Universidad Carlos III

Heurística y Optimización 2024-25

Práctica 2

Satisfacción de Restricciones y Búsqueda Heurística

Fecha: **20/12/24**

GRUPO: **85**

Alumnos: **MANUEL ANDRES TRUJILLO / 100423448**

VICENTE ANTONIO BARBATO / 100438114

Tabla de contenido

Introducción

En el presente trabajo abordaremos la resolución de dos problemas principales utilizando técnicas de Satisfacción de Restricciones (CSP) y búsqueda heurística, aplicadas al contexto de mantenimiento de aviones y planificación de rodaje en un aeropuerto. Para el primer problema, implementaremos un modelo formalizado utilizando la librería python-constraint, que nos permitirá validar asignaciones óptimas y respetar las restricciones impuestas. El segundo problema será modelado como una búsqueda en grafos, utilizando el algoritmo A* junto con funciones heurísticas diseñadas específicamente.

A lo largo de esta documentación, se desarrollará una descripción detallada de los modelos empleados, exponiendo con claridad la lógica detrás de cada restricción y decisión de diseño. Asimismo, se analizará en profundidad la implementación de los algoritmos, justificando las estrategias elegidas y los pasos necesarios para su desarrollo.

El estudio incluye también un análisis de resultados obtenido a partir de diversos casos de prueba, los cuales se han diseñado para verificar el correcto funcionamiento de los algoritmos y evaluar su rendimiento. Se presentarán las soluciones encontradas, estadísticas relevantes y comparativas entre las heurísticas propuestas.

Finalmente, concluimos con un apartado de reflexiones y conclusiones, donde se destacarán los aprendizajes obtenidos, los retos enfrentados y las observaciones generales derivadas de la práctica. Esta estructura tiene como objetivo proporcionar una visión global y clara del trabajo realizado, facilitando la comprensión de los problemas abordados y sus respectivas soluciones.

Descripción Formal de los Modelos

Parte 1: Validación con Python - constraint

Para llevar a cabo la implementación de la primera parte de la práctica, iniciamos desarrollando una función encargada de la apertura y lectura del archivo de entrada adaptado al formato solicitado, comprobando el correcto paso de parámetros para la ejecución del código desarrollado, además procedemos al almacenamiento de la información de entrada proporcionada en distintas variables, así poder llevar a cabo la clasificación del fichero de entrada de forma explícita y poder realizar el desarrollo del problema de satisfacción de restricciones.

En el caso de las variables, hemos definido los aviones en cada franja horaria, y como dominio hemos establecido las posiciones disponibles para ellos, clasificándose como talleres estándar (STD), talleres especializados (SPC) y zonas de aparcamiento (PRK). La asignación la hemos llevado a cabo

recorriendo la estructura de datos que contiene los aviones y sus distintas características. Por otro lado, se lleva a cabo una distinción entre los aviones que tienen tareas asignadas del tipo 2 (T2) de aquellos que no tienen. De forma que en caso de que el avión tenga tareas T2, su dominio se restringe a las posiciones correspondientes a talleres especializados durante las franjas horarias asignadas a estas tareas, sin embargo, aviones que no tienen taras T2 asignadas o ya han sido realizadas, su dominio se extiende a talleres estándar o a las zonas de aparcamiento, según corresponda.

Habiendo definido en lo que a variables y dominios respecta, procedemos con la explicación de las restricciones necesarias para poder hallar el conjunto de soluciones que se adapten a nuestro problema CSP. Iniciamos describiendo la implementación de las restricciones en lo que respecta a la capacidad de talleres y parkings, y la forma de asignación. En cuanto a la primera restricción, hace referencia a que cada avión debe estar asignado a una única posición (taller o parking) en cada franja horaria. Esta condición la cubrimos al definir variables con un dominio único por franja horaria. Dado que cada variable $X_{i,t}$ representa la posición de un avión i en la franja t.

Por otro lado, nos encontramos con la segunda restricción, encargada de la capacidad máxima de aviones que puede albergar cada taller / parking para una misma franja horaria, siendo el límite máximo dos aviones y en ningún caso puede ser ambos del tipo Jumbo (JMB). Para llevar a cabo esta restricción, decidimos organizar las posiciones ocupadas en cada taller haciendo uso de una estructura dinámica, esto nos permite manejar de manera eficiente las asignaciones y realizar las comprobaciones necesarias sobre las capacidades de los talleres sin necesidad de recorrer todas las posiciones de forma manual. Las verificaciones realizadas consisten en, primero analizar que ningún taller tenga más de dos aviones asignados. Posteriormente, se evalúa si en las posiciones ocupadas hay más de un avión Jumbo. Por último, para los casos en los que hay exactamente dos aviones, comprobamos que estén conformados por una combinación válida. limitándose a dos aviones estándar o a una combinación de un avión estándar y un avión Jumbo.

En cuanto a lo que respecta a la tercera restricción, la cual se encarga de regular la asignación de tareas de mantenimiento en talleres estándar y especializados. En este caso, los talleres especializados tienen la flexibilidad de realizar tanto tareas estándar como especializadas, mientras que los talleres estándar únicamente pueden realizar tareas estándar. Para garantizar el cumplimiento de esta regla, tomamos la decisión de asociar las tareas especiales de cada avión con talleres de tipo SPC, de manera que, si un avión tiene programada tanto una tarea SPC como una STD, al menos una de las franjas horarias del día debe asignarse a un taller especializado. Esta verificación se realiza evaluando las tareas programadas para cada avión y asociando las franjas horarias correspondientes a talleres especializados cuando existen tareas especiales pendientes a realizar.

Por otro lado, la cuarta restricción nos asigna prioridad en lo que al orden de realización de las tareas se refiere en caso de que el avión lo requiera, ya que en caso contrario, será indiferente si se comienza llevando a cabo las tareas de

tipo 1 de primero o las de tipo 2. Para modelar esta restricción, se asignaron franjas horarias específicas a las tareas de tipo 2 antes que las de tipo 1. De forma que el la se divide las asignaciones del trabajo llevado a cabo en los talleres y evita que se viole la secuencia requerida. Facilitando no solo la tarea de división de restricciones, sino que también facilita el análisis de soluciones válidas, ya que el orden queda visualmente explícito.

En cuanto a la quinta y sexta restricción están relacionadas a la capacidad de movilidad de los aviones entre los distintos talleres / aparcamientos, una encargándose específicamente de la movilidad entre aviones Jumbo y otra restringiendo de igual forma a todos los aviones. Siendo la cuarta la que restringe a todos los aviones, de forma que si un taller o parking está ocupado, al menos una posición adyacente (vertical u horizontal) debe estar vacía, asegurando que los aviones puedan maniobrar de forma segura. Para llevar a cabo la implementación generamos un análisis dinámico, en el cual evaluamos todas las posiciones adyacentes a cada taller o parking ocupado, considerando su disponibilidad en tiempo real durante la generación de las posibles soluciones. Esta dinámica permite verificar si las posiciones contiguas están libres sin necesidad de realizar un análisis exhaustivo de todo el espacio, optimizando así el rendimiento del modelo.

Finalmente, la sexta restricción es la encargada de preservar que dos aviones de tipo Jumbo (JMB) no pueden estar asignados a talleres adyacentes en la misma franja horaria. Esto se debe a las mayores dimensiones de estos aviones y la necesidad de más espacio para maniobrar. El enfoque desarrollado para dicha restricción consistió en modelar esta restricción como una limitación explícita de distancia mínima entre las posiciones ocupadas por aviones Jumbo durante cada franja horaria. Esto implica que, para cada pareja de aviones JMB, se verifica que las posiciones asignadas no sean adyacentes. Garantizando que, en cada intervalo de tiempo, los talleres o espacios ocupados por estos aviones no comprometan la maniobrabilidad.

Parte 2: Planificación con búsqueda heurística

1. Modelización del Problema

El problema de rodaje de aviones lo hemos modelado como un problema de búsqueda en el que se establecen algunas restricciones. Esto permite representar los movimientos de los aviones desde sus posiciones iniciales hasta sus destinos como una secuencia de acciones dentro de un mapa.

Estados:
Un estado describe la situación actual del problema en un momento determinado y está formado por:

- Las posiciones actuales de los aviones en el mapa, indicadas como coordenadas (x,y).

- El camino que cada avión ha realizado hasta ese punto.
- El costo total acumulado para llegar al estado actual.
- Una estimación heurística de cuánto falta para que los aviones lleguen a sus destinos.

Operadores:
En cada iteración, los aviones pueden ejecutar de manera simultánea las siguientes acciones:

- Moverse hacia arriba (↑).
- Moverse hacia abajo (↓).
- Moverse hacia la izquierda (←).
- Moverse hacia la derecha (→).
- Esperar en su posición actual (w).

Restricciones:

- Solo un avión puede ocupar la misma celda en un instante de tiempo.
- No se permite que dos aviones crucen simultáneamente entre dos celdas adyacentes.
- Las celdas marcadas como grises (G) no se pueden transitar.
- Las celdas amarillas (A) tienen que ser transitadas sin parar.

Coste:

Cada acción tiene un coste de 1 unidad de tiempo. Nuestro objetivo es minimizar el makespan, es decir, el tiempo total que todos los aviones necesitan para alcanzar sus destinos.

2. Justificación de las Decisiones Tomadas

Ahora explicaremos las decisiones tomadas para la modelización del problema:

- **Uso del algoritmo A***: El algoritmo A* es una herramienta eficiente para encontrar soluciones óptimas en problemas de búsqueda. Este algoritmo funciona evaluando cada estado mediante una función de coste que permite priorizar los estados más prometedores, reduciendo el espacio de búsqueda.

- **Heurísticas implementadas**:
    1. **Distancia Manhattan**: Hemos elegido la heurística Manhattan porque es rápida, eficiente y fácil de implementar. Esta heurística calcula la suma de las diferencias absolutas entre las coordenadas (x,y) de las posiciones actuales y los destinos. Su simplicidad no solo garantiza un bajo costo computacional, sino que también nos permitió obtener buenos resultados en mapas pequeños. Su funcionamiento será evaluado en los casos de prueba con más detalles.

2. **Floyd-Warshall**: Decidimos implementar esta heurística porque proporciona una mayor precisión al calcular las distancias mínimas entre todas las celdas transitables. Aunque requiere un preprocesamiento más costoso, observamos que fue especialmente útil en mapas más grandes y complejos, donde las rutas óptimas no eran evidentes a simple vista. Su comportamiento lo analizaremos en detalle en los casos de prueba.

- **Validación de movimientos**: La función es_valido verifica que cada movimiento cumpla con las restricciones del problema, como evitar colisiones, cruces simultáneos y el uso de celdas no transitables.
- **Ejecución simultánea de movimientos**: Los movimientos de los aviones se procesan al mismo tiempo en cada iteración, respetando las restricciones y permitiendo que el problema sea modelado como un sistema multiagente.
- **Generación de sucesores**: Utilizamos el producto cartesiano de los movimientos posibles para generar sucesores. Esto asegura que todas las combinaciones válidas se consideren, garantizando que no se omitan posibles soluciones.

Análisis de resultados

Parte 1: Validación con Python - constraint

Para llevar a cabo el análisis de resultados, hemos propuesto distintas entradas para demostrar el comportamiento de nuestro modelo en distintos escenarios. El enfoque que desarrollaremos será realizar casos de prueba en los que se proporcionará una entrada correcta al modelo, por lo que el comportamiento esperado es que se obtengan soluciones factibles. Por otro lado, proporcionaremos escenarios en los que la falta mayor cantidad de datos de entrada, garantizando que no se puedan satisfacer todas las restricciones necesarias, provocando así una salida con cero soluciones factibles.

- Comenzamos con un caso de prueba con entradas muy reducidas (maintenance01), con la finalidad de demostrar de forma visual el cumplimiento de todas las restricciones. Al ser un caso de prueba tan controlado, podemos analizar de forma sencilla cómo se cumplen todas las restricciones. Primero observar las restricciones de movilidad, tanto entre todos los aviones, respetando siempre un taller libre en horizontal o vertical, como entre dos aviones JMB, en los cuales se respeta la no

ocupación de talleres adyacentes. Por otro lado, analizar la capacidad máxima de los talleres, realizando distintas combinaciones de aviones para un mismo taller, pero siempre preservando que sean o dos STD, o un STD y un JMB. Por otro lado podemos observar como se hace el correcto uso del parking cuando los aviones se encuentran sin tareas pendientes a realizar. Finalmente destacar que cuando se activa la condición de prioridad, siempre se asigna al iniciar un taller SPC (en caso de el avión tener tareas de tipo 2) y de no estar activada dicha condición, puede iniciar tanto con un taller STD como con un SPC.

- En el caso de (maintenance02), modificando el escenario anterior y extendiendolo a una matriz de 3x3 con tres franjas horarias, definimos un total de 7 aviones entre STD y JMB, con la finalidad de que al únicamente tener disponibles 9 talleres por cada franja horaria, resulta imposible respetar todas las restricciones de movilidad y obteniendo como resultado cero soluciones factibles.

- Por último hemos implementado un caso de prueba en el cual realizamos un input con valores más exigentes, con la intención de comprobar el correcto funcionamiento no solo con entradas tan sencillas como con el primer caso de prueba, sino exigirle más a nuestro modelo. Para (maintenance03) implementamos 4 franjas horarias, con una matriz de 5x5 y un total de 4 aviones con sus respectivas tareas asignadas, obteniendo como resultado un número total de 209.280, que es un número razonable para la cantidad de combinaciones que podemos realizar con una matriz de dicha cantidad de dimensiones y para nuestro conjunto de aviones. Tras analizar un conjunto de 20 soluciones elegidas al azar, dentro del total de soluciones y analizar minuciosamente el cumplimiento de restricciones, podemos afirmar que nuestro modelo construye soluciones robustas para distintos tipos de entradas.

Por otro lado, destacar que no hemos llevado a cabo esta última prueba con el ejemplo del enunciado, debido a que, es un conjunto muy demandante, necesitando muchísimo tiempo y un requerimiento computacional alto.

.

Parte 2: Planificación con búsqueda heurística

Para el análisis de resultados de la parte 2 hemos llevado a cabo una cantidad significativa de casos de prueba para verificar que nuestra implementación funciona de manera eficiente en los mapas donde se puede encontrar una solución, ya que hay mapas donde nuestro código no va a poder encontrar solución por las restricciones que tenemos.

- **Mapa 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

  - **Estadísticas:**

	Tiempo total	Makespan	h inicial	Nodos expandidos
Distancia de Manhattan	0.00s	9	9	82
Floyd-Warshall	0.07s	9	15	13

  - **Solución para ambas heurísticas:**

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

  - **Análisis:**

  Podemos observar que Manhattan es más rápida en calcular la solución pero menos precisa, es por eso que hay más nodos expandidos. En cambio Floyd-Marshall puede ser un algoritmo más lento y tiene costo mayor inicial pero es más informada y precisa por lo que resulta en una solución con menos nodos expandidos.

- **Mapa 2:**

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

  - **Estadísticas:**

	Tiempo total	Makespan	h inicial	Nodos expandidos
Distancia de Manhattan	0.01s	5	13	307
Floyd-Warshall	0.15s	5	13	6

  - **Solución para ambas heurísticas:**

(1,0) ↓ (2,0) ↓ (3,0) → (3,1) → (3,2) → (3,3)
(3,0) → (3,1) → (3,2) → (3,3) ↑ (2,3) ↑ (1,3)
(2,1) ↑ (1,1) ↑ (0,1) → (0,2) w (0,2) w (0,2)

  - **Análisis:**

Podemos observar que Manhattan es más rápida en calcular la solución debido a su simplicidad, pero menos precisa, lo que resulta en mayor cantidad de nodos expandidos. En cambio, Floyd-Warshall es un algoritmo más lento inicialmente, pero al ser más informada y precisa, logra expandir menos nodos durante la búsqueda.

- **Mapa 3:**

4
(0,0) (4,4)
(4,0) (0,4)
(0,4) (4,0)
(4,4) (0,0)
B;B;B;B;B
B;G;B;A;B
G;B;G;B;B
B;G;B;B;G
B;B;A;B;B

  - **Estadísticas:**

	Tiempo total	Makespan	h inicial	Nodos expandidos
Distancia de Manhattan	0.80s	22	32	11694
Floyd-Warshall	6.64s	12	32	18

  - **Solución:**

    - Distancia de Manhattan:

AVION 1: (0,0) ↓ (1,0) w (1,0) ↑ (0,0) → (0,1) → (0,2) ↓ (1,2) → (1,3) → (1,4) ↓ (2,4) w (2,4) w (2,4) ← (2,3) ↓ (3,3) ↓ (4,3) → (4,4) w (4,4) w (4,4) w (4,4) w (4,4) w (4,4) w (4,4)

AVION 2: (4,0) ↑ (3,0) w (3,0) w (3,0) w (3,0) w (3,0) w (3,0) w (3,0) w (3,0) w (3,0) w (3,0) w (3,0) ↓ (4,0) → (4,1) → (4,2) → (4,3) ↑ (3,3) ↑ (2,3) ↑ (1,3) → (1,4) ↑ (0,4)

AVION 3: (0,4) ↓ (1,4) ↓ (2,4) ← (2,3) ↓ (3,3) ↓ (4,3) ← (4,2) ← (4,1) w (4,1) w (4,1) w (4,1) w (4,1) w (4,1) w (4,1) → (4,2) ↑ (3,2) w (3,2) w (3,2) ↓ (4,2) ← (4,1) ← (4,0) w (4,0) w (4,0)

AVION 4: (4,4) ← (4,3) ↑ (3,3) ← (3,2) w (3,2) w (3,2) w (3,2) w (3,2) → (3,3) ↑ (2,3) ↑ (1,3) ← (1,2) ↑ (0,2) ← (0,1) ← (0,0) w (0,0) w (0,0) w (0,0) w (0,0) w (0,0) w (0,0) w (0,0) w (0,0)

- Algoritmo Floyd-Warshall:

AVION 1: (0,0) → (0,1) → (0,2) → (0,3) w (0,3) w (0,3) w (0,3) w (0,3) ↓ (1,3) ↓ (2,3) ↓ (3,3) ↓ (4,3) → (4,4)

AVION 2: (4,0) → (4,1) → (4,2) ↑ (3,2) → (3,3) ↑ (2,3) ↑ (1,3) → (1,4) ↑ (0,4) w (0,4) w (0,4) w (0,4) w (0,4)

AVION 3: (0,4) ↓ (1,4) ↓ (2,4) w (2,4) w (2,4) w (2,4) w (2,4) ← (2,3) ↓ (3,3) ↓ (4,3) ← (4,2) ← (4,1) ← (4,0)

AVION 4: (4,4) ← (4,3) ↑ (3,3) ↑ (2,3) ↑ (1,3) ← (1,2) ↑ (0,2) ← (0,1) ← (0,0) w (0,0) w (0,0) w (0,0) w (0,0)

- **Análisis:**

Al igual que en el mapa 1 y mapa 2, en este caso en el cual tenemos un mapa más grande y con más aviones, la heurística de Manhattan es más rápida en calcular la solución, pero menos precisa, lo que provoca una gran cantidad de nodos expandidos. Por otro lado, Floyd-Warshall, aunque es más lento debido a su etapa inicial de cálculo de distancias, es más informada y eficiente, resultando en una solución con significativamente menos nodos expandidos.

- **Mapa 4:**

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

- **Estadísticas:** No se encontró solución

- **Solución:** No se encontró solución
- **Análisis:**

En este caso, no se ha encontrado una solución debido a que existe una fila completa de celdas grises (G), que bloquea completamente las rutas necesarias para que los aviones lleguen a sus destinos. Esto demuestra que, independientemente de la heurística utilizada, el diseño del mapa es clave en la posibilidad de encontrar una solución.

A través de estos casos de prueba, podemos observar que la elección de la heurística influye significativamente en el rendimiento del algoritmo A*. Mientras que Manhattan es más rápida en términos de cálculo inicial, su simplicidad puede resultar en una mayor cantidad de nodos expandidos. Por otro lado, Floyd-Warshall, aunque es más lento, suele ser más eficiente en la búsqueda cuando los mapas presentan obstáculos más complejos. Además, el diseño del mapa es un factor clave en la posibilidad de encontrar soluciones, independientemente de la heurística utilizada.

Conclusión

En esta práctica hemos abordado problemas de planificación y optimización mediante el uso de modelado de restricciones y técnicas de búsqueda heurística. En la Parte 1, tuvimos que modelar el mantenimiento de aviones como un CSP, donde hemos garantizado que se respetaran restricciones críticas, como la capacidad de los talleres, el orden de las tareas y la correcta asignación de maniobras. Finalmente, hemos realizado una implementación efectiva que generó soluciones válidas y prácticas para los casos diseñados.

En la Parte 2, hemos realizado el diseño e implementación de un algoritmo A* con dos heurísticas admisibles, lo que permitió observar cómo estas afectan la rapidez y precisión. Durante el desarrollo nos dimos cuenta de que la complejidad de los mapas hacía más difícil encontrar soluciones. Comparar las heurísticas de Manhattan y Floyd-Warshall permitió analizar de manera práctica las diferencias entre tiempo de cálculo inicial y eficiencia en la búsqueda.