Práctica 2: *Satisfacción de Restricciones y Búsqueda Heurística*

# Heurísticas y Optimización.

Grado de Ingeniería en Informática. Curso 2021-2022

Planning and Learning Group
Departamento de Informática

## 1. Objetivo

El objetivo de esta práctica es que el alumno aprenda a modelar y resolver problemas tanto de satisfacción de restricciones como de búsqueda heurística.

## 2. Enunciado del problema

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, como se muestra en la Figura 1. 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.

Para el desarrollo de esta práctica simplificaremos el problema real y asumiremos que estamos preparando la estiba de barcos que tienen una única bahía de carga y que en cada celda cabe un sólo contenedor. Igualmente solo consideraremos dos tipos de contenedores:

- Estándar: Contenedor normal que puede ir en cualquier celda siempre que esté en la base de la pila o que la celda inferior esté ocupada.
- Refrigerado: Contenedor que guarda carga perecedera y que además de las restricciones de un contenedor estándar, debe colocarse en las celdas especiales con suministro de energía.

Al llegar al puerto de partida, cada contenedor tiene asociado un identificador, un puerto de destino y un indicador de si es refrigerado o no.

### 2.1. Parte 1: Validación con Python constraint

Para esta parte asumiremos que existe un puerto de partida, y sólo dos puertos de destino. Los barcos irán primero al puerto 1 y luego al puerto 2. Nos interesa saber si dada una lista de contenedores en el puerto de origen y un mapa de la bahía de carga de un barco:

1. Es posible colocar todos los contenedores en la bodega cumpliendo las restricciones de almacenaje.
2. Es posible colocar todos los contenedores cumpliendo las restricciones anteriores y con la posibilidad de que no sea necesario recolocar ningún contenedor en el puerto 1.

Para esta parte se pide:


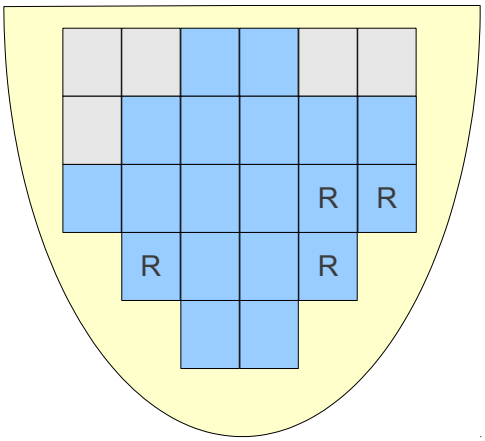


Figura 1: Imagen de un buque de contenedores en un puerto (Fuente: http://www.worldshipping.org) y del esquema de una sección transversal de la bodega

1. Modelar estas preguntas como un problema de satisfacción de restricciones CSP de modo que la respuestas afirmativas permitan obtener una configuración viable de la bodega.
2. Utilizando la librería *python-constraint*, desarrollar un programa que codifique dicho modelado de modo que el problema pueda representarse y resolverse con dicho programa.

El programa desarrollado recibirá como entrada dos ficheros. Uno con el mapa de la bahía de carga y el segundo con la lista de los contenedores a cargar en el puerto de partida. Los alumnos deben generar sus propios casos de prueba, con el fin de analizar el comportamiento del programa, siguiendo el siguiente ejemplo.

| | | |
|---|---|---|
| N N N N<br>N N N N<br>E N N E<br>X E E X<br>X X X X | 1 S 1<br>2 S 1<br>3 S 1<br>4 R 2<br>5 R 2<br>... | N: Celda normal<br>E: Celda con energía<br>X: Posición no disponible<br>S: Contenedor estándar<br>R: Contenedor refrigerado |

El programa deberá ejecutarse desde una consola o terminal con el siguiente comando:

```
python CSPStowage.py <path> <mapa> <contenedores>
```

Donde `path` define la ruta donde se encuentran los ficheros, y `mapa` y `contenedores` son los nombres de los ficheros de entrada correspondientes.

El programa debe generar un fichero de salida en el mismo directorio donde se encuentran los ficheros de entrada. El nombre del fichero de salida será de la forma `<mapa>-<contenedores>.output`. Por ejemplo, si los ficheros utilizados en la llamada son `mapa1` y `contenedores1`, el nombre del fichero de salida será `mapa1-contenedores1.output`.

El fichero de salida tendrá en la primera línea el número de soluciones encontradas, y a continuación todas las soluciones al problema, una por línea. En cada solución, la celda que corresponde a cada contenedor se expresará con el siguiente formato: `id-contenedor: (pila, profundidad)`. Un posible ejemplo del contenido del fichero de salida sería el siguiente:

```
Número de soluciones: <n>
{3: (3, 2), 1: (3, 1), 4: (2, 3), 0: (3, 0), 2: (2, 2)}
{3: (3, 2), 1: (3, 1), 4: (2, 3), 0: (3, 0), 2: (1, 3)}
...
```

Donde, la primera solución corresponde con la siguiente colocación de los contenedores:

```
N N N 0
N N N 1
E N 2 3
X E 4 X
X X X X
```

Es decir, la numeración de las celdas empieza en la esquina superior izquierda con valor (0,0), incrementando su valor de izquierda a derecha (pilas) y de arriba a abajo (profundidad).

Las llamadas al programa para ejecutar los casos de prueba que diseñe el alumno se deben incluir en un script llamado `CSP-calls.sh`. Un posible ejemplo del contenido de este script es el siguiente:

```
python CSPStowage.py ./CSP-tests mapa1 contenedores1
python CSPStowage.py ./CSP-tests mapa2 contenedores2
...
```

## 2.2. Parte 2: Planificación con Búsqueda Heurística

Para esta parte se plantea construir la planificación completa de carga y descarga de los contenedores. Para construir dicha planificación se utilizarán algoritmos de búsqueda heurística que construyan una secuencia de acciones que permita llevar todos los contenedores a su puerto de destino. Igual que en la parte anterior, se asume que todos los contenedores están en el puerto de inicio, y ahora éstos se pueden *cargar* en la bahía de carga en cualquier orden, siempre que se respeten las restricciones de almacenaje. Los contenedores se podrán *descargar* en su puerto de destino o inclusive en cualquier otro puerto, si fuera necesario para algún tipo de recolocación (sin limitación en la altura y cantidad de pilas en el puerto). Además, hay que considerar los viajes del barco, que puede *navegar* del puerto inicial al puerto 1, y del puerto 1 al puerto 2. Se quiere optimizar el coste total. Los movimientos de un contenedor tienen un coste fijo y uno variable que depende del desplazamiento. Este desplazamiento lo representaremos por el parámetro $\Delta$, que representa el número de celdas en altura que tiene que recorrer un contenedor. El coste individual de las acciones se calcula de la siguiente forma:

| Acción | Coste |
|---|---|
| cargar | $10 + \Delta$ |
| descargar | $15 + 2\Delta$ |
| navegar | 3500 |

En esta parte se pide:

1. Modelar el problema de planificación de la carga y descarga de contenedores como un problema de búsqueda.
2. Implementar el algoritmo A$^*$ (en el lenguaje de programación que se considere) que permite resolver el problema.
3. Diseñar e implementar funciones heurísticas (al menos 2), que estimen el coste restante, de modo que sirvan de guía para los algoritmos de búsqueda heurística.
4. Proponer casos de prueba y resolverlos con la implementación desarrollada. Estos casos se deben generar razonablemente dependiendo de la eficiencia alcanzada en la implementación.
5. Realizar un análisis comparativo del rendimiento del algoritmo con las heurísticas implementadas.

La implementación deberá recibir los mismos parámetros de entrada que en la parte 1 y adicionalmente el nombre de la heurística. Los casos de prueba pueden ser diferentes a los de la parte 1. Se deberá ejecutar desde una consola o terminal con el siguiente comando:

```
./ASTARStowage.sh <path> <mapa> <contenedores> <nombre-heurística>
```

Donde, `ASTARStowage.sh` es el script que permite invocar al programa desarrollado.

Los posibles valores del parámetro que representa el nombre de la heurística deberán describirse en la memoria, así como en la ayuda de la implementación desarrollada.

Al igual que en la primera parte, las llamadas al programa para ejecutar los casos de prueba que diseñe el alumno se deben incluir en un script adicional llamado `ASTAR-calls.sh`, cuyo contenido será de la forma:

```
./ASTARStowage.sh ./ASTAR-tests mapa1 contenedores1 heurística1
...
```

El programa debe generar dos ficheros de salida. Ambos se deben generar en el mismo directorio donde se encuentran los ficheros de entrada:

- Fichero con la solución del problema. Debe contener la secuencia de acciones que conforman la solución al problema. En cada línea se mostrará la acción correspondiente precedida de un número que indicará el orden de la acción en la secuencia:

  ```
  1.  <acción1>(<parametro1>, <parametro2>, ...)
  2.  <acción2>(<parametro1>, <parametro2>, ...)
  ```

  El nombre del fichero será de la forma `<mapa>-<contenedores>-<heuristica>.output`. Por ejemplo para la llamada anterior el fichero con la solución se denominará: `mapa1-contenedores1-heuristica1.output` y se encontrará en el directorio `./ASTAR-tests`.

- Fichero de estadísticas. Este fichero debe contener información relativa al proceso de búsqueda, como el tiempo total, coste total, longitud de la solución y los nodos expandidos. Por ejemplo,

  ```
  Tiempo total: 145
  Coste total: 54
  Longitud del plan: 27
  Nodos expandidos: 132
  ```

  En este caso el fichero tendrá extensión `.stat`: `<mapa>-<contenedores>-<heuristica>.stat`.

## 3. Directrices para la Memoria

**La memoria debe entregarse en formato pdf y tener un máximo de 15 hojas en total**, incluyendo la portada, el índice y la contraportada. Al menos, ha de contener:

1. Breve introducción explicando los contenidos del documento.
2. Descripción de los modelos, argumentando las decisiones tomadas.
3. Análisis de los resultados.
4. Conclusiones acerca de la práctica.

| **Importante:** Las memorias en un formato diferente a pdf serán penalizadas con 1 punto. |
|---|

La memoria **no debe incluir código fuente** en ningún caso.

# 4. Evaluación

La evaluación de la práctica se realizará sobre 10 puntos. Para que la práctica sea evaluada deberá realizarse al menos la primera parte de la práctica:

1. Parte 1 (4 puntos)
    - Modelización del problema (1 punto).
    - Implementación del modelo (2 puntos).
    - Resolución y análisis de los casos de prueba (1 punto).
2. Parte 2 (6 puntos)
    - Modelización del problema (1 puntos).
    - Implementación del algoritmo (3 puntos).
    - Resolución de casos de prueba y análisis comparativo de las heurísticas implementadas (2 puntos).

En la evaluación de la modelización del problema, un modelo correcto supondrá la mitad de los puntos. Para obtenerse el resto de puntos, la modelización del problema deberá:

- Ser formalizada correctamente en la memoria.
- Ser, preferiblemente, sencilla y concisa.
- Estar bien explicada (ha de quedar clara cuál es la utilidad de cada variable/restricción).
- Justificarse en la memoria todas las decisiones de diseño tomadas.

En la evaluación de la implementación del modelo, un modelo correcto supondrá la mitad de los puntos. Para obtenerse el resto de puntos, la implementación del problema deberá:

- Corresponder íntegramente con el modelo propuesto en la memoria.
- Entregar código fuente correctamente organizado y comentado. Los nombres deben ser descriptivos. Deberán añadirse comentarios en los casos en que sea necesario para comprenderlo.
- Contener casos de prueba que muestren diversidad para la validación de la implementación.

# 5. Entrega

Se tiene de plazo para entregar la práctica hasta el 19 de diciembre a las 23:55.
Sólo un miembro de cada pareja de estudiantes debe subir:

- Un único fichero `.zip` a la sección de prácticas de Aula Global denominado *Entrega Práctica 2*.
- El fichero debe nombrarse `p2-NIA1-NIA2.zip`, donde `NIA1` y `NIA2` son los últimos 6 dígitos del NIA (rellenando con 0s por la izquierda si fuera preciso) de cada miembro de la pareja. Ejemplo: `p2-054000-671342.zip`.
- La memoria en formato pdf debe entregarse a través del enlace Turnitin denominado *Entrega Memoria Práctica 2*. La memoria debe entregarse en formato pdf y debe llamarse `NIA1-NIA2.pdf` — después de sustituir convenientemente el NIA de cada estudiante. Ejemplo: `054000-671342.pdf`

La descompresión del fichero entregado en primer lugar debe generar un directorio llamado `p2-NIA1-NIA2`, donde `NIA1` y `NIA2` son los últimos 6 dígitos del NIA (rellenando con 0s por la izquierda si fuera preciso) de cada miembro de la pareja. Este directorio debe contener: primero, la misma memoria en formato pdf que ha sido entregada a través de Turnitin, y debe llamarse `NIA1-NIA2.pdf` — después de sustituir convenientemente el NIA de cada estudiante; segundo, un fichero llamado *autores.txt* que identifique a cada autor de la practica en cada línea con el formato: NIA Apellidos, Nombre. Por ejemplo:

```
054000 Von Neumann, John
671342 Turing, Alan
```

La descompresión de este fichero debe producir al menos dos directorios llamados exactamente "`parte-1`" y "`parte-2`", que contengan los archivos necesarios para ejecutar correctamente cada una de las partes.

**Importante:** no seguir las normas de entrega puede suponer una pérdida de hasta 1 punto en la calificación final de la práctica.

Se muestra a continuación una distribución posible de los ficheros que resultan de la descompresión: