 15 \cdot 0 + (2 - 0) \cdot 3500 = 7050$
(3, (0, 1), 0, None , 0)	$h = 2 \cdot 12,5 \cdot 1 + 15 \cdot 1 + (2 - 0) \cdot 3500 = 7040$
(3, (0, 1), 3, (1, 1)), 0)	$h = 2 \cdot 12,5 \cdot 0 + 15 \cdot 2 + (2 - 0) \cdot 3500 = 7030$
(3, (0, 1), 3, (1, 1), 1)	$h = 2 \cdot 12,5 \cdot 0 + 15 \cdot 2 + (2 - 1) \cdot 3500 = 3530$
(1, None, 3, (1, 1), 1)	$h = 2 \cdot 12,5 \cdot 0 + 15 \cdot 1 + (2 - 1) \cdot 3500 = 3515$
(1, None, 3, (1, 1), 2)	$h = 2 \cdot 12,5 \cdot 0 + 15 \cdot 1 + (2 - 2) \cdot 3500 = 15$
(1, None, 2, None, 2)	$h = 2 \cdot 12,5 \cdot 0 + 15 \cdot 0 + (2 - 2) \cdot 3500 = 0$

Y tal y como debe ser, $h=0$ en el nodo final

Resolución de los casos de prueba

Todos los casos de prueba fueron ejecutados de forma independiente y los mapas 5 y 7 no fueron incluidos para la heurística 1 en el .sh debido a que no devuelven resultado en un tiempo razonable.

Caso a probar	Mapa	Cont	h	Output	Conclusiones
1-. Mapa básico 2x2			h_1		Se encuentra la solución rápidamente y no existe aún gran diferencia entre heurísticas
			h_2		

2-. Mapa básico con eléctricos			h_1	<pre> 1- Cargar (3, (1, 1)) 2- Cargar (2, (0, 1)) 3- Cargar (1, (0, 0)) 4- Navegar (0) 5- Descargar (1, 1) 6- Descargar (3, 1) 7- Navegar (1) 8- Descargar (2, 2) Tiempo Total: 0.3340754508972168 segundos Coste Total: 7087 Longitud del Plan: 8 Nodos Expandidos: 1114 </pre>	El contenedor eléctrico se coloca correctamente y se empieza a apreciar la diferencia entre ambas heurísticas
			h_2	<pre> 1- Cargar (2, (0, 1)) 2- Cargar (1, (0, 0)) 3- Cargar (3, (1, 1)) 4- Navegar (0) 5- Descargar (1, 1) 6- Descargar (3, 1) 7- Navegar (1) 8- Descargar (2, 2) Tiempo Total: 0.011003255844116211 segundos Coste Total: 7087 Longitud del Plan: 8 Nodos Expandidos: 200 </pre>	
3-. Mapa bodega pequeña			h_1	<pre> 1- Cargar (3, (1, 1)) 2- Cargar (1, (1, 0)) 3- Cargar (2, (2, 1)) 4- Navegar (0) 5- Descargar (1, 1) 6- Descargar (3, 1) 7- Navegar (1) 8- Descargar (2, 2) Tiempo Total: 0.30207395553588867 segundos Coste Total: 7096 Longitud del Plan: 8 Nodos Expandidos: 1060 </pre>	La solución se desarrolla correctamente respetando las casillas X
			h_2	<pre> 1- Cargar (3, (1, 1)) 2- Cargar (1, (1, 0)) 3- Cargar (2, (2, 1)) 4- Navegar (0) 5- Descargar (1, 1) 6- Descargar (3, 1) 7- Navegar (1) 8- Descargar (2, 2) Tiempo Total: 0.01100301742553711 segundos Coste Total: 7096 Longitud del Plan: 8 Nodos Expandidos: 199 </pre>	

4-. Ejemplo no solución	<div>E N</div>	<div>1 S 1 2 R 2</div>	h_1 h_2	<div>No hay ninguna solución para este problema</div> <div>Tiempo Total: 0.00099945068359375 segundos Coste Total: - Longitud del Plan: - Nodos Expandidos: 19</div> <div>No hay ninguna solución para este problema</div> <div>Tiempo Total: 0.0 segundos Coste Total: - Longitud del Plan: - Nodos Expandidos: 19</div>	<p>No se encuentra solución como es esperado.</p> <p>Cabe destacar que la segunda heurística encuentra este caso casi instantáneamente</p>
5-. Mapa moderado	<div>E N E N N N</div>	<div>1 S 1 2 R 2 3 S 2 4 R 1</div>	h_1 h_2	<div>No se generan ficheros</div> <div> 1- Cargar (2, (2, 0)) 2- Cargar (1, (0, 1)) 3- Cargar (4, (0, 0)) 4- Cargar (3, (1, 1)) 5- Navegar (0) 6- Descargar (4, 1) 7- Descargar (1, 1) 8- Navegar (1) 9- Descargar (3, 2) 10- Descargar (2, 2) </div> <div>Tiempo Total: 0.9422118663787842 segundos Coste Total: 7121 Longitud del Plan: 10 Nodos Expandidos: 1754</div>	<p>Como podemos observar la heurística 1 no es capaz de resolver este problema en un tiempo razonable</p>
6-. Mismo mapa con menos cont	<div>N N N E N E</div>	<div>1 S 1 2 R 2 3 S 2</div>	h_1	<div> 1- Cargar (3, (1, 1)) 2- Cargar (2, (0, 1)) 3- Cargar (1, (0, 0)) 4- Navegar (0) 5- Descargar (1, 1) 6- Navegar (1) 7- Descargar (2, 2) 8- Descargar (3, 2) </div> <div>Tiempo Total: 9.526461839675903 segundos Coste Total: 7087 Longitud del Plan: 8 Nodos Expandidos: 4786</div>	<p>En este ejemplo se observa cómo el carácter de complejidad factorial afecta mucho al rendimiento simplemente eliminando un container de la lista</p>

			h_2	<pre> 1- Cargar (2, (0, 1)) 2- Cargar (1, (0, 0)) 3- Cargar (3, (1, 1)) 4- Navegar (0) 5- Descargar (1, 1) 6- Navegar (1) 7- Descargar (2, 2) 8- Descargar (3, 2) </pre> <pre> Tiempo Total: 0.04700970649719238 segundos Coste Total: 7087 Longitud del Plan: 8 Nodos Expandidos: 346 </pre>	
7-.Mapa completo y complejo	<pre> N N N N X N N X X E E X X X X X </pre>	<pre> 1 S 1 2 S 1 3 R 2 4 R 2 </pre>	h_1	No se generan ficheros	<p>En esta prueba de un mapa avanzado se comprueba la eficiencia de h_2 respecto de h_1, puesto que solo la segunda heurística puede completar el trabajo en un tiempo razonable</p>
			h_2	<pre> 1- Cargar (2, (0, 0)) 2- Cargar (3, (1, 2)) 3- Cargar (1, (1, 1)) 4- Cargar (4, (2, 2)) 5- Navegar (0) 6- Descargar (2, 1) 7- Descargar (1, 1) 8- Navegar (1) 9- Descargar (3, 2) 10- Descargar (4, 2) </pre> <pre> Tiempo Total: 6.652073383331299 segundos Coste Total: 7124 Longitud del Plan: 10 Nodos Expandidos: 4002 </pre>	

Análisis de los casos de prueba

Para rematar esta parte 2 de la práctica, es necesario el analizar el rendimiento de nuestra implementación del algoritmo de búsqueda heurística A*. Como se puede observar en cada uno de los casos de prueba propuestos y resueltos, nuestra implementación es capaz de encontrar sabiamente soluciones al problema propuesto o detectar si ese problema no tiene una solución. Esto, lo hace en un tiempo razonablemente bueno.

Pasando a comparar las implementaciones de las heurísticas propuestas podemos concluir con que:

- La heurística 2, es más informada que la heurística 1 ya que se acerca más al coste más óptimo de la solución.
- La heurística 1 es igual de válida que la heurística 2, puesto que nunca sobrestiman el coste real de la solución.
- La efectividad de la segunda heurística reside en el factor de puertos restantes, donde incentiva que el barco navegue siempre que se cumplan sus precondiciones.
- La complejidad escala de forma factorial puesto que la utilización de más contenedores implica todas las posibilidades de los anteriores multiplicadas por las nuevas posibilidades de este nuevo contenedor
- En un problema real, nuestro grupo elegiría utilizar la heurística 2, ya que explora muchos menos nodos y resuelve el problema en un tiempo considerablemente menor al de la heurística 1.

Para concluir, podemos decir que la implementación del algoritmo de A* hecha por este grupo de prácticas es capaz de resolver cualquier problema con un nivel de complejidad bajo o medio en tiempos muy pequeños. En cuanto a problemas con un nivel alto o muy alto de complejidad se resolverá en tiempos razonables con la heurística 2, al ser más informada. Mientras que, con la heurística 1 el tiempo para resolver este problema será mayor o imposible.

Conclusiones generales de la práctica

Esta práctica cumple el objetivo de poner en valor algunos de los sistemas que permiten modelizar un problema y obtener su solución, así como valorar el procedimiento necesario para su implementación y construcción completa. Es por esto mismo que podemos destacar en esta conclusión cómo funcionan estos sistemas.

Modelar un problema con CSP permite resolver problemas acotados por unas restricciones específicas. En el problema se ha de definir unas variables, su dominio y unas restricciones formales. CSP se apoya para conseguir la solución en la camino y arco consistencia, que restringirán posibles valores a cada una de las variables para cumplir las restricciones propuestas por el problema. Este método es muy potente ya que siempre va a encontrar una solución al problema propuesto. Aunque, tiene una tara, siempre va a revisar todos los posibles valores para todas las variables, lo que lo convierte en un problema exponencialmente complejo.

Por otra parte, A* es buen método para realizar una búsqueda cuando podemos definir una heurística con la información adicional que podemos obtener del problema, sus estados y sus operadores, siendo la mejor opción para este caso. El rendimiento de A* basa su fuerza en la capacidad de definir una heurística lo más informada posible siempre que respete el principio de $h(n) \leq h^*(n)$ puesto que un algoritmo A* con función $h=0$ se comportaría igual que Dijkstra.

En cuanto a nuestro desempeño general en la práctica creemos que hemos realizado un trabajo excepcional en la primera parte cuidando todos los detalles de modelización e implementación sin ningún reproche.

Respecto a la segunda parte, hemos alcanzado el objetivo principal de modelizar e implementar el algoritmo correctamente (además de definir una heurística 2 que creemos que tiene gran potencial), pero nos sentimos inconformes con el rendimiento general del programa. Este ha sido un tema recurrente en su desarrollo y que aunque fue mejorando con el tiempo (implementación de lista simplemente enlazada como lista de abiertos y reducción de la complejidad en los métodos) no es la ideal.

Después de todo esto igualmente creemos que hemos hecho un buen trabajo y esperamos que así sea.