22 de diciembre de 2020

HEURÍSTICA Y OPTIMIZACIÓN

PRÁCTICA 2: SATISFACCIÓN DE RESTRICCIONES Y BÚSQUEDA HEURÍSTICA

Grupo Reducido 82

Cristina Piña Miguelsanz, 100406007
David Juárez Martínez-Villanueva, 100405824

ÍNDICE

1) Introducción

El objetivo de esta práctica es aplicar y ampliar los conocimientos sobre problemas de satisfacción de restricciones (CSP) así como de búsqueda heurística.

Para ello, se desarrollarán dos partes: Una primera parte en la que se hará uso de la librería python-constraints para solventar un problema CSP y una segunda parte donde se desarrollarán los conocimientos de búsquedas heurísticas implementando dos heurísticas diferentes en Java.

Tras modelizar e implementar estas dos partes se realizará un análisis de los resultados, así como una serie de conclusiones acerca de la práctica.

2) Descripción de los modelos

a) Parte 1: Asignación de antenas de transmisión a satélites

i) Modelización del problema

En este problema deseamos realizar una asignación de satélites y sus franjas horarias a antenas de transmisión.

Datos e información del problema:

- Tenemos 6 satélites y 12 antenas de transmisión.
- Cada satélite dispone de franja/s horaria/s en las que son "visibles" por diversas antenas, tal y como se muestra en la siguiente figura:

Satélite	Franja	Antenas
SAT1	00:00 - 12:00	ANT1, ANT2, ANT3, ANT4
SAT2	00:00 - 12:00	ANT1, ANT2, ANT3
SAT3	06:00 - 12:00	ANT4, ANT6
SAT3	13:00 - 16:00	ANT7, ANT9, ANT10
SAT4	16:00 - 00:00	ANT8, ANT11, ANT12
SAT5	06:00 - 13:00	ANT1, ANT7, ANT12
SAT6	09:00 - 13:00	ANT7, ANT9
SAT6	13:00 - 19:00	ANT3, ANT4, ANT5

Tabla 1: Satélites, franjas horarias y antenas

Definición del modelo:

El problema a resolver se trata de un problema de satisfacción de restricciones (CSP). Esto puede definirse mediante una terna (X, D ,C), donde X es un conjunto **finito** de variables, representado de la siguiente manera: $X=\{x_i\}^n_{i=1}$.

Variables X = (SAT1, SAT2, SAT31, SAT32, SAT4, SAT5, SAT61, SAT62).

Las variables representan cada uno de los satélites en cada una de sus franjas horarias. Las variables han sido nombradas como "SATY", donde "Y" hace referencia al número de satélite.
En algunos casos, como en los satélites 3 y 6, se asigna más de una franja horaria, por lo que añadiremos un carácter adicional "Z" de la manera "SATYZ", que hace referencia a cada una de estas franjas. Por ejemplo, SAT31 hace referencia al satélite 3 en su primera franja horaria y SAT 32 hace referencia a este mismo satélite 3 pero en su segunda franja horaria.

Dominio Di = { antenas para las que el satélite en cierta franja es visible }

Cada variable está definida sobre un conjunto de dominios D_i . Cada D_i está formado por unos valores, que deben representarse de la forma más clara posible de cara a encontrar una solución en base a la satisfacción de ciertas restricciones. Como es lógico, es imprescindible añadir el número de antena asignada a cada uno de los satélites. Por lo tanto, al principio se podría denotar cada valor como "A", el cual podrá tener un valor comprendido entre 1 y 12 dependiendo de las antenas que tenga disponibles cada satélite para escoger en la franja asignada.
Sin embargo, esta definición no es suficiente para satisfacer todas las restricciones que se nos plantean. Concretamente, en una de las restricciones nos piden que si ocurre un determinado suceso, la franja horaria debe ser la misma. Si observamos la información que se tiene del problema, podemos distinguir dos tipos de franjas: aquellas que comienzan antes de las 12 de la mañana (franja horaria de mañana), y aquellas que comienzan después de las 12 (franja horaria de tarde). Concluimos denotando cada uno de los valores de cada dominio como "FA", donde "F" es la franja horaria a la que pertenece cada satélite con un valor (F = 1) si la franja en la que se opera es de mañana o (F=2) si es de tarde.

Por tanto, los valores D_i posibles que puede tener cada variable se muestran a continuación:

Dsat1 = {11, 12, 13, 14}
Dsat2 = {11, 12, 13}
Dsat31 = {11, 16} / **Dsat32** = {27, 29, 210}
Dsat4 = {28, 211, 212}
Dsat5 = {11, 17, 112}
Dsat61 = {17, 19} / **Dsat62** = {23, 24, 25}

Restricciones:
Las restricciones asociadas a este problema son las siguientes:

1) $\mathbf{x_{SAT1} = x_{SAT2}}$ A SAT1 y SAT2 se les debe asignar la misma antena.

2) $x_{SAT2} \neq x_{SAT4}$, $x_{SAT2} \neq x_{SAT5}$, $x_{SAT4} \neq x_{SAT5}$ A SAT2, SAT4 y SAT5 se les debe asignar antenas diferentes.

3) x_{SAT5}="112" ? $x_{SAT4} \neq$"211" Si a SAT5 se le asigna ANT12, a SAT4 no se le puede asignar ANT11 (el símbolo ? representa la acción a realizar si se cumple la expresión del principio).

4) (($x_{SAT4.A}$="7" and $x_{SAT5.A}$="12") or ($x_{SAT4.A}$="12" and $x_{SAT5.A}$="7")) and $x_{SAT5.A} = x_{SAT5.F}$
Si a una solución se asignan las antenas ANT7 y ANT12, se deben asignar ambas a franjas horarias que comiencen antes de las 12:00 o franjas horarias que comiencen después de las 12:00. Aplicable entre SAT4 y SAT5, SAT32 y SAT4/SAT5 o SAT61 y SAT5/SAT4.

5) $x_i \neq \{\Phi\}, \forall_i \in X$ Todos los satélites deben tener asignada una antena de transmisión para cada una de sus franjas horarias visibles.

NOTA: Para verificar las restricciones 1) y 2), supondremos que el valor de la franja horaria nos es indiferente, teniendo en cuenta únicamente el valor de la antena como se relata en el enunciado.

ii) Implementación del algoritmo

Este algoritmo lo implementamos en Python haciendo también uso de la librería *python-constraint*. Primero de todo importamos la librería y, a continuación, creamos el problema como objeto.

A continuación, modificaremos la información mostrada en la Tabla1. Para la creación de cada variable y la posible asignación de valores que podría tomar hacemos uso de *addVariable*.
El valor de cada una de las variables se almacenará en una cadena de caracteres.

Finalmente, codificamos las restricciones mostradas anteriormente. Cada una de ellas será añadida mediante el uso de la función *addConstraint,* encargada de ejecutar la función pasada como parámetro.

Cabe destacar que para la implementación de ciertas restricciones necesitamos desplazar los posibles valores a asignar, con el fin de comparar la franja horaria o el número de antena asignado a cada satélite. Para ello, utilizamos el concepto de "slicing", que consiste en obtener el valor de la cadena de caracteres a partir de un número "n" de posiciones hacia la derecha que deseemos.

Una vez codificado el problema, podemos imprimir una de las soluciones obtenidas mediante *getSolution*, o bien devolver una lista con todas y cada una de las soluciones mediante la función *getSolutions*. Si deseamos saber cuál es el número total de soluciones obtenido usaremos la función *len* de python, cuyo argumento será la lista adquirida con *getSolutions*.

b) Parte 2: Búsqueda Heurística

La implementación de este problema se realiza en Java, al ser el lenguaje de programación más conocido por ambos integrantes de la práctica.

i) Modelización del problema

1. Datos del problema

EL objetivo de este problema es el de transmitir a base todas las observaciones programada, mediante el uso de dos satélites, SAT1 y SAT2.

Hay que tener en cuenta que cada satélite sólo puede hacer una operación a la vez y que cada operación se tarda una hora en realizar.

Los costes asociados a cada acción (energía gastada) se reciben como parámetro de entrada al problema, y no son necesariamente iguales para ambos satélites. Se recibe también como parámetro de entrada las unidades que se recargan en cada operación de carga, así como la batería máxima de cada satélite.

Las acciones que puede realizar cada uno de los satélites según la definición del problema son IDLE (no hacer nada), observar una muestra, transmitir una muestra a la base, girar (para cambiar de banda de observación) y cargar (para recargar la batería del satélite, que se va gastando al realizar acciones). Las precondiciones y efectos de cada una de estas acciones serán explicados en el apartado inferior de definición de operadores.

2. Representación de los estados

Definimos cada estado como una tupla: E = {CSAT1, BSAT1, OSAT1, CSAT2, BSAT2, OSAT2, OT, Hora}

Donde:

- *CSAT1* indica el nivel de carga de la batería del satélite 1.
- *BSAT1* = {01, 12} e indica la banda de observación del satélite 1.
- *OSAT1* Observaciones tomadas por SAT1 y aún no transmitidas.
- *CSTA2* indica el nivel de carga de la batería del satélite 2.
- *BSAT2* = {23, 12} e indica la banda de observación del satélite 2.
- *OSAT2* Observaciones tomadas por SAT2 y aún no transmitidas.
- *OT* Observaciones ya transmitidas a la base.
- *Hora* = [0 … 23] e indica la hora en la que nos encontramos.

3. Definición de los operadores

A continuación, definimos los operadores, así como sus precondiciones y efectos. Tenemos 5 acciones diferentes (IDLE, Observa OZ, Transmite OZ, Gira, Carga) y 2 satélites. Lo cual nos daría

un total de 5 x 5 = 25 operadores distintos. Para simplificar la posterior programación, hemos decidido crear un único operador al cual le pasamos como parámetro las acciones a realizar.

Operador: **realizarAcciones(operaciónSAT1, obvsSAT1, operaciónSAT2, obvsSAT2)**

Los parámetros obvsSAT1 y obvsSAT2 solo se usan en caso de que se esté realizando una observación (ya que se debe indicar qué muestra se observa), en caso contrario ese parámetro se pasará como nulo. A la hora de realizar transmisiones, se transmite por defecto la más antigua.

- **Precondiciones y efectos**

A continuación, mostraremos las precondiciones que deben cumplirse para que se puedan realizar las acciones y los efectos de aplicarlas, tanto en SAT1 como en SAT2. Mostraremos sólo las operaciones del primer satélite, debido a que las del segundo satélite son equivalentes, realizándolas sobre parámetros de SAT2.

IDLE:

Precondiciones	Efectos
No tiene precondiciones.	Hora = Hora + 1

Observa OZ: Observa una muestra OZ (O1, O2, …)

Precondiciones	Efectos
Hora >= 0 && Hora < 12	Hora = Hora + 1
CSAT1 >= gasto_obvs_SAT1	CSAT1 = CSAT1 - gasto_obvs_SAT1
OZ debe estar la misma banda y hora que SAT1.	OSAT1 = OSAT1 + OZ
OZ noE OT && OZ noE OSAT1 && OZ noE OSAT2	

Transmite: Se transmite la observación OZ hecha hace más tiempo

Precondiciones	Efectos
Hora >= 0 && Hora < 12	Hora = Hora + 1
CSAT1 >= gasto_transm_SAT1	CSAT1 = CSAT1 - gasto_transm_SAT1.
OZ E OSAT1	OSAT1 = OSAT1 - OZ && OT = OT + OZ

Gira: Cambia de banda de observación

Precondiciones	Efectos
Hora >= 0 && Hora < 12	Hora = Hora + 1
CSAT1 >= gasto_giro_SAT1	CSAT1 = CSAT1 - gasto_giro_SAT1
	Banda SAT1 cambia

Carga

Precondiciones	Efectos
Hora >= 0 && Hora < 12	Hora = Hora + 1
CSAT1 < CSAT1_max	CSAT1 = CSAT1 + energia_cargaSAT1 (o BSAT1_max si se supera).

4. Definición de objetivo a optimizar:

Los parámetros que pueden ser optimizados son la longitud del plan, es decir, el tiempo (horas) que se tarda en hacer todas las retransmisiones, y la energía consumida por los satélites.
En nuestro caso, crearemos unas heurísticas para optimizar la longitud del plan.

5. Definición de heurísticas admisibles

Las heurísticas pueden plantearse en base a varios criterios. En este caso, las heurísticas a explicar se basan en la relajación de las distintas restricciones que se exponen en el problema, facilitando la resolución de este. Las heurísticas, creadas mediante la relajación de algunas de estas restricciones, son las siguientes:

a. Heurística h_1

En esta heurística relajamos todos los grupos de precondiciones: no vamos a tener en cuenta la banda de observación asociada, al igual que las horas en las que se puedan realizar operaciones los satélites y sean visibles por la estación base y el gasto de unidades de energía por cada operación. Lo único que se tendrá en cuenta es el número de observaciones del terreno que falten por observar y transmitir, teniendo en cuenta que cada satélite solo puede observar u transmitir una muestra cada hora. El coste de la función heurística h(n,t) de cada nodo es el siguiente: $\mathbf{h_1(n,t)=(2*}\boldsymbol{NoObservados + ObservadosNoTransmitidos}\mathbf{)/2}$. La deducción de esta fórmula proviene de esta forma: *NoObservados* son las observaciones del terreno que todavía no se han realizado, mientras que *ObservadosNoTransmitidos* son aquellas observaciones que ya se han realizado pero que todavía no han sido transmitidas a la estación base. El coste en horas tanto para observar como para transmitir es unitario. Sin embargo, una vez se realice una observación, debemos tener también en cuenta que se realizará una acción más para transmitirla a la estación base, por lo que el coste de los elementos no observados será el doble.

Por último, tenemos dos satélites capaces de completar acciones. Al relajar las precondiciones, cualquiera de los satélites puede operar con cualquier observación, en cualquier hora y sin ningún tipo de gasto, por lo que el coste total de observación y transmisión se reduce a la mitad.

b. Heurística h_2

La creación de esta heurística está creada en vista de la heurística anterior. En este caso, relajaremos la mayoría de grupos de precondiciones, a excepción de la relacionada con la banda de observación, por lo que en algunos casos será necesario que el satélite gire para captar la observación. El coste asociado $h_2(n,t)$ asociado a cada nodo depende de si hubiera una observación disponible en la banda de alguno de los astros . Si la hubiera, estaríamos en las mismas condiciones que en h_1, y a consecuencia de ello su valor sería el mismo: $\mathbf{h_2(n,t)=(2*}\boldsymbol{NoObservados}\mathbf{ + }\boldsymbol{ObservadosNoTransmitidos}\mathbf{)/2}$. En el caso contrario, sería necesario que al menos uno de los satélites girara, aumentando en uno su coste, siendo este el coste total: $\mathbf{h_2(n,t)=(2*}\boldsymbol{NoObservados}\mathbf{ + }\boldsymbol{ObservadosNoTransmitidos}\mathbf{)/2 + 1}$.
h_2 estará más informada que h_1 al demostrar que $h_2(n) \geq h_1(n) \forall n$.

ii) Implementación del algoritmo

Como hemos explicado al principio de este apartado, decidimos implementar el modelo creado en lenguaje Java. Se ejecuta poniendo en terminal **./Cosmos.sh <problema.prob> <heurística>** donde <heurística> será **heuristica1** o **heuristica2** (o cualquier otro string si queremos probar con h=0) y <problema.prob> será del tipo **./ejemplos/problema.prob** en caso de almacenar los .prob en una carpeta "ejemplos". En Cosmos.sh se puede encontrar una mayor explicación sobre cómo realizar la ejecución.

A continuación, describiremos las distintas clases que hemos implementado para resolver el problema:

Parseador.java: Se encarga de, pasado un fichero como parámetro (problema.prob) obtener los datos sobre el problema que proporciona. Destacan dos de sus métodos: *parseSAT*, que sirve para obtener los datos proporcionados sobre cada satélite, y *parseMuestras*, que se encarga de devolver las muestras a recoger. Una posible estructura del fichero de entrada es la siguiente:

```
OBS: (0,1);(1,1) : Banda y hora a la que pertenece cada observación
SAT1: 1;1;1;1;1  :  En orden representan: coste observación, transmisión, giro,
SAT2: 1;1;1;1;1     unidades recargadas en carga y máximo de batería.
```

Satelite.java: Define el conjunto de atributos de un satélite: el coste de realizar las acciones "observa", "transmite" y "gira", las unidades de energía obtenidas al ejecutar el operador "carga" y las bandas admitidas((0,1) y (1,2) en SAT1, y (2,3) y (1,2) en SAT). Usaremos una matriz de enteros para almacenar estas tuplas.

MuestraObservacion.java: Define el conjunto de atributos de una muestra: la banda de observación en la que se encuentra, hora en la que se puede observar y un identificador, para distinguir correctamente a qué observación nos referimos.

Problem.java: Es la encargada de guardar los datos iniciales del problema pasado por el fichero de entrada "problema.prob". Contiene una lista *muestrasAobservar* con todas las muestras que deben transmitirse a la base para finalizar el problema. Contiene también dos objetos de tipo SATelit (*SAT1* y *SAT2*), el estado inicial S, la heurística mediante la cual se resuelve, y un String final con las diferentes acciones que puede realizar un satélite.

Finalmente, contiene un constructor al cual se llama pasando como parámetro el problema.prob y que hace uso del anteriormente comentado parseador, así como un método para comprobar si un estado es final (método *isFinal*).

Estado.java: Representar la situación del problema en un instante específico, en función de la hora en la que se encuentre y las acciones realizadas. Incluye un objeto de la clase problema, una lista con las observaciones ya transmitidas a la base, la hora en ese instante, la carga y banda tanto del primer como segundo satélite, listas para almacenar las observaciones tomadas pero no transmitidas por los satélites y el valor f(n), igual a la suma de las variables g(n) y h(h) donde g(n) es la distancia del nodo inicial hasta el nodo en ese momento. Cuenta también con una referencia al estado padre (nulo en el caso del estado inicial), así como un string con la operación mediante la cual se generó el estado.

Esta clase contiene dos constructores (una para el estado inicial, y otro para el resto de estados), tres funciones de cálculo de costes (coste g, h, y f) y una función para crear los sucesores del estado, así como otros métodos usados dentro de este último. De entre todas estas funciones, destacan la calculadora del coste heurística h *calcularCosteHeurístico* que implementa las heurísticas definidas en el apartado 2.2.1, y la función de creación de sucesores *crearSucesores*, que intenta crear todos los sucesores posibles, para ello verificaremos las acciones válidas posibles, llamaremos a la función auxiliar *realizarAcciones* , que utiliza otro método auxiliar *validPreconditions* para ver si se cumplen las precondiciones necesarias (usando ComprobadorPrecondiciones) , implementará los cambios pertinentes mediante *implementEffects* (con ImplementadorEfectos) y por último volverá a *crearSucesores* donde creará los sucesores hijos con los valores de sus atributos actualizados.

CosmosMain.java: esta clase es la que contiene el método main, donde crea el problema y llama al algoritmo A* mediante una función implementada en esa misma clase. El método se denomina *Astar*, y su estructura sigue el siguiente pseudocódigo:

```
ABIERTA=I, CERRADA=Vacío, EXITO=Falso
Hasta que ABIERTA esta vacío O EXITO
    Quitar el primer nodo de ABIERTA, N
    SI N es Estado-final ENTONCES EXITO=Verdadero
    SI NO Expandir N y meterlo en CERRADA, generando el conjunto S de sucesores de N
        Para cada sucesor s en S
            1. Si s no está ni en ABIERTA y CERRADA se inserta en orden en ABIERTA
            2. Si está en ABIERTA y la función de evaluación f() de s es mejor, se elimina
            el que ya estaba en ABIERTA y se introduce s en orden
            3. Si está en CERRADA se ignora
Si EXITO Entonces Solución=camino desde N a I a través de los punteros
Si no Solución=Fracaso
```

Usamos métodos auxiliares para calcular si nos encontramos en el estado final, encontrar el camino final desde un estado inicial al final u ordenar los distintos nodos que vayamos almacenando con el fin de reducir el tiempo de ejecución. Se tiene control de errores, verificando que se pasen de manera correcta los parámetros necesarios.

FileManager.java y Printer.java: Las siguientes clases están destinas a, una vez se encuentre una solución, crear los ficheros "problema.prob.output"y "problema.prob.statistics" y redirigir distintos datos de interés en cada uno de ellos.

ComprobadorPrecondiciones.java: Comprueba, para cada acción que puede realizar un satélite (IDLE, Observar, Transmitir, Cargar, Girar) si se cumplen las precondiciones necesarias para realizarla, devolviendo true o false.

ImplementadorEfectos.java: Aplica sobre un estado sucesor los efectos derivados de haber aplicado a su estado padre las acciones para generar este nuevo estado.

c) Análisis de resultados

3.1 Resolución y análisis de la parte 1

Una vez modelado el problema, procedemos a analizar los resultados obtenidos, así como realizar varios casos de prueba a raíz del problema inicial. Sin restricciones, el número total de posibles soluciones sería de $4x3^5x2^2=3888$. Con los datos proporcionados en el caso propuesto, obtenemos un total de **468 posibles soluciones**, donde se le asigna a cada variable un determinado valor en cada una de ellas. Debido a que se trata de un problema de decibilidad, lo único que nos interesa es saber si se llega a una o varias soluciones y la asignación de variables para llegar a estas, por lo que no se tiene en cuenta el coste que tendría llegar a una u otra. A continuación, modificamos los datos iniciales con el fin de estudiar si hay una variación en el resultado y en qué medida lo hace.

Caso de Prueba 1: Consiste en extender el dominio de ciertas variables sin variar el número de antenas totales, es decir, el/los satélites escogidos poseerán más antenas a las que podrían ser asignados. Por ejemplo, modificaremos el dominio de los satélites 2, 4 y 5, debido a que son satélites que tienen protagonismo en la mayoría de restricciones:
Dsat2 = {11, 12, 13, 14} **Dsat4** = {28, 211, 212,29} **Dsat5** = {11, 17, 112,14}
Si ejecutamos el programa de nuevo, observamos que prácticamente se triplican el número de soluciones posibles, con 1380 en total. La variación es muy elevada, pero tiene lógica por varias razones. El satélite 2 aumenta su dominio en un valor, que precisamente se encuentra en el primer satélite, haciendo que aumenten las asignaciones legales posibles por la restricción 1) explicada en apartados anteriores. Además, el valor de este satélite debe ser distinto al de los satélites 4 y 5 , a los cuales también se les ha añadido un valor a su dominio y tienen más opciones para que se cumpla la restricción 2). También habría que tener en cuenta que, sin restricciones, el número de soluciones ascendería de $4^4x3^2x2^2=9216$

Caso de Prueba 2: Volvamos al caso inicial propuesto, y esta vez reduciremos el dominio de una de las variables, para ver el efecto que tiene en el resultado final. En este caso, al satélite 4 no se le podrá asignar la antena 11: **Dsat4** = {28, 212}
El número de soluciones resultante se reduce a aproximadamente la mitad con 288 soluciones posibles. Con un posible valor menos, se reduce el número total de opciones en las que poder elegir. Esto hará que sea más difícil encontrar soluciones en las que el valor de los satélites 2, 4 y 5 difiera (restricción 2)). Por último, la restricción 4), en la que la antena asignada al satélite 4 no sea la 11 si en el satélite 5 se asigna la antena 12, carece de sentido, viendo que nunca llegará a alcanzar ese valor al ser suprimido.

Caso de Prueba 3: Los posibles casos de prueba no solo se reducen a la extensión o reducción de dominios de una variable, también es posible probar qué pasaría si añadimos o quitamos satélites. A la hora de eliminar satélites, debemos tener en cuenta dos factores: el primero de ellos es en que el número total de soluciones posibles si no hubiera restricciones sería mucho menor. Sin embargo, también debemos tener en cuenta la influencia que tiene el satélite en las restricciones implementadas. Por ejemplo, el satélite 6 en su franja de mañana (SAT62) no se ve influenciado por ninguna de las restricciones implementadas, por lo que si la eliminamos el número de soluciones final será 468/3=156, donde "3" son los tres valores posibles de antenas que se le podrían haber asignado. Si quitamos el satélite 2, también debemos modificar las restricciones 1) y 2) , eliminando esta variable de ellas y dando libertad al primer satélite de asignar cualquier valor dentro de su dominio. Como resultado final, se obtienen 744, muchas más que en el caso inicial propuesto. Es cierto que el número de soluciones si no hubiera restricciones disminuye, pero también lo es que se imponen menos restricciones dejando así que podamos hacer un mayor número de asignaciones válidas.

3.2. Resolución y análisis de la parte 2

Una vez concluida la modelización del problema propuesto y su respectiva implementación en el lenguaje Java en función del algoritmo A*, llevaremos a cabo una serie de casos de prueba que nos permitirán analizar la eficiencia de nuestro programa por medio de varios parámetros como el tiempo transcurrido, operaciones ejecutadas y nodos expandidos. Además, cada caso creado se ejecutará con las dos heurísticas definidas con el objetivo de comparar los resultados obtenidos entre uno y otro, cuál de las dos será más eficiente y por qué. **Nota:** Si el parámetro pasado como argumento no coincide con los nombres de las dos heurísticas, la heurística será nula y se ejecutará el problema con un algoritmo de fuerza bruta (en amplitud, h=0).
Una de las maneras de verificar el funcionamiento del programa es modificar los parámetros que pasemos en el fichero de entrada "problema.prob", creando así múltiples combinaciones donde varía el número y/o posición de las observaciones, configuración de los satélites , etc.

Caso de Prueba 1 (problema1.prob): Consiste en colocar cuatro observaciones: las dos primeras se encontrarán en la misma banda y a la misma hora, mientras que las dos siguientes se ubicará en la banda restante, a una hora diferente alejada de las anteriores. En este contexto, se deben realizar varios giros para tomar las observaciones a la vez siempre y cuando dispongan de la batería necesaria (si no, deberán transitar por las 12 horas de sombra). Como ya sabemos, en h_1 se relajan todas las restricciones, mientras que en h_2 no se relaja la restricción relacionada con la banda a la que están asociados los satélites (está más informada y se aproxima más al problema real). A través de esta justificación, h_2 deberá obtener mejores resultados que h_1. En este caso, también añadiremos el caso en el que la heurística sea nula con intención de verificar que, efectivamente, el uso de heurísticas nos ayuda a encontrar una solución de manera mucho más eficiente. Los datos del problema inicial serán los siguientes:

OBS: (0,1);(1,1);(2,4);(3,4)
SAT1: 1;1;1;1;5
SAT2: 1;1;1;1;5

Ejecutamos este parámetro y, en función de la heurística usada encontramos las siguientes soluciones:

Heurística	Tiempo total (ms)	Coste total	Longitud	Nodos expandidos
Heurística nula	40000	7	7	13930
Heuristica1	556	7	7	977
Heuristica2	481	7	7	941

Y los siguientes planes de ejecución:

Heurística nula	Heuristica1	Heuristica2
SAT1: IDLE, SAT2: Gira SAT1: Observa O1, SAT2: Observa O2 SAT1: IDLE, SAT2: Transmite SAT1: Gira, SAT2: Gira SAT1: Observa O3, SAT2: Observa O4 SAT1: Transmite, SAT2: Carga SAT1: Transmite, SAT2: Transmite	SAT1: IDLE, SAT2: Gira SAT1: Observa O1, SAT2: Observa O2 SAT1: Transmite, SAT2: Gira SAT1: Gira, SAT2: Carga SAT1: Observa O3, SAT2: Observa O4 SAT1: Transmite, SAT2: Transmite SAT1: IDLE, SAT2: Transmite	SAT1: IDLE, SAT2: Gira SAT1: Observa O1, SAT2: Observa O2 SAT1: Transmite, SAT2: Gira SAT1: Gira, SAT2: Carga SAT1: Observa O3, SAT2: Observa O4 SAT1: Transmite, SAT2: Transmite SAT1: IDLE, SAT2: Transmite

Con estos resultados reafirmamos las ideas planteadas al principio: la introducción de una heurística mejora significativamente el rendimiento del problema, siempre y cuando esté bien implementada. Se observa que h_2 mejora la eficiencia de h_1 al estar más informada (se aproxima más al problema real) .

Caso de Prueba 2 (problema2.prob): Los datos del problema serán similares al anterior caso, a excepción de la posición en las bandas y la hora de posible observación de cada una de las observaciones. Para esta prueba, dos de las observaciones estarán en la primera banda, mientras que las otras dos se encontrarán en la segunda. Sin embargo, cada una de ellas será la única a observar dentro de su hora, evitando que se realicen giros entre los satélites. En base a los conocimientos detallados en la definición de las heurísticas, h_2 estará más informada que h_1 siempre y cuando algún satélite realice una función de giro. No obstante, el valor heurístico en sus nodos será el mismo si no se girara como en este caso en concreto. Como conclusión, podemos decir que los resultados obtenidos deberán ser iguales en cuestión de nodos expandidos, longitud del plano y coste total, y con una diferencia despreciable de tiempo (estado de la máquina que ejecute los casos en ese preciso momento, por ejemplo). Los datos del problema, así como los resultados obtenidos, se muestran a continuación. Al ser el plan resultante igual para ambas heurísticas, solo se muestra una vez:

```
1. SAT1: IDLE, SAT2: IDLE
2. SAT1: IDLE, SAT2: IDLE
3. SAT1: IDLE, SAT2: IDLE
4. SAT1: Observa O1, SAT2: IDLE
5. SAT1: Transmite, SAT2: Observa O3
6. SAT1: IDLE, SAT2: IDLE
7. SAT1: IDLE, SAT2: Transmite
8. SAT1: IDLE, SAT2: Observa O4
9. SAT1: IDLE, SAT2: IDLE
10. SAT1: Observa O2, SAT2: Transmite
11. SAT1: Transmite, SAT2: IDLE
```

OBS: (0,3);(1,9);(2,4);(3,7)
SAT1: 1;1;1;1;5
SAT2: 1;1;1;1;5

Heurística	Tiempo (ms)	Coste	Longitud	Nodos expandidos
Heuristica1	6367	11	11	7159
Heuristica2	6518	11	11	7159

Tal y como se esperaba, al no producirse ninguna operación de giro el cálculo de sus costes es equivalente, por lo que expandirán los mismos nodos en un periodo de tiempo muy similar.

Caso de Prueba 3 (problema3.prob): En estos ejemplos anteriores nos hemos centrado en cómo actuaban las heurísticas en función de la disposición en la que se ubicaban las observaciones. No obstante, la configuración inicial del satélite de los estos casos de prueba es muy importante de cara al resultado final. Si nos fijamos bien en los ficheros de entrada de los problemas, la batería máxima de cada satélite y el poco coste de las posibles acciones permitía que se pudiesen observar y transmitir todas las operaciones con un coste y longitud del plan muy bajo, sin tener apenas que recargar sus baterías para efectuar más acciones.
Por ejemplo, si cambiáramos la configuración de los satélites y no se tuviera la suficiente batería para tomar una observación en las horas de luz o no diera tiempo suficiente a transmitirla a la base, se debe transitar por las horas de sombra hasta llegar a una solución, aumentando exponencialmente los costes globales del problema. Esto también podría suceder, si se añadieran más observaciones y/o estas se colocarán dentro de las horas finales de luz, faltando tiempo suficiente para terminar sus tareas dentro de ese periodo.

Para ejemplificar estos hechos, modificaremos la configuración de los satélites del primer caso de prueba, bajando su batería máxima a una única unidad. Esto, principalmente, impedirá que se puedan observar y transmitir todas las operaciones en un corto periodo de tiempo y tenga que transitar durante las horas de sombra hasta volver a ser visibles por la estación base. Esto implica que el coste, tanto en tiempo como en longitud, energía y memoria sea exponencialmente mayor. Los datos del nuevo problema serán:

OBS: (0,1);(1,1);(2,4);(3,4)
SAT1: 1;1;1;1;1
SAT2: 1;1;1;1;1

En función de las heurísticas H_1 y H_2, hallamos las siguientes soluciones:

Heurística	Tiempo (ms)	Coste	Longitud	Nodos expandidos
Heuristica1	3486	29	29	12670
Heuristica2	3226	29	29	12662

```
1. SAT1: IDLE, SAT2: IDLE
2. SAT1: IDLE, SAT2: IDLE
3. SAT1: Gira, SAT2: IDLE
4. SAT1: Carga, SAT2: IDLE
5. SAT1: Observa O3, SAT2: Observa O4
6. SAT1: IDLE, SAT2: Carga
7. SAT1: IDLE, SAT2: Transmite
8. SAT1: IDLE, SAT2: IDLE
9. SAT1: Carga, SAT2: IDLE
10. SAT1: Transmite, SAT2: IDLE
11. SAT1: Carga, SAT2: Carga
12. SAT1: Gira, SAT2: Gira
13. SAT1: IDLE, SAT2: IDLE
14. SAT1: IDLE, SAT2: IDLE
15. SAT1: IDLE, SAT2: IDLE
16. SAT1: IDLE, SAT2: IDLE
17. SAT1: IDLE, SAT2: IDLE
18. SAT1: IDLE, SAT2: IDLE
19. SAT1: IDLE, SAT2: IDLE
20. SAT1: IDLE, SAT2: IDLE
21. SAT1: IDLE, SAT2: IDLE
22. SAT1: IDLE, SAT2: IDLE
23. SAT1: IDLE, SAT2: IDLE
24. SAT1: IDLE, SAT2: IDLE
25. SAT1: Carga, SAT2: Carga
26. SAT1: Observa O1, SAT2: Observa O2
27. SAT1: IDLE, SAT2: Carga
28. SAT1: Carga, SAT2: Transmite
29. SAT1: Transmite, SAT2: IDLE
```

Caso de Prueba 4 (problema4.prob): También podemos hacer modificaciones en el coste de las acciones. En la siguiente prueba, se expondrá un problema en el que los costes de transmisión y observación sean el doble que el coste de recarga de batería, necesitando recargar más de una vez para poder ejecutar las acciones (habrá observaciones que no puedan ser tomadas/transmitidas por falta de tiempo y por tanto tendrá que dar una vuelta completa hasta poder recargar y realizar las acciones restantes). Los datos que provienen del fichero de salida se basan en el enunciado del segundo caso de prueba y se muestran a continuación:

OBS: (0,3);(1,9);(2,4);(3,7)
SAT1: 2;2;1;1;2
SAT2: 2;2;1;1;2

```
1. SAT1: IDLE, SAT2: IDLE
2. SAT1: IDLE, SAT2: IDLE
3. SAT1: IDLE, SAT2: IDLE
4. SAT1: Observa O1, SAT2: IDLE
5. SAT1: IDLE, SAT2: Observa O3
6. SAT1: IDLE, SAT2: Carga
7. SAT1: Carga, SAT2: Carga
8. SAT1: Carga, SAT2: Observa O4
9. SAT1: IDLE, SAT2: IDLE
10. SAT1: Observa O2, SAT2: Carga
11. SAT1: Carga, SAT2: Carga
12. SAT1: Carga, SAT2: Transmite
13. SAT1: IDLE, SAT2: IDLE
14. SAT1: IDLE, SAT2: IDLE
15. SAT1: IDLE, SAT2: IDLE
16. SAT1: IDLE, SAT2: IDLE
17. SAT1: IDLE, SAT2: IDLE
18. SAT1: IDLE, SAT2: IDLE
19. SAT1: IDLE, SAT2: IDLE
20. SAT1: IDLE, SAT2: IDLE
21. SAT1: IDLE, SAT2: IDLE
22. SAT1: IDLE, SAT2: IDLE
23. SAT1: IDLE, SAT2: IDLE
24. SAT1: IDLE, SAT2: IDLE
25. SAT1: Transmite, SAT2: Carga
26. SAT1: Carga, SAT2: Carga
27. SAT1: Carga, SAT2: Transmite
28. SAT1: Transmite, SAT2: IDLE
```

Como vimos, en la solución óptima no se realiza ninguna acción de giro. Es por esto que mostraremos únicamente los resultados de la primera heurística, dado que los de la segunda serán equivalentes:

Heurística	Tiempo (ms)	Coste	Longitud	Nodos expandidos
Heuristica1 = Heuristica2	32449	28	28	27590

Caso de Prueba 5: Por último, si el coste de alguna acción fuera mayor al coste de la batería máxima de alguno de los satélites, el problema carecería de sentido dado que no podría ejecutar las acciones esenciales de un satélite para observar y transmitir las observaciones a la estación base.

3) Conclusiones acerca de la práctica

La primera parte de esta práctica nos ha permitido aprender sobre el modelado de problemas de satisfacción de restricciones (CSP), y ver cómo un cambio en los dominios de las variables así como en la cantidad de variables, produce que se generen más o menos soluciones al problema. Nos ha permitido también conocer python constraints y practicar Python.

Durante la segunda parte hemos aplicado nuestros conocimientos sobre la resolución de problemas mediante el uso de heurística, practicando a modelar problemas de este tipo. Además, hemos podido comprobar como, tal y como habíamos aprendido en la asignatura, el uso de una heurística mejor informada produce una mayor eficiencia (en nuestro caso, generando menos nodos la segunda que la primera heurística).

Consideramos que esta práctica nos ha ayudado mucho a interiorizar los conceptos aprendidos, tanto en las clases prácticas como teóricas durante la segunda mitad del curso. Creemos que es muy importante dar una visión real de la teoría dada, a través un caso de uso de la vida cotidiana. Nos ha parecido por tanto una práctica interesante y útil. Destacar que el número de horas invertidas ha sido mayor que en la primera práctica (unas treinta horas cada integrante de la pareja) y la complejidad mayor. Pese a todo, nos parece una buena práctica a mantener.