**PRÁCTICA 2**

**MEMORIA**

Heurística y Optimización

Miguel Ávila Martinez 100383360

Nicolás Arnedo Villanueva 100386663



## 

[**Introducción**](#_yhh3ou3ogxf3) **2**

[**Parte 1: CSPStowage**](#_89udxqk7haqo) **2**

[Modelo](#_m4it5iyfwxpp) 2

[Análisis de Resultados](#_554dlwyot2um) 3

[Caso de Prueba 1](#_xvwxx7lwx6xh) 3

[Caso de Prueba 2](#_h65f4wcuamc3) 4

[Caso de Prueba 3](#_1117jyd5xdav) 4

[Caso de Prueba 4](#_c2uw1jah30ml) 5

[Caso de Prueba 5](#_ibzvwk51jbb5) 5

[**Parte 2: ASTARStowage**](#_g93euqxme1ms) **5**

[Modelo](#_r80hfyh9xavg) 5

[Estados](#_81eae58o2dd1) 6

[Operadores](#_icxpkt2kzc3o) 7

[Heurísticas](#_yxxc08n0c9x7) 8

[**Conclusión**](#_cdqi16kuuu) **9**

## 

## **Introducción**

La estiba es la técnica de colocar la carga de un barco para aprovechar al máximo el espacio, mantener la seguridad y hacer eficiente la descarga en los puertos de destino. Para el caso de los barcos de contenedores, parte de la estiba consiste en decidir la posición donde irá colocado cada contenedor.

La bodega de un barco de contenedores se divide normalmente en compartimentos estandarizados llamados bahías de carga, que corresponden a secciones transversales del barco. Cada bahía puede almacenar varias pilas de contenedores de distinta altura. La posición particular dentro de una bahía se denomina celda. Existen celdas especiales que tienen suministro de energía para mantener refrigerados algunos contenedores.

Esta práctica se dividirá en dos partes, por lo que se tienen dos objetivos distintos:

* Crear un modelo por satisfacción de restricciones que permita cargar los contenedores en el barco cumpliendo las restricciones y de modo que no haya que hacer cambios cuando se pare en su configuración cuando se pare en el puerto 1.
* Crear un modelo por búsqueda heurística que calcule el camino de menor coste para cargar todos los contenedores y repartirlos en los puertos 1 y 2

Para la elaboración, primero se plantean los problemas sobre papel y luego se desarrollarán como un programa. Por último destacar que se ha utilizado python en los dos ejercicios.

## **Parte 1: CSPStowage**

### **Modelo**

Este problema de satisfacción de restricciones se define como una terna (X, D, C) donde:

* es el conjunto de variables, que en este caso serán dos, los contenedores normales y los refrigerados.
* representa los dominios de cada variable, que también hay dos, el dominio de donde se pueden colocar los normales y el dominio de donde se pueden colocar los refrigerados.
* es el conjunto de restricciones de nuestro problema, que en nuestro caso tendremos 3 restricciones.

Se tendrá una variable por cada contenedor, por lo que según el caso de prueba que se haga variará el número de variables. El dominio también variará de caso de prueba en caso de prueba según el mapa del barco. Finalmente, las restricciones sí que se mantendrán en todos los casos de prueba.

Las 3 restricciones creadas en nuestro modelo son:

1. No se pueden almacenar dos contenedores en el mismo sitio
2. No se puede colocar un bloque en una celda que no tenga nada debajo, porque entonces estaría flotando
3. Si dos contenedores tienen diferente puerto de destino, se comprueba que el de menor puerto no se encuentra debajo del de mayor puerto

### **Análisis de Resultados**

#### **Caso de Prueba 1**

*mapa1.txt contenedores1.txt*

Este es el escenario de nuestro primer caso de prueba, es uno muy simple para comprobar que nuestro programa funcionaba, al menos en una escala pequeña, correctamente. El resultado que nos devuelve el problema es el siguiente:

Número de soluciones: 2

{1: (1, 1), 3: (0, 1), 4: (0, 0), 2: (1, 0)}

{1: (0, 1), 3: (1, 1), 4: (1, 0), 2: (0, 0)}

Al ser un caso simple, es muy fácil comprobar que el programa está funcionando correctamente. Los contenedores que vayan al mismo puerto se deben de poner en la misma pila, por lo que se pueden colocar los que van al puerto 1 en la pila 0, que sería una solución, o en la pila 1, que sería la otra situación.

Obviamente, los contenedores refrigerados se deben colocar en las celdas con energía, y como se puede ver, el contenedor 1 y 3 están en celdas con profundidad 1 en las dos soluciones, por lo que el programa funciona correctamente.

#### **Caso de Prueba 2**

*mapa2.txt contenedores2.txt*

En este segundo caso de prueba se pone como escenario el ejemplo de la práctica. Ya habiendo comprobado que el programa funciona correctamente con el caso de prueba 1 que es simple, aquí se busca llevarlo a un problema más realístico y ver que el número de soluciones obtenido es elevado, ya que este crece exponencialmente a medida que aumenta el mapa y los contenedores que tenemos.

El número de soluciones obtenido para este problema son 1368. Mostrarlas todas es irrelevante, pero se puede ver que el número está correcto resolviendo el problema como el número de combinaciones posibles para encontrar un número, que sería mayor que 1368, ya que las combinaciones no tendrían en cuenta las restricciones, pero a ojo se vería que está bien.

#### **Caso de Prueba 3**

*mapa3.txt contenedores3.txt*

En este caso de prueba, se explora una situación que no tiene solución, ya que se tienen 5 contenedores refrigerados y en el mapa solo existen 4 celdas con energía. Por lo que el programa no debería devolver ninguna solución.

Número de soluciones: 0

Efectivamente, el archivo mapa3-contenedores3.output.txt nos muestra eso, lo cual nos asegura el funcionamiento correcto del modelo y su implementación.

#### **Caso de Prueba 4**

*mapa4.txt contenedores4.txt*

Este caso de prueba 4 sirve para comprobar que en la pila que se coloque el contenedor refrigerado 1, que solo puede ser la pila 1 o la 2, encima de este solo se coloque bloques que vayan a ir al puerto 1, o no se coloque ningún bloque. El conjunto de todas las soluciones posibles es 120, pero vemos que si cogemos una de las posibles soluciones se cumple que:

{1: (1, 2), 2: (2, 2), 4: (2, 0), 5: (2, 1), 3: (1, 1), 7: (1, 0)}

En esta solución, el bloque refrigerado que va al puerto 1, de ID 1, se ha colocado en la pila 1, y encima de este los dos bloques normales 3 y 7 que también van al puerto 1

#### **Caso de Prueba 5**

*mapa5.txt contenedores5.txt*

Este caso de prueba está creado para comprobar de una manera simple que la restricción de que no se coloquen contenedores en celdas flotantes (es decir sin suelo o sin un contenedor debajo). Lógicamente, si funciona, este problema solo puede tener dos soluciones posibles.

## **Parte 2: ASTARStowage**

### **Modelo**

Para la segunda parte de la práctica se tiene un problema que requiere técnicas de búsqueda heurística para resolver. Se pretende encontrar la solución de coste mínimo de cargar todos los contenedores en un barco y repartirlos entre los puertos 1 y 2. Para encontrar la solución óptima se usará el algoritmo A\* pero primero hay que plantear el modelado que se le va a dar al problema para poder cumplir con todas las restricciones que tiene.

#### ***Estados***

En primer lugar crear los estados del modelo. En todo momento del problema se consideró que era necesario tener en cuenta la siguiente información:

* la posición del barco
* la información de donde están colocado los contenedores en el barco
* los contenedores que están en el puerto inicial
* los contenedores que están en el puerto 1
* los contenedores que están en el puerto 2

Por ello, un estado se planteó formalmente como una terna:

, , , )

En la que:

* B ∈ [*0,1,2*] indica la posición del barco, que puede estar en cualquiera de los tres puertos
* donde x ∈ [*0,1,2*] indican una lista de los IDs de los contenedores que están en dicho puerto “x”
* Es una lista de los contenedores que están en el barco, cada contenedor vendrá representado por una sublista que guarde el id del contenedor, la pila y la profundidad en la que está colocado dentro del barco. Si hay “n” contenedores en el barco, habrá *n* sublistas.

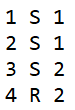
Esta fue la primera propuesta de modelo. Sin embargo, a la hora de realizar la implementación del código, fue necesario la implementación de una variable adicional para poder generar los distintos estados que se crean a raíz de un estado.

Por lo tanto, la nueva terna queda:

, , , )

Donde *DC* es el Dominio de las posiciones en las que se pueden colocar los contenedores dentro del barco. Esta variable contiene las posiciones con dos vectores.

Usaremos un ejemplo con un mapa y una lista de contenedores para plantear el estado inicial, un estado intermedio y un estado final:

*mapa contenedores*

Partiendo de esto, el estado inicial quedaría de la siguiente forma:

, , , )

Un posible estado al parar y descargar en el puerto 1 quedaría:

, , , )

Y el estado final sería:

, , , )

#### ***Operadores***

A continuación, los 3 operadores, que ya vienen en el enunciado de la práctica, pero con sus precondiciones, efectos y coste.

| Operador | Precondición | Efecto | Coste |
| --- | --- | --- | --- |
| Cargar (Px , By , C[id,pila,prof] , CB[], DC[]) | * x=y * C[id] ∈ Px * DC[] ≠ Ø | * remove C[id] de Px * add C[id, pila, prof] a CB[] * remove (pila,prof) de DC[] | 10 + prof |
| Descargar(Px , By , C[id,pila,prof] , CB[], DC[]) | * x=y * C[id] ∈ CB[] | * remove C[id,pila,prof] de CB[] * add C[id] a Px * add (pila,prof) a DC[] | 15 + 2\*prof |
| Navegar(Px, Py, Bz) | * x=z * y=x±1 | * Bz → By | 3500 |

Las precondiciones del operador cargar son:

* El barco está en puerto
* El contenedor está en el puerto
* El dominio de posibles posiciones disponibles en el barco no está vacío. Si estuviera vacío significa que el barco está lleno y no se pueden cargar más contenedores.

Los efectos de este son:

* quitar el ID del contenedor de la lista de contenedores del puerto
* añadir el ID y la pila y profundidad del sitio en el que se carga a la lista de contenedores en el barco
* eliminar la dupla (pila, profundidad) de la lista de dominio de posiciones del contenedor

Las precondiciones del operador descargar son:

* El barco está en puerto
* El contenedor está en el barco

Los efectos de este son:

* quitar el ID,profundidad y pila de la lista de contenedores en el barco
* añadir el ID a la lista de contenedores del puerto
* añadir la dupla (pila, profundidad) a la lista de dominio de posiciones del contenedor

Las precondiciones de la operación navegar son:

* El barco está en puerto de partida
* El puerto de destino es mayor o inferior en 1 que el de origen. Esto es así ya que nuestros puertos son el 0,1 y 2. Entonces, desde el puerto 0 solo puede ir al 1, desde el 1 puede ir al 0 y 2, y desde el 2 solo puede ir al 1.

#### ***Heurísticas***

Para generar una heurística, obtenemos un problema relajado a partir del original, simplemente eliminando alguna precondición de los operadores.

En primer lugar, se puede eliminar la precondición x=y, lo cual significa que no hay que comprobar si el barco está en el puerto, lo que significa que está en todos los puertos. Esto produce que el tercer operador no sea necesario, ya que sería como si el barco estuviera en medio de los tres puertos, y solo se necesitaría la grúa para cargar y descargar. Por tanto, la heurística resultante de este problema relajado es:

Como no hay que navegar, el coste del problema será el número de veces que se ejecuta la función cargar y descargar, que es el número de contenedores que hay en el puerto inicial, ya que todo contenedor cargado deberá ser descargado.

En segundo lugar, se puede eliminar la precondición y=x±1, lo cual significa que el barco puede navegar desde el puerto 0 hasta el puerto 2 directamente sin tener que parar en el puerto 1. Esta heurística será un relajación si y sólo si la lista de contenedores solo tiene contenedores con puerto de destino el 2. Esta heurística sería igual que la anterior, pero solo sumandole el coste de navegar:

## Conclusión

Lamentablemente, no se ha podido llegar a completar al 100% la práctica. La primera parte se pudo realizar, aunque nos costó bastante, pero se sacó adelante y el programa que hemos creado cumple con todas las restricciones del problema y funciona correctamente.

Sin embargo, para la segunda parte no se pudo realizar lo mismo. Esto se debe principalmente a una mala gestión del tiempo y planificación, ya que consideramos que con un par de días más se habrían podido resolver los problemas que nos estaban surgiendo en la implementación del A\*.

Creemos que la modelización de la parte 2 y las heurísticas que hemos planteado son correctas y habrían funcionado, pero los problemas que nos han impedido generar la implementación son los siguientes:

* No se ha podido generar los distintos estados que aparecen a raíz de cargar un bloque. Para generar estos se implementó el dominio de las posiciones disponibles en el barco, que en teoría funciona, pero en la implementación no se ha sabido como programarlo.
* No se ha conseguido expandir los nodos
* Las restricciones de colocación creadas en la primera parte no hemos sido capaces de remodelarlas para que se aplicaran con el operador de carga
* No se ha podido comprobar si los nodos estaban siendo expandidos correctamente

Como conclusión, nos llevamos un mal sabor de boca por no haber podido resolver este problema, pero se ha intentado en la medida de lo que se ha podido. Ha sido una práctica que nos parecía interesante el tópico, y estaba bien explicada, se entendía claramente lo que cada enunciado pedía y lo que había que hacer.