Práctica 2

Satisfacción de Restricciones y Búsqueda Heurística

Miguel Jimeno Casas - 100495932 - 100495932@alumnos.uc3m.es

Ismael Plaza Martín – 100499716 - 100499716@alumnos.uc3m.es

Grupo 82 - Ingeniería Informática

Introducción

Durante esta práctica se va a llevar a cabo la resolución de dos problemas, cada uno resuelto con un enfoque diferente.

En la primera parte del problema, nos pide que modelemos mediante **CSP (Problema de Satisfacción de Restricciones)** el mantenimiento de una flota de aviones. Asignando los aviones de la flota tanto a parkings como a talleres diferentes. Todo esto teniendo en cuenta diferentes restricciones que se basan en la información de cada avión que se quiere alojar en cada taller o parking y en el conjunto de todos los aviones.
Este apartado pone énfasis en el análisis y la gestión de restricciones para obtener configuraciones óptimas en escenarios definidos.

En la segunda parte, se aborda un problema de planificación mediante **búsqueda heurística**. Se trata de un problema típico de planificación de trayectorias y prevención de conflictos en un entorno dinámico. En concreto, para seguir el contexto de la práctica, se basa en aviones. Cada avión debe seguir un camino desde su posición inicial hasta su pista de despegue, respetando restricciones temporales y espaciales. Buscando reducir al máximo el tiempo total necesario para que todos los aviones lleguen a su destino (si es posible).
Recurriendo para resolver el problema al algoritmo de A* con heurísticas admisibles. Al usar dos heurísticas diferentes, se comparará el rendimiento de las mismas.

Descripción del Modelo CSP

Como todo modelo de satisfacción de restricciones, debemos de definirlo como una tupla **(X, D, C)** donde, **X**=Conjunto de variables, **D**=Dominio de las variables y **C**=Conjunto de restricciones que debe de seguir el problema.

Definición de la variable $\left(X_{i,t}\right)$

La variable X_{it} indica la asignación de un avión *i,* en una franja horaria *t*, donde:

- i ∈ A, el conjunto de aviones de la flota.
- t ∈ T, el conjunto de franjas horarias disponibles.

Conjuntos Relacionados:

1. **Conjunto de Aviones**:
   A={a1,a2,a3,…,an} .Cada avión a_i tiene asociada la siguiente información:

   - **ID**: Identificador único para cada avión.

- **TIPO**: Tipo de avión (STD: estándar o JMB: jumbo).
- **T**: Indicador booleano (T/F) que determina si las tareas de tipo 2 deben realizarse antes que las de tipo 1.
- **T1**: Número de tareas tipo 1 asignadas al avión.
- **T2**: Número de tareas tipo 2 asignadas al avión.}

2. **Conjunto de Franjas Horarias**:
   T={1,2,3,…,m}, donde *m* representa el número total de franjas horarias disponibles.

Definición del dominio (Posiciones)

El dominio inicial de todas las variables abordará el conjunto entero de todas las posiciones de la matriz de entrada (mapa de talleres y pakings). Cada posición se encontrará dentro de un subconjunto que se especificará en los datos. Tendremos tres diferentes subconjuntos:

- **SPC** (Taller Especialista)
- **STD** (Taler Estándar)
- **PRK** (Parking)

Un ejemplo para que sea más visual podría ser:

Posiciones = {(0,0); (0,1); (0,2); (1,0); (1,1); (1,2)}

Datos = {STD:{(0, 0); (0, 1)}, SPC:{(1, 0); (1, 2)}, PRK:{(0, 2); (1, 1)}}

Restricciones (C)

Para modelar el problema se han delimitado varias restricciones, unas pedidas expresamente por el enunciado, y otras que hemos definido nosotros, ya que no se dejaban claras algunas pautas del comportamiento de los aviones en ciertas situaciones.
Por lo que C será el conjunto de las siguientes restricciones:

- **Restricción de asignación única (R1)**

  Cada avión debe estar asignado a una posición (taller o parking) en cada franja horaria. Esto garantiza que no queden aviones sin asignar.

  $\exists X_{i,t} \forall i \in A, \forall t \in T$

- **Restricción de capacidad (R2)**

En cada franja horaria, ninguna posición puede alojar más de 2 aviones, y si uno de ellos es un avión JMB, el otro debe ser un avión STD.

$$\forall t \in T, \forall (x,y) \in Posiciones, \left(\sum_{i \in A} \delta(x,y,i,t) < 2 \right)$$

$$\left(\sum_{i \in A} \delta(x,y,i,t) = 2 \right) \rightarrow (JMB \in i \wedge STD \in i)$$

- **Restricción de tareas (R3)**

  El taller SPC puede realizar tanto tareas de tipo 1 como tipo 2. Sin embargo el taller STD solo puede realizar tareas de tipo 1, no tareas de tipo 2.

  $$\forall t \in T, \forall (x,y) \in Posiciones, \left(\delta(x,y,i,t) = 1 \wedge \left(T_1 \in i \right) \right) \rightarrow (Posición(x,y) = SPC \vee STD)$$

  $$\forall t \in T, \forall (x,y) \in Posiciones, \left(\delta(x,y,i,t) = 1 \wedge T_2 \in i \right) \rightarrow (Posición(x,y) != STD)$$

- **Restricciones de adyacencia (R4)**

  Si una posición (x,y)(x, y)(x,y) está ocupada en una franja horaria, al menos una de sus posiciones adyacentes debe estar vacía. A parte, si los aviones son JMB, no podrán estar en posiciones adyacentes.

  $$\forall t \in T, \forall (x,y) \in Posiciones, \left(\sum_{(x',y') \in (adyacentes(x,y))} \delta(x',y',i,t) \leq 1 \right)$$

- **Restricción prioridad de resolución de tareas (R5)**

  Si un avión tiene dentro de un información REST = T en vez de una F, hay que asegurarse de que las tareas de tipo 2 se realicen antes que las de tipo 1.

  $$\forall i \in A, si \left(T_1 \left(X_{i,t_1} \right) \wedge T_2 \left(X_{i,t_2} \right) \wedge REST_i = T \right) \rightarrow t_1 > t_2$$

- **Restricción de completar tareas (R6)**

  Se deben de resolver todas las tareas antes de que se acaben la franjas horarias disponibles.

$$\forall i \in A, \sum_{t \in T} (asignacion(x, y, i, t) = 1 \wedge (Posicion(x, y) != PRK)) \geq T_1(i) + T_2(i)$$

Por lo que **C = {R1, R2, R3, R4, R5, R6}** terminando de definir así el conjunto de restricciones del problema de CSP.

Decisión de diseño:

En este apartado, se va a declarar las decisiones de diseño a la horra de gestionar el código, ya que hay partes que no quedan suficientemente claras con el enunciado proporcionado.

Al no especificar la capacidad de los parkings, a estos les hemos tratado como a los talleres, por lo que se podrán almacenar como máximo a dos aviones, uno JMB y un STD o dos STD

En la parte de la práctica donde se mencionan *"por lo que si se deben realizar m tareas de mantenimiento de tipo 2, el avion deberá pasar por m talleres especialistas en m franjas horarias antes de poder ser asignado a un taller estandar. "*
Al no quedarnos del todo claro este apartado, decidimos tratarlo como que no tiene porque cambiar de taller. Podrá hacer los m arreglos en el mismo taller si se da la posibilidad.

Aunque dentro de los datos del avión de la variable en cuestión RESTR = "T", no tendrá porque realizarse este tipo de tareas nada más empezar. Se podrá ir al parking antes de empezar a resolver tareas, pero eso sí; una vez empieze a resolver tareas de tipo 2, no podrá parar hasta que las resuelva todas.

Análisis de los resultados (CSP)

Lo primero es definir el formato del resultado:

En primer lugar, aparece el número total de soluciones posibles que hay para resolver ese problema.
A continuación se muestra el número de soluciones, como máximo 30, para no saturar el fichero de salida. Cada solución se compone de una especie de matriz de asignación, que refleja la distribución de los aviones a lo largo de las franjas horarias, cumpliendo todas las restricciones del problema (en el caso de que se genere solución). La matriz tiene como

número de filas el número de aviones presentes en los datos, y como número de columnas, el número de franjas horarias disponibles. Cada celda de la matriz se corresponde con una asignación a una posición de taller o parking para cada avión en cada franja horaria.

Una vez definido el formato de salida, tomaremos dos casos diferentes, para ver cuales son los resultados en función de los datos introducidos, para hacer un análisis que cubra diferentes casos de uso.

Caso 1: El problema es satisfacible y se resuelve correctamente siendo delimitado por las restricciones.

Input:

```
Franjas: 3
3x3
STD:(0,1) (1,0) (1,1) (1,2)
SPC:(2,1) (2,0)
PRK:(0,0) (0,2) (2,2)
1-JMB-T-2-1
2-STD-F-1-1
3-STD-F-3-0
```

Solución posible:

```
Solucion 1:
1-JMB-T-2-1:  SPC(2, 0) SPC(2, 0) SPC(2, 0)
2-STD-F-1-1:  PRK(2, 2) SPC(2, 0) SPC(2, 0)
3-STD-F-3-0:  STD(1, 0) SPC(2, 1) SPC(2, 1)
```

Al ser un escenario de una matriz (3x3), el número de soluciones posibles que cumplen las restricciones delimitadas anteriormente, son `4112316`

En concreto, hemos elegido la solución un por escoger una y analizar.

Podemos ver que se cumplen todas las restricciones, por ejemplo, en la misma franja horaria, no se almacenan más de dos aviones (y no se almacenan dos si son JMB), se realizan todas las tareas antes de finalizar las franjas horarias y para ser más concretos, las de tipo 2 asegura que se resuelven en talleres SPC. Podemos ver la diferencia entre le avión 1 y el avión 3, el primer avión tiene como RESTR = T por lo que no puede ser que en la primera franja esté en un STD, mientras que el tercero al tener RESTR = F sí que puede ir en la primera franja horaria a un STD; de hecho, lo hace. Así podríamos ver tiene en cuenta todas las restricciones.

Caso 2: El problema no es satisfacible, por lo que el número de soluciones es 0.

Input:

```
Franjas: 2
2x3
STD: (0,2) (1,0)
SPC: (0,0) (1,2)
PRK: (0,1) (1,1)
1-JMB-T-0-2
2-STD-F-2-1
3-JMB-F-0-2
```

Output:

```
N. Sol: 0
```

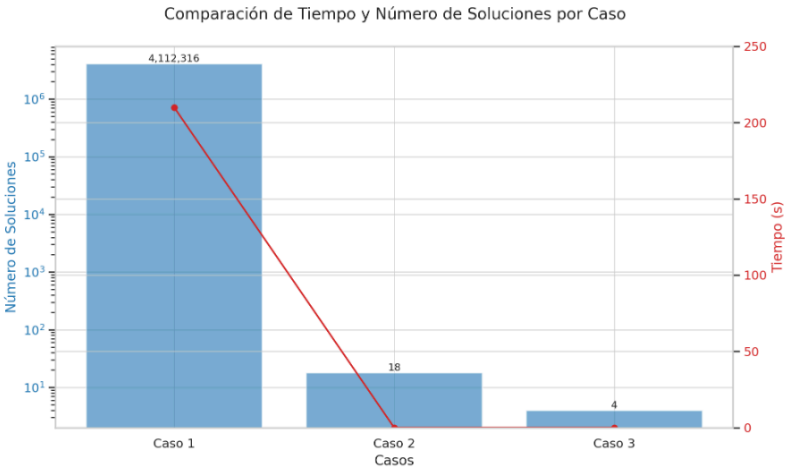
Podemos ver, que como buscábamos, no hay solución debido a que el avión con ID=2, tiene 3 tareas que realizar y solo hay 2 franjas. Por lo que es imposible que se

terminen las tareas antes que las franjas.

Finalizamos así el pequeño análisis de los resultados generados por el código. En la entrega de código, se han abordado todos los casos de prueba para que se vea claramente, que cumple todas las restricciones que hemos mencionado y requiere el problema.

Para finalizar el análisis de los resultados de la parte del problema resuleto con CSP, analizarremos una gráfica, que incluye métricas para problemas de dificultad incremental. Para mostrar los resultados de forma compacta.

La gráfica se va a realizar en función de los tres primeros casos de prueba propuesto en el código. Estos casos son los únicos que proporcionan un número de soluciones mayores que 0, ya que hay algunas soluciones que pasan el filtro de todas las restricciones anteriormente descritas.



Como podemos observar hay una clara diferencia entre tanto el número de soluciones y el tiempo de ejecución del caso 1, con el del 2 y 3. Esto se debe, prácticamente en su plenitud, al tamaño de la matriz donde se situan los talleres y los parkings. El primer caso se trata de una matriz 3x3 mientras que las otras 2 son 2x2, lo que reduce mucho las cantidad de soluciones posibles. Como es lógico, al tener muchas más posibilidades de soluciones el primer caso, el tiempo de ejecución aumenta exponencialmente.

Descripción del Modelo de Búsqueda Heurística

Este modelo se centra en la busqueda heurística abordando un problema de rodaje de aviones utilizando el algoritmo A* junto con técnicas heurísticas avanzadas. El objetivo principal es optimizar el makespan, es decir, el tiempo total necesario para que todos los aviones completen sus trayectorias, asegurando una resolución efectiva del problema.

Entorno

- **Aviones:**
  Cada avión tiene una posición inicial(x_i, y_i) y otra posición final(x_f, y_f) a alcanzar, definidas por coordenadas sobre la matriz del mapa.

- **Mapa:**
  El mapa se modelada a partir de una matriz formada por tres tipos diferentes de celdas:
  'B' representa las celdas transitables sin ninguna restricción añadida
  'A' representa las celdas transitables, con la restricción de que no está permitido detenerse.
  'G' representa las celdas que no son transitables

Elementos del modelo

Para poder formular el problema con claridad, se van a definir:

- **Espacio de estados:**
  Cada estado representa la configuración de las posiciones de cada avión en un espacio de tiempo, el tiempo que ha transcurrido (movimientos realizados) desde la salida del avión y el historial de movimientos.

- **Acciones:**
  Corresponde a las posibles posiciones a las que el avión se puede mover, en base a las restricciones propuestas por el enunciado.

- **Restricciones:**
  Restricciones del mapa:
  El avión no podrá moverse a las celdas definidas con una 'G' y no podrá detenerse en las celdas definidas con una 'A'.
  Capacidad de las celdas:
  No podrá haber más de un avión por celda en el mismo espacio de tiempo.
  Cruces entre aviones:
  No se permiten intercambios de posiciones entre dos aviones en un mismo paso

- **Función objetivo:**
  Cada avión debe alcanzar su posición final (x_f, y_f) respetando todas las restricciones descritas.

Algoritmo A*

Función de evaluación

El algoritmo utiliza una función de evaluación **f(n)** para decidir el orden de exploración de los nodos. Esta función combina dos componentes:

Coste acumulado **g(n)**: Representa el coste desde el estado inicial hasta el estado actual.
Estimación heurística **h(n)**: Una estimación del coste restante para alcanzar el objetivo desde el estado actual.

$$f(n) = g(n) + h(n)$$

La implementación diseñada, admite dos tipos diferentes de heurísticas:

1. **Heurística Manhattan:** Calcula la distancia Manhattan entre las posiciones actuales y los objetivos de cada avión, sumando las distancias individuales.

$$h_{Manhattan} = \sum_{i=1}^{n} \left|\left(x_i - x_i^f\right)\right| + \left|\left(y_i - y_i^f\right)\right|$$

2. **Heurística de congestión:** A la heurística de Manhattan, añade penalizaciones basadas en interferencias entre aviones y posibles bloqueos en el mapa.

$$h_{Congestion} = \sum_{i=1}^{n} d_i + 5 \cdot interference_i + 20 \cdot blockage_i + 15 \cdot occupation_i$$

Donde:

$$d_i = \left|\left(x_i - x_i^f\right)\right| + \left|\left(y_i - y_i^f\right)\right|$$

Teniendo en cuenta que (x, y) son las coordenadas cartesianas que representan la posición

actual de un avión en el mapa, y $x^f e y^f$ son las coordenadas del objetivo final al que dicho avión debe llegar.

Ambas heurísticas son admisibles y consistentes, lo que garantiza que no sobreestiman el coste real restante y permiten a A* encontrar soluciones más óptimas para el problema.

Expansión de nodos

En cada iteración, el algoritmo selecciona el nodo con el menor valor de f(n) de la lista abierta (cola de prioridad). El nodo se expande generando todas las combinaciones posibles de movimientos válidos para los aviones. Las combinaciones que resultan en cruces o colisiones entre aviones se descartan.

Criterios de parada

El algoritmo solo termina si llega a uno de los dos siguientes casos, o todas las posiciones actuales de los aviones coinciden con sus posiciones objetivos. O la lista abierta queda vacía, lo que indica que no hay soluciones posibles para ese problema.

Análisis de Resultados Búsqueda Heurística

Como en el problema de CSP, lo primero es definir el formato de los resultados generados. Con cada entrada, se pueden llegar a generar hasta 4 archivos distintos 2 outputs, que será donde se verá reflejado el movimiento de los aviones (en caso de llegar a una respuesta). Y otros dos que son las estadisticas, donde se mostrará tanto tiempo total, makespan,h inicial y nodos expandidos que nos servirá para futuros análisis.

Una vez definido el formato del resultado, se pasa a la observación de ejemplo en concreto con el que se pueda obtener un resultado.

Podemos ver el resultado del caso 2, el cual se trata de un mapa, que solo tiene un camino posible para ir al punto objetivo buscado por los dos aviones, por lo que se deberían cruzar, pero como hay un saliente en el mapa para hacer que los aviones no se crucen, da una solución al problema.

Input:

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

Soluciónes posibles:

Solución encontrada con la heurística de Manhattan:

Avión 1: (0, 3) ← (0, 2) ↓ (1, 2) w (1, 2) w (1, 2) w (1, 2) w (1, 2) w (1, 2) ↑ (0, 2) ← (0, 1) ← (0, 0) ↓ (1, 0) ↓ (2, 0) ↓ (3, 0) → (3, 1) → (3, 2) → (3, 3)

Avión 2: (3, 3) ← (3, 2) ← (3, 1) ← (3, 0) ↑ (2, 0) ↑ (1, 0) ↑ (0, 0) → (0, 1) → (0, 2) → (0, 3) w (0, 3) w (0, 3) w (0, 3) w (0, 3) w (0, 3) w (0, 3) w (0, 3) w (0, 3)

Mientras que la solución que proporciona para la heurísitca de conflicto es diferente:

Avión 1: (0, 3) ← (0, 2) ↓ (1, 2) w (1, 2) w (1, 2) ↑ (0, 2) ↓ (1, 2) ↑ (0, 2) → (0, 3) ← (0, 2) ← (0, 1) ← (0, 0) ↓ (1, 0) ↓ (2, 0) ↓ (3, 0) → (3, 1) → (3, 2) → (3, 3)

Avión 2: (3, 3) ← (3, 2) ← (3, 1) ← (3, 0) ↑ (2, 0) ↑ (1, 0) ↑ (0, 0) → (0, 1) → (0, 2) ↓ (1, 2) ↑ (0, 2) ↓ (1, 2) ↑ (0, 2) w (0, 2) w (0, 2) w (0, 2) w (0, 2) → (0, 3)

Análisis de las Restricciones

Dividiremos las restricciones en secciones para que esta parte quede más clara. Las divisiones serán:

Restricciones del Mapa

En los casos propuestos, los aviones permanecen constantemente en las celdas accesibles (B, A) y no se detienen al estar en una celda 'A' comprobado en el caso 4.

Caso 4: Para este caso, como no puede permanecer en la celda esperando, no hay ninguna solución posible.

Input:

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

Solución posible:

```
1   No se encontró solución.
2
```

Por otro lado, en el caso 9 y en el 3, se observa claramente que los aviones no acceden a las celdas marcadas por ‘G’.

Caso 3: Todas las celdas son tipo 'G'.

Input:

```
2
(0,0) (0,2)
(1,1) (0,0)
G;G;G
G;G;G
```

Solución posible:

```
1   No se encontró solución.
2
```

Caso 9: La matriz esta dividida por una franja completa de celdas tipo 'G', por lo que el avión dos no puede llegar a su destino.

Input:

```
2
(0,0) (0,2)
(1,1) (0,0)
G;G;G
G;G;G
```

Solución posible:

```
1    No se encontró solución.
```

Restricción de Capacidad

En ninguno de los casos propuestos ha habido mas de un avión por celda en un mismo instante de tiempo, respetando así la restricción.
Esto se puede confirmar en los casos 6 y 7.

Caso 6: Los aviones inician desde la misma posición, por lo tanto, el problema esta mal planteado.

Input:

```
2
(0,0) (0,2)
(0,0) (1,1)
B;B;B
B;B;B
```

Solución posible:

```
Error: Los aviones empiezan en la misma posición. No es posible resolver el problema.
```

Caso 7: Los aviones tienen el objetivo en la misma celda,.

Input:

```
2
(0,0) (0,2)
(1,0) (0,2)
B;B;B
B;B;B
```

Solución posible:

```
1    No se encontró solución.
```

Restricción de Cruce

La restricción se respeta en todos los casos de prueba y funciona correctamente. Un ejemplo de esto es el caso 5.

Caso 5: El mapa en este caso es un 1x3 con dos aviones, lo que hace que sea imposible que cualquiera de los dos aviones llegue a la posición objetivo sin cambiar posiciones entre ellos.

Input:

```
2
(0,0) (0,2)
(0,2) (0,0)
B;B;B
```

Solución posible:

```
No se encontró solución.
```

Comparación de las Heurísticas

Para realizar esta comparación, nos vamos a ayudar de una tabla, la cual define el problema que se está tratando, y las diferencias entre el uso de las diferentes heurísticas.

Problema	Heurística Manhattan	Heurística Congestión
Caso 1: 2 (0,0) (0,2) (0,2) (0,0) B;B;B G;A;G	Tiempo total: 0.00s Makespan: 4 h inicial: 4 Nodos expandidos: 6	Tiempo total: 0.00s Makespan: 4 h inicial: 44 Nodos expandidos: 10
Caso 2: 2 (0,3) (3,3) (3,3) (0,3) B;B;B;B B;G;B;G A;G;G;G A;A;B;B	Tiempo total: 0.01s Makespan: 17 h inicial: 6 Nodos expandidos: 68	Tiempo total: 0.02s Makespan: 17 h inicial: 36 Nodos expandidos: 106
Caso 8: 3 (0,0) (1,2) (3,0) (0,3) (3,2) (2,1) B;A;G;B G;A;B;B A;A;A;A B;B;B;B	Tiempo total: 0.01s Makespan: 6 h inicial: 11 Nodos expandidos: 17	Tiempo total: 48.28s Makespan: 6 h inicial: 21 Nodos expandidos: 2137

Conclusión general:
Tras analizar los tres problemas, se concluye lo siguiente:

1. **Eficiencia**: La heurística Manhattan es consistentemente más eficiente que la heurística de congetión, expandiendo menos nodos en todos los casos.
2. **Calidad de las soluciones**: Ambas heurísticas generan soluciones con el mismo makespan, por lo tanto, estan igualadas en este apartado
3. **Consumo de tiempo**: La heurística Manhattan resuelve los problemas significativamente más rápido que la heurística de congestión en los problemas más complejos, como el caso 8, donde la diferencia es abismal.
4. **Estimación inicial (h)**: La heurística de congestión tiende a sobreestimar el costo inicial, lo que puede explicar su mayor expansión de nodos y el tiempo que tarda en realizar los problemas.

La heurística de Manhattan demuestra ser significativamente más eficiente que la de congestión. Esto se refleja en un menor tiempo de resolución y en la expansión de menos nodos, especialmente en escenarios más complejos donde la diferencia de rendimiento es notable.