Práctica 2: *Satisfacción de Restricciones y Búsqueda Heurística*

Heurística y Optimización.

Grado de Ingeniería en Informática. Curso 2024-2025

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

Tras los buenos resultados obtenidos con la venta de billetes y la asignación de pistas de aterrizaje, la compañía aérea quiere dar solución a varios problemas mediante técnicas de satisfacción de restricciones y búsqueda heurística.

2.1. Mantenimiento de flota de aviones

El problema del mantenimiento de una flota de aviones consiste en decidir dónde y cuándo se realizan tareas de mantenimiento, respetando ciertas restricciones: la capacidad de los talleres mecánicos, el orden temporal entre ciertas tareas de mantenimiento (por ejemplo, inflar las ruedas y verificar la presión de inflado debe hacerse después de cambiar una rueda defectuosa), etc. La compañía aérea ha conseguido simplificar el problema, clasificando las tareas de mantenimiento necesarias en dos tipos (tipo 1 estándar y tipo 2 especialista), haciendo el mantenimiento de toda la flota el mismo día (dividido en n franjas horarias) y en el mismo lugar, evitando tener que lidiar con la programación de los vuelos, ya que su flota de aviones no tiene vuelos programados ese día.

Hay dos tipos de aviones en la flota: aviones estándar (*STD*) y aviones jumbo (*JMB*). La compañía aérea alquila hangares que cuentan con dos tipos de talleres mecánicos y parkings:

- Taller estándar (*STD*): realiza tareas de mantenimiento de tipo 1.
- Taller especialista (*SPC*): realiza tareas de mantenimiento de tipo 1 y 2.
- Parking (*PRK*): sólo se usan para almacenar aviones que no necesitan tareas de mantenimiento.

Una posible disposición espacial de los talleres a los que tiene acceso la compañía aérea puede observarse en la Tabla 1. Los talleres estándar se muestran en color verde y los talleres especialistas se muestran en color naranja, mientras que los parkings se muestran en blanco. Para referirse a cada taller, se interpreta la tabla como una matriz, siendo entonces el parking de la esquina superior izquierda la posición $(0, 0)$, y el único taller especialista de la última fila se encuentra en la posición $(4, 3)$.

2.2. Parte 1: Validación con Python constraint

Para realizar el mantenimiento de los distintos aviones, se deben tener en cuenta las siguientes restricciones:

PRK	STD	PRK	SPC	PRK
STD	STD	STD	STD	PRK
STD	SPC	STD	SPC	PRK
SPC	PRK	PRK	STD	PRK
PRK	STD	STD	SPC	PRK

Tabla 1: Conjunto de talleres mecánicos.

1. Todo avión en la flota debe tener asignado un único taller o un parking en cada franja horaria del problema.

2. Cada taller puede atender hasta 2 aviones en una franja horaria, pero en ningún caso podrá haber más de un avión JUMBO en la misma franja horaria en el mismo taller.

3. Un taller especialista puede realizar tareas de mantenimiento estándar, pero en ningún caso un taller estándar podrá realizar tareas de mantenimiento especialista. Es decir, si un avión tiene programada una tarea de mantenimiento especialista y otra estándar, deberá tener asignado al menos un taller especialista en alguna de las franjas horarias del día.

4. Se podrá especificar un orden para las tareas de mantenimiento que necesita un avión. En particular, el único orden que se puede exigir es que todas las tareas de tipo 2 se realicen antes que todas las tareas de tipo 1. Los mecánicos de cualquier taller que tenga asignado un avión con alguna tarea de mantenimiento pendiente trabajarán en esa tarea de inmediato, por lo que si se deben realizar m tareas de mantenimiento de tipo 2, el avión deberá pasar por m talleres especialistas en m franjas horarias antes de poder ser asignado a un taller estándar.

5. Si un taller o parking tiene asignado algún avión en una franja horaria, al menos uno de los talleres o parkings adyacentes vertical y horizontalmente deberá estar vacío para garantizar la maniobrabilidad de los aviones. Por ejemplo, si un avión está en el parking $(0, 0)$ en la franja horaria i, al menos uno de los talleres estándar de las posiciones $(0, 1)$ o $(1, 0)$ no podrá tener asignado ningún avión en la franja i.

6. En ningún caso dos aviones JUMBO podrán tener asignados talleres adyacentes en la misma franja horaria para garantizar la maniobrabilidad de los aviones.

Para esta parte se pide:

1. Modelar este problema como un problema de satisfacción de restricciones (CSP) de modo que se puedan obtener asignaciones válidas de aviones a talleres y parkings en las n franjas horarias, y que se planifiquen todas las tareas de mantenimiento.

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 un fichero, con toda la información necesaria para resolver el problema, sobre los talleres y aviones. Tendréis que generar vuestros propios casos de prueba, con el fin de analizar el comportamiento del programa, siguiendo el siguiente ejemplo:

```
Franjas: 4
5x5
STD:(0,1) (1,0) (1,1) (1,2) (1,3) (2,0) (2,2) (3,3) (4,1) (4,2)
SPC:(0,3) (2,1) (2,3) (3,0) (3,3)
PRK:(0,0) (0,2) (0,4) (1,4) (2,4) (3,1) (3,2) (3,4) (4,0) (4,4)
1-JMB-T-2-2
2-STD-F-1-3
3-STD-F-3-0
4-JMB-T-1-1
5-STD-F-2-2
...
```

Estructura del fichero:

```
1ª línea: franjas horarias del problema
2ª línea: tamaño de la matriz de talleres
3ª línea: posiciones de los talleres estándar
4ª línea: posiciones de los talleres especialistas
5ª línea: posiciones de los parkings
6ª y posteriores: código de identificación de los
   aviones [ID-TIPO-RESTR-T1-T2]
      ID: nº secuencial
      TIPO: STD/JMB (Estándar/Jumbo)
      RESTR: T/F (tareas tipo 2 antes que tipo 1)
      T1: número (número de tareas de tipo 1)
      T2: número (número de tareas de tipo 2)
```

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

`python CSPMaintenance.py <path maintenance>`

Donde `<path maintenance>` define la ruta donde se encuentra el fichero de entrada correspondiente a los datos del problema, y `CSPMaintenance.py` debe ser el nombre de tu script.

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 `<maintenance>.csv`. Por ejemplo, si el fichero utilizado en la llamada es `maintenance01`, el nombre del fichero de salida será `maintenance01.csv`.

El fichero de salida tendrá en la primera línea el número de soluciones encontradas y, a continuación, al menos una solución del problema (aunque es deseable que muestren algunas más de forma aleatoria). Un posible ejemplo del contenido del fichero de salida sería el siguiente:

```
N. Sol: j
Solución 1:
   1-JMB-T-2-2: SPC(0,3), SPC(0,3), STD(0,1) STD(0,1)
   2-STD-F-1-3: SPC(4,3), SPC(4,3), SPC(4,3) SPC(4,3)
   3-STD-F-3-0: STD(1,0), PRK(0,0), STD(1,0) STD(1,0)
   4-JMB-T-1-1: SPC(2,3), PRK(0,0), STD(1,0) PRK(0,0)
   5-STD-F-2-2: SPC(2,3), SPC(2,3), SPC(2,3) SPC(2,3)
Solución 2:
   1-JMB-T-2-2: SPC(0,3), SPC(0,3), STD(0,1) STD(0,1)
   2-STD-F-1-3: SPC(4,3), SPC(4,3), SPC(4,3) SPC(4,3)
   3-STD-F-3-0: STD(1,0), PRK(0,0), STD(1,0) STD(1,0)
   4-JMB-T-1-1: SPC(2,3), SPC(2,3), STD(1,0) PRK(0,0)
   5-STD-F-2-2: SPC(2,3), SPC(2,3), SPC(2,3) SPC(2,3)
Solución 3:
   ...
```

Donde:

1. La primera línea indica el número de soluciones encontradas.
2. Las siguientes líneas representan a qué taller o parking está asignado cada avión en cada franja horaria

   a) `ID-AVION` seguido de tantos pares como franjas horarias haya en el problema.

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

```
python CSPMaintenance.py ./CSP-tests/maintenance01
python CSPMaintenance.py ./CSP-tests/maintenance02
...
```

2.3. Planificación de rodaje de aviones

Se entiende por rodaje al movimiento de un avión en el suelo. En un momento dado, múltiples aviones pueden necesitar llegar desde una posición inicial hasta la pista que tienen asignada para el despegue. La compañía aérea quiere un programa que, dado un mapa de un aeropuerto, un conjunto de aviones, una posición inicial y una pista de despegue asignada a cada avión, encuentre un camino que especifique el rodaje que cada avión debe realizar para llegar a su pista asignada. Es fundamental que el programa garantice que no se produzca ninguna colisión en el proceso de rodaje de todos los aviones.

2.4. Parte 2: Planificación con búsqueda heurística

La estructura de las pistas del aeropuerto se representa con un mapa como el de la Figura 1. Los aviones pueden realizar movimientos a las pistas (celdas) adyacentes vertical u horizontalmente, o pueden esperar en la celda en la que se encuentran. Rodar de una pista a otra adyacente toma a todos los aviones exactamente una unidad de tiempo. Dos aviones nunca pueden estar en la misma pista al mismo tiempo, para evitar colisiones. Además, por seguridad, dos aviones no pueden usar la misma carretera para moverse de una pista a otra al mismo tiempo. Es decir, si hay dos aviones a_1 y a_2 en celdas adyacentes c_1 y c_2 respectivamente en un instante de tiempo, no podrá estar el avión a_1 en la celda c_2 y el avión a_2 en la celda c_1 en el instante de tiempo siguiente.

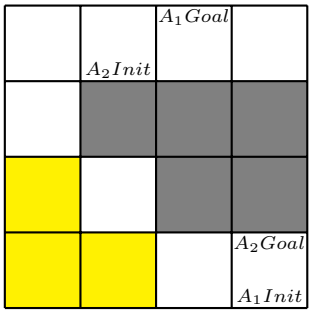


Figura 1: Mapa esquemático con la ubicación de dos aviones y la estructura del aeropuerto. En las celdas amarillas los aviones **no** podrán detenerse. Por las celdas de color gris los aviones **no** pueden circular.

En dicho mapa podemos observar la siguiente información:

- Celda marcada con A_iInit representa la posición inicial del avión i.
- Celda marcada con A_iGoal representa la pista de despegue a la que debe llegar el avión i.
- Celdas de color gris son celdas por las que los aviones no pueden transitar.

- Celdas de color amarillo son celdas en las que los aviones deben transitar sin parar.

Una solución al problema del rodaje de aviones es, para cada avión, la secuencia de movimientos (incluyendo las acciones de esperar) que debe realizar para llegar desde su celda inicial a su celda asignada sin chocar con ningún otro avión. Por ejemplo, una solución para el problema del mapa de la Figura 1 sería:

	1	2	3	4	5	6	7	8	9
A_1	$\leftarrow$	$\leftarrow$	$\uparrow$	w	$\leftarrow$	$\uparrow$	$\uparrow$	$\rightarrow$	$\rightarrow$
A_2	$\leftarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	w	w

Tabla 2: Posible solución al problema de rodaje de la Figura 1. En cada instante de tiempo hasta que ambos aviones hayan alcanzado su pista designada, cada avión deberá realizar un movimiento o esperar explícitamente (representado con w).

La espera de a_1 en el instante de tiempo 4 no era necesaria para evitar una colisión, pero es una solución válida (aunque no necesariamente óptima, dependiendo de la métrica que se defina).

En esta parte se pide:

1. Modelar el problema de rodaje de aviones como un problema de búsqueda con el objetivo de minimizar el *makespan* (tiempo total necesario para que todos los aviones llegue a su posición de destino).
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. Todas las heurísticas deberán ser admisibles, y se debe justificar por qué lo son.
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 como parámetros de entrada:

1. La dirección del fichero *mapa.csv* con toda la información sobre la ubicación de los aviones, pistas asignadas, etc. Ese fichero, *mapa.csv*, tendrá el siguiente formato (que se corresponde con el mostrado en la Figura 1):

```
2
(3,3) (0,2)
(0,1) (3,3)
B;B;B;B
B;G;G;G
A;B;G;G
A;A;B;B
```

   La primera línea tiene un único número n indicando el número de aviones en el problema. Las siguientes n líneas contienen dos pares de números, representando la posición inicial y la pista asignada a cada avión. El resto de las líneas se corresponden con un fila de la matriz que representa al mapa. Una *B* es una casilla blanca, una *G* una gris y una *A* una amarilla. Nótese que el mapa no tiene que ser necesariamente cuadrado.

2. Un parámetro (*num-h*) para determinar la heurística que debe utilizar la implementación (***1*** para la primera heurística implementada y ***2*** para la segunda).

Los casos de prueba se tendrán que poder ejecutar desde una consola o terminal con el siguiente comando:

```
python ASTARRodaje.py <path mapa.csv> <num-h>
```

Donde, `ASTARRodaje.py` es el script que permite invocar al programa desarrollado,
`path mapa.csv` define el mapa con todos los elementos importantes a tener en cuenta y `num-h` es el anteriormente comentado.

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` (que debe tener permisos de ejecución), cuyo contenido será de la forma:

```
python ASTARRodaje.py ./ASTAR-tests/mapa.csv 1
python ASTARRodaje.py ./ASTAR-tests/mapa.csv 2
python ASTARRodaje.py ./ASTAR-tests/mapa02.csv 1
python ASTARRodaje.py ./ASTAR-tests/mapa02.csv 2
...
```

Donde `./ASTAR-tests` es la dirección de los ficheros de los casos de prueba. De esta manera se podrán ejecutar varios casos de prueba a la vez. 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 una línea por cada avión del problema, y en esta línea se mostrarán todos los movimientos y posiciones del avión. Así, `(x,y)` $\rightarrow$ `(x,y+1)` representa un movimiento a la derecha. Por ejemplo, la solución de la Tabla 2 en este formato sería:

  `(3,3)` $\leftarrow$ `(3,2)` $\leftarrow$ `(3,1)` $\uparrow$ `(2,1)` w `(2,1)` $\leftarrow$ `(2,0)` $\uparrow$ `(1,0)` $\uparrow$ `(0,0)` $\rightarrow$ `(0,1)` $\rightarrow$ `(0,2)`
  `(0,1)` $\leftarrow$ `(0,0)` $\downarrow$ `(1,0)` $\downarrow$ `(2,0)` $\downarrow$ `(3,0)` $\rightarrow$ `(3,1)` $\rightarrow$ `(3,2)` $\rightarrow$ `(3,3)` w `(3,3)` w `(3,3)`

  El nombre del fichero será de la forma `<mapa>-<num-h>.output` donde `<num-h>` es el número de la heurística utilizada en la ejecución. Por ejemplo para la primera línea de la llamada anterior el fichero con la solución se denominará `mapa-1.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 de ejecución en segundos, el makespan, el valor heurístico del estado inicial y los nodos expandidos. Por ejemplo,

  ```
  Tiempo total: 145s
  Makespan: 38
  h inicial: 15
  Nodos expandidos: 900000
  ```

  En este caso el fichero tendrá extensión `.stat`: `<mapa>-<num-h>.stat`.

3. Desarrollo de la práctica

La práctica se realizará en grupos de dos estudiantes.

4. 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.

5. 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.

6. Entrega

Se tiene de plazo para entregar la práctica hasta el 20 de diciembre de 2024 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`

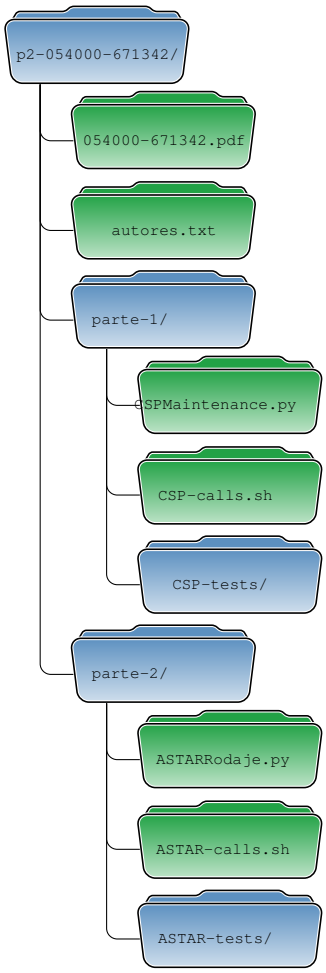
La descompresión del fichero entregado en primer lugar debe generar un directorio llamado `p2-NIA1-NIA2`, donde `NIA1` y `NIA2` son los últimos 6 dígitos del NIA (rellenando con 0s por la izquierda si fuera preciso) de cada miembro de la pareja. Este directorio debe contener: primero, la misma memoria 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:


p2-054000-671342/
054000-671342.pdf
autores.txt
parte-1/
CSPMaintenance.py
CSP-calls.sh
CSP-tests/
parte-2/
ASTARRodaje.py
ASTAR-calls.sh
ASTAR-tests/