MEMORIA

Satisfacción de Restricciones y Búsqueda Heurística

Heurística y Optimización

2021/2022

Grupo 81-11

Alumnos: Carlos Rodríguez Calzada: 100428991

Pablo Santa Cruz Sánchez: 100428988

ÍNDICE

DESCRIPCIÓN DE LOS MODELOS

INTRODUCCIÓN

En este documento se va a estudiar la resolución de dos problemas especificados en el enunciado de la práctica. Se analizarán tanto los **modelos** implementados como los **resultados** obtenidos. Por último añadiremos unas breves **conclusiones** sobre los problemas que hemos tenido durante la realización de la práctica.

PARTE 1: VALIDACIÓN CON PYTHON CONSTRAINT

Para el modelado **CSP**, es preciso determinar qué constituyen las variables, cuáles son sus dominios, y qué restricciones son necesarias para cumplimentar los requisitos exigidos en el enunciado de la práctica:

Las **variables** están compuestas por los **container**, de tal forma que su **subíndice** clarifica el **ID** del container:

$$X = \{x_i : i = \text{id container}\}$$

Por otro lado, tenemos los **dominios** de las variables, que serán **diferentes** según el **tipo** de container (**estándar** o **refrigerado**).
Así, los contenedores **estándar** podrán tomar cualquier posición posible en la matriz (columna, fila) a excepción de las casillas reservadas al suelo o las columnas donde haya "X" flotantes. Su dominio se representa por **D** .

Sin embargo, los contenedores **refrigerados** verán reducido su dominio ($\mathbf{D_r}$) a las casillas donde se permita colocar este tipo de container, las celdas con energía, que será un subconjunto del dominio de los contenedores estándar:

$$D_r \subseteq D$$

Por último, tenemos las **restricciones**, mediante las cuales los dominios de las variables se verán reducidos para conseguir soluciones:

R1 - Que los valores obtenidos sean todos diferentes entre sí: Es una restricción obvia, puesto que no podemos tener dos contenedores ocupando el mismo espacio.

R2 - Que no haya contenedores flotantes: Esta restricción no permite la existencia de contenedores en posiciones de la matriz (columna, fila) que no estén soportados por el suelo o por un contenedor en una altura inmediatamente inferior (columna, fila+1).

R3 - Que no haya que reordenar los contenedores: Es decir, que es obligatorio que los contenedores que vayan al puerto i estén apilados sobre otros que vayan al mismo puerto o sobre contenedores que vayan al puerto k (donde: k>i). De esta manera, los contenedores podrán descargarse fácilmente sin tener que recolocar nada.

PARTE 2: PLANIFICACIÓN CON BÚSQUEDA HEURÍSTICA

En el modelo de búsqueda heurística, es necesario determinar varios elementos:

Espacio de estados: Que definirá la situación del problema en cada momento. Para representar de manera correcta el estado, tendremos que almacenar diversos datos sobre el mismo. En nuestro caso, un estado se representa mediante una **tupla** de dos elementos: **(P, posicion_contenedores)**.

- **P** representa el **puerto** en el que se encuentra el **barco**.
- **posicion_contenedores** es un diccionario, que almacenará **información** de cada uno de los **contenedores** del problema (no solo de los que están en el barco). Esta información estará compuesta por la situación actual (la posición en el barco si está en él o el puerto en caso contrario), el tipo de contenedor y el puerto destino.

Cabe destacar que, siguiendo este método de representación, el **estado inicial** será (0, {contenedores}). Por otro lado, el **estado meta** será (X, {contenedores}) donde X será el último puerto en el que descargará y donde, para todos los contenedores, su posición será la de un puerto, que tendrá que coincidir con el puerto al que va destinado.

A continuación, debemos concretar los operadores necesarios para representar las acciones posibles en el problema. La ejecución de estos operadores se verá condicionado por el cumplimiento de una serie de precondiciones:

OP1 - Cargar(): El barco cargará un contenedor del puerto en una posición posible del barco. Para que esta acción se ejecute, el puerto en el que se encuentra el barco no debe estar vacío.

OP2 - Descargar(): El barco descargará un contenedor de su cubierta. Este operador se ejecutará si en el barco hay algún contenedor.

OP3 - Navegar(): Con la ejecución de este operador, el barco navegará del puerto que se encuentra al puerto siguiente. Es imprescindible que el barco no pueda navegar si le faltan contenedores por descargar en ese puerto o queden en el puerto contenedores que deba llevar al siguiente.

DEFINICIÓN DE HEURÍSTICAS

Al tener que definir dos heurísticas hemos visto oportuno **discriminarlas** según la consideración de **altura** de los contenedores. Una heurística tiene en cuenta los costes añadidos por apilar en columnas (h2) y en otro simplemente se toman los valores mínimos (h1).

H1: Esta heurística **relaja** los costes de altura al **apilar** y **desapilar** un container, de tal forma que serán constantes independientemente de la altura del contenedor. Esta heurística

funcionará de forma **eficiente** para un **número** de contenedores relativamente **reducido**, en tanto la altura de las pilas en el barco no aumente demasiado (ahí se perdería información). Sin embargo, sí se tienen en cuenta tanto los costes de navegación, como los mínimos de carga y descarga (suponemos que siempre se apila en el suelo). Por esto, el valor alcanzado por esta heurística nunca superará al de la heurística perfecta. Por esto, podemos afirmar que esta heurística es **admisible**. Así:

$$\mathbf{h1(n)} = (\sum (C_i + D_i)) + N$$

Donde **Ci** son los costes de **carga** de cada contenedor del barco, **Di** los de **descarga** (ambos cálculos teniendo en cuenta coste mínimo) y **N** el coste de **navegación** hasta el último puerto en el que haya que descargar.

H2: Esta heurística tiene en cuenta costes de **altura**, pero sólo para un operador. H2 **obvia** los costes de **carga** de un contenedor, y se centra en descargar y navegar. Hemos preferido obviar los de carga puesto que suponen una **penalización menor** que los de descarga en el cálculo de costes. Esta heurística podría no ser viable en otros contextos, pero por la naturaleza de este problema (el barco no puede navegar hacia atrás) esto no supone un problema. La heurística devolverá valores **poco realistas** mientras está cargando **inicialmente** todos los contenedores desde la bahía inicial, sin embargo, cuando los haya cargado todos, el valor calculado por la heurística será muy **cercano** al de la heurística **perfecta salvo** por el coste de carga en las **reordenaciones** (puesto que lo único que le queda es descargar y navegar).
En cualquier caso nunca superará los costes de la heurística perfecta por lo que también es una heurística **admisible**. De esta manera:

$$\mathbf{h2(n)} = (\sum D_i) + N$$

Donde **Di** son los los costes de **descarga** de cada contenedor del barco (teniendo en cuenta la ponderación por **altura**) y **N** el coste de **navegación** hasta el último puerto en el que haya que descargar.

DETALLES DE IMPLEMENTACIÓN

Para la implementación de esta parte, debemos precisar algunos detalles sobre la implementación.
En cuanto a la **cola de prioridad**, hemos implementado una **lista doblemente enlazada**. En vez de ordenar la lista cada vez que se inserta un elemento, hemos preferido que la **inserción** se haga **ordenadamente** (los nodos cuya función de evaluación devuelve un valor menor, siempre se colocarán al inicio de la lista). Cuando vayamos a **expandir** un nodo, siempre expandiremos el **head** de la lista.
Por otro lado, también vemos necesario explicar el funcionamiento de la estructura de datos que utilizamos para almacenar el estado. La **complejidad** para acceder a un elemento del estado (un contenedor) es **O(1)**.

ANÁLISIS DE RESULTADOS

PARTE 1: VALIDACIÓN CON PYTHON CONSTRAINT

Una vez implementado el modelo con las restricciones, hemos diseñado una serie de configuraciones para analizar cómo se comporta el algoritmo en **diferentes escenarios**. Para un mejor análisis, todos los mapas, a excepción del último, se basan en el **mapa** del enunciado con **ligeros cambios** en cada uno de ellos.

Configuración 1 (mapa1 contenedores1):

Este es el caso más **simple**, tenemos 5 contenedores refrigerados. Sin embargo, en el mapa solo hay 4 celdas con energía. Es **imposible** obtener una **solución**.

Configuración 2 (mapa2 contenedores2):

En este caso seguimos con el mapa original. El problema es que todos los contenedores **refrigerados** pertenecen al **puerto 1** y las celdas de **energía** se encuentran en la parte **inferior** de la bodega. En este mapa, existen tantos contenedores como celdas con energía. De esta manera, siempre que haya contenedores del puerto 2, será **imposible** encontrar **solución sin reordenamiento**. En el fichero de salida expresamos que no se puede dar una solución sin reordenar y mostramos las soluciones posibles con el reordenamiento.

Configuración 3 (mapa3 contenedores3):

Esta configuración es muy **similar** a la **configuración 2**. La única diferencia se encuentra en el **intercambio** de los **tipos** entre un contenedor del puerto 1 y otro del puerto 2. Ahora, mientras no haya un número grande de contenedores del puerto 2, todos estos pueden apilarse encima del contenedor del puerto 2 que se encuentra en la celda de energía. El contenedor normal del puerto 1 puede colocarse donde sea en la bodega. Esta configuración sí **nos devuelve soluciones**, un total de 216.

Configuración 4 (mapa4 contenedores4):

En esta configuración simplemente queríamos mostrar una **decisión de diseño** que hemos tomado a la hora de implementar el modelo. Hemos asumido que si existe una **posición inválida** en la bodega, “X”, que **no** pertenece al **suelo** de esta, **invalida** todas las posiciones que se encuentran por **encima** de esa posición. Porque si se colocara un contenedor encima de ella, estaría flotando.
Para demostrar esto, contamos con 4 contenedores refrigerados y 4 celdas de energía. Sin embargo, una de estas celdas de energía se encuentra encima de una “X” que no pertenece al suelo. Por ello, la celda de energía se **invalida** y no quedan suficientes para asignar todos los contenedores refrigerados. **No hay solución**.

Configuración 5 (mapa5 contenedores5):

Para este caso, hemos forzado que las **celdas de energía** estén, de algún modo, en la **parte superior** de la bodega. Seguimos contando con 4 contenedores refrigerados y 4 celdas de energía. Entonces, al **no** existir **suficientes** contenedores que al apilarse formen un **suelo** para estos, **no existe solución**, porque estarían flotando.

Configuración 6 (mapa6 contenedores6):

Para este último caso, hemos **simplificado** el problema drásticamente. Tenemos un mapa **2x3** con sólo **3 contenedores**. La idea es poder observar fácilmente las soluciones que se obtienen, siendo solamente 2 soluciones. Con ellas se puede comprobar de una forma muy sencilla que el **sistema funciona**.

PARTE 2: PLANIFICACIÓN CON BÚSQUEDA HEURÍSTICA

Configuración 1 (mapa1 contenedores1):

La configuración está pensada para que **no** tenga **solución**. Haciendo que el algoritmo no sea capaz de encontrarla (al detectar una **lista abierta vacía**) y una vez termine de ejecutarse, devuelva que no tiene solución.

Configuración 2 (mapa2 contenedores2):

En este caso, forzamos a que haya **reorganizaciones** sí o sí. A diferencia de la parte 1, que se buscaba no tenerlas, aquí podemos ver en el output como se realizan, encontrando la solución.

Configuración 3 (mapa3 contenedores3):

Esta es una configuración **normal** y corriente. Puede haber reorganizaciones, pero el algoritmo elige el camino que las evita a la hora de cargar los contenedores.

Configuración 4 (mapa4 contenedores4):

Este es el mismo caso que la configuración 4 de la parte 1, tenemos la "**X**" **flotando**, que invalida la cuarta celda de energía e **impide** que el algoritmo encuentre **solución** puesto que no la tiene.

Configuración 5 (mapa5 contenedores5):

La idea de esta configuración es la misma que la configuración 5 de la parte 1. Volvemos a **forzar** que las celdas de **energía** están **arriba** del todo y no hay suficientes contenedores que hacen de suelo. Vuelve a no dar solución.

Configuración 6 (mapa6 contenedores6):

Esta configuración sigue siendo un caso más, **simple**, para observar el funcionamiento del algoritmo. Devuelve **solución**.

ANÁLISIS COMPARATIVO DE HEURÍSTICAS

Como hemos mencionado anteriormente, nuestras **heurísticas** se diferencian entre sí por **varios factores**, pero el principal es que uno se tiene en cuenta la **altura** de los contenedores y en otro no.
La heurística **h1 no** tiene en **cuenta** costes de **altura**, siempre son **mínimos** para cualquiera de sus operadores (carga y descarga). Esto no supone un gran problema con un **número reducido** de contenedores, puesto que la **altura** de las pilas **no** será tan **significativa** como para suponer una gran pérdida de información del estado.

Sin embargo, en la heurística **h2 sí** se tiene en cuenta este factor, el **comportamiento** de esta heurística es ligeramente **diferente**, puesto que al ignorar los costes de carga, el valor de la **heurística crecerá** conforme se vayan cargando contenedores. Cuando ya están cargados otorgará una **información** bastante **precisa**, puesto que al tener en cuenta costes de **navegación** (como h1) y de **altura**, otorgará valores realistas e incluso alcanzará el valor de la **heurística perfecta si** en la configuración del problema **no existen reordenaciones**.
Esta **segunda heurística** funciona **mejor** que h1 cuando hay **muchos contenedores**, pero puesto que la capacidad computacional de nuestro programa no nos permite apilar **cantidades** suficientemente **grandes**, esta característica **no** se hará realmente **visible** en la **práctica**.
Cabe destacar también que h1 no presenta uno de los “puntos débiles” de h2, en h1 no se pierde información debido a reordenamientos.

CONCLUSIONES

Esta práctica nos ha servido para afianzar nuestros conocimientos sobre resolución de problemas con diversos métodos. En la **primera sección**, la mayor parte de nuestro tiempo la pasamos pensando cómo **modelar** el problema. Sin embargo, también nos costó **entender** al principio el funcionamiento de **python constraint**, puesto que no sabíamos cómo la API **operaba** las **restricciones**.

Por otro lado, en la segunda sección la mayor parte de nuestro **tiempo** la empleamos en la **implementación** del algoritmo **A***. Al realizar la implementación vimos necesario **realizar** alguna pequeña **modificación** en nuestro **modelo** para que funcionara correctamente.
En esta sección, sufrimos más problemas en la implementación que conceptuales, debido al uso de clases y la compartición de información.