Universidad Carlos III
Grado de Ingeniería en Informática
HEURÍSTICA y Optimización
Práctica 2: Satisfacción de Restricciones y Búsqueda Heurística
Curso 2022-2023

Miguel Castuera García

NIA: 100451285

Beatriz Villacorta Llorente

NIA: 100451062

Grupo 84

Repositorio de Git: Practica 2

Heurística y optimización

Practica 2

Miguel Castuera y Beatriz Villacorta , 2022

1. INTRODUCCIÓN

En este documento se encuentra la resolución de la práctica 2. Se trata de asignar al alumnado asientos de autobuses teniendo en cuenta las preferencias y necesidades particulares de cada alumno.
Para ello

2. DESCRIPCIÓN MODELO PARTE 1

2.1 CARACTERÍSTICAS DEL PROBLEMA

Para la realización de este problema hay que tener en cuenta las diferentes necesidades del alumnado mostradas en el siguiente apartado.

2.2 MODELADO DEL PROBLEMA

Un problema de satisfacción de restricciones puede ser representado mediante una terna (X, D, C) donde X es un conjunto de n variables {xi...xn} X = {xi} n i=1, D es una tupla <Di,...,Dn>de dominios finitos donde se interpretan las variables X, donde Di es el dominio que contiene los posibles valores que pueden asignarse a la variable x. y C{c1, c2, ..., cp} es un conjunto finito de restricciones. Cada restricción está definida sobre un conjunto de K variables.

2.2.1 Datos

29	25	21	17	puerta	13	9	5	1	chofer
30	26	22	18		14	10	6	2	
PASILLO									
31	27	23	19		15	11	7	3	
32	28	24	20	puerta	16	12	8	4	puerta

Tabla esquema de la distribución de los espacios del autobús.

Los asientos marcados en azul están designados preferentemente a los alumnos con movilidad reducida.
Los espacios marcados en verde son los pasillos y en rojo las puertas.

2.2.2 Variables, Dominio y Restricciones

VARIABLES

Las variables representan el número de alumnos a los que hay que asignar un asiento, tenemos una variable por alumno.

DOMINIO

El dominio de valores de cada variable está compuesto por los posibles asientos que pueden ocupar los alumnos. Di = {1, 2, 3, 4, ..., 30, 31, 32}

Cada variable tendrá un dominio, es decir a un alumno asignaremos un dominio (asiento).

RESTRICCIONES

Cada restricción Rij representa todos los valores (tomados de los dominios de las variables xi y xj respectivamente) que son simultáneamente legales.
Para establecer las restricciones hay que tener en cuenta las necesidades del alumnado.

Todo el alumnado tiene que tener asignado un asiento y solo uno.
El dominio no puede ser vacío y dos alumnos no pueden tener el mismo asiento.
X_i != 0 ∀ i

Si hay alumnos con movilidad reducida tendrán que sentarse en asientos designados a tal fin quedando libre, el asiento contiguo (por ejemplo, si se le asigna el asiento número 1 debería quedar libre el 2, y si se le asigna el 2 debería quedar libre el 1) . No considerando contiguos asientos como el 2 y el 3 ya que están separados por el pasillo.

X_R= D={(17,18), (19,20), (13,14), (15,16), (1,2), (3,4)

Los alumnos de primer ciclo se deben sentar en la parte delantera del autobús (asientos del 1 al 16.)
$\forall X_i \in X_N$ tal que $X_i \in$ Ciclo 1 Xc,
$D_i = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$

Los alumnos de segundo ciclo en la parte posterior (asientos del 17 al 32)
$\forall X_i \in X_N$ talque $X_i \in$ Ciclo 2 Xc,
$D_i = \{17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32\}$

Si dos alumnos son hermanos deberán sentarse uno justo al lado del otro.
$\forall X_i \in X_N$ y $\forall X_j \in X_N$ tal que X_i y X_j son hermanos
$D_{i,j} = \{(1,2), (3,4), (5,6), (7,8), (9,10), (11,12), (13,14), (15,16), (17, 18), (19, 20), (21, 22), (23, 24), (25, 26), (27, 28), (29, 30), (31, 32)\}$

$X_{i,IDhermano} = X_{j,id}$ and $X_{j,IDhermano} = X_{i,ID}$
R Xi, Xj = {(1, 2), (3, 4), (5, 6), (7, 8), (9, 10), (11, 12), (13, 14), (15, 16)}

Si los dos hermanos fueran de ciclos distintos se haría una excepción a la regla anterior forzando a que los dos se sentaran en la zona destinada a alumnos de primer ciclo y ocupando el mayor la posición de "pasillo".

$\forall X_i \in X_N$ y $\forall X_j \in X_N$ tal que X_i y X_j son hermanos de distinto ciclo

Dominio del hermano mayor: {2,3,6,7,10,11,14,15}

Dominio del hermano menor: {1,4,5,8,9,12,13,16}

$\{R_{X_1} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}\}$

si Xic > Xjc

Xic = {14, 10, 6, 2, 3, 7, 11, 15}

Si los dos hermanos fueran "conflictivos" sí pueden sentarse juntos (de hecho deben hacerlo) pero no podrán tener a otro alumno "conflictivo" sentado cerca de ninguno de los dos.

$\forall X_i \in X_N$ y $\forall X_j \in X_N$ tal que X_i y X_j son hermanos y ambos conflictivos

$D_{i,j} = \{(1,2), (3,4), (5,6), (7,8), (9,10), (11,12), (13,14), (15,16), (17, 18), (19, 20), (21, 22), (23, 24), (25, 26), (27, 28), (29, 30), (31, 32)\}$

Si uno de los dos hermanos tuviera movilidad reducida no sería necesario que los dos hermanos se sentaran juntos, pero sí deben estar sentados en la misma sección (parte delantera o trasera del autobús según sea el caso).

$\forall X_i \in X_N$ y $\forall X_j \in X_N$

Hermanos y R →dominio {(1-16, 1-16) ó (16-32, 16-32)}

2.3 CODIFICACIÓN DEL PROBLEMA EN PYTHON

Para abrir los ficheros de casos de prueba hemos creado un input que añade los datos del archivo de entrada a una lista llamada **students**.

El archivo de entrada nos proporciona la información de los alumnos de tal manera que

id alumno	Ciclo 1 ó 2	Conflictivo C o no X	movilidad reducida R o no X	IDhermano 0 si no tiene
1	2	X	X	X

Inicializamos 3 listas vacías: **MR, A, students2.**

Creamos una lista vacía llamada **MR** en la que posteriormente añadiremos a los alumnos con movilidad reducida y otra **A** a la que se añadirá el resto de los alumnos.

A continuación se añade en otra lista llamada **students2** los que se encuentra en la lista **students** añadiendo espacios entre los alumnos.

Para dividir los alumnos en las listas MR y A, tenemos que comprobar si estos tienen movilidad reducida o no. Esta información la obtenemos en la posición 3 de la lista. Si en esa posición nos encontramos una *R* quiere decir que tiene movilidad reducida y añadimos al alumno en la lista MR, en caso contrario lo añadimos a A.

Después se establece el dominio de las variables. En este problema las variables son los alumnos. Tenemos tantas variables como alumnos contenga el fichero de entrada y el dominio son los asientos que el autobús tiene disponible para los alumnos. Pero no todas las variables tienen el mismo dominio ya que los alumnos con movilidad reducida solo se pueden sentar en los asientos establecidos [1, 2, 3, 4, 13, 14, 15, 16, 17, 18, 19, 20].

Se crean las variables y se asignan los dominios a esas variables usando problem.**addVariables**.

Una vez establecidas las variables y el dominio se añaden las restricciones del problema.

Con **AllDifferentConstraint** comprobamos que cada alumno tiene un asiento y solo 1 para que dos alumnos no puedan sentarse en el mismo asiento.

Luego comprobamos que los alumnos con movilidad reducida tienen libre el asiento contiguo con la función **movilidadReducida.**

Nos aseguramos de que los alumnos conflictivos están separados con la función **alumnoConflictivo.**

Las funciones **ciclo1** y **ciclo2** aseguran que los alumnos se sientan en las partes del autobús que les corresponde.

Para saber si los hermanos están bien sentados tenemos **colocación de hermanos** que comprueba que están juntos y **colocación hermanos C1** que hace que el hermano mayor se siente en el pasillo y en los asientos pertenecientes al ciclo 1.

Por último tenemos un bucle que va recorriendo la lista **students2**. A lo largo de la ejecución de este bucle se van añadiendo las restricciones del problema mediante la función addConstraint y se van añadiendo las funciones mencionadas anteriormente.

Para finalizar se añaden las soluciones en un archivo .output.

2.4 RESOLUCIÓN Y ANÁLISIS DE LOS CASOS DE PRUEBA

Para realizar las pruebas hemos creado 5 casos de prueba.

2.4.1 Caso de prueba 1

El primer fichero, alumnos.txt es el caso de ejemplo que se mostraba en el enunciado de la práctica, contiene a 8 alumnos. Entre los cuales solo hay una pareja de hermanos. El 3 y el 1. Pero el hermano 1 es conflictivo y el 3 no, por lo que el programa los coloca separados en el mismo ciclo. El resto de alumnos están repartidos a lo largo de los ciclos.

```
{'4CR': 20, '8XR': 18, '1CX': 8, '7CX': 30, '3XX': 7, '2XX': 25, '5XX': 15, '6XX': 16}
```

2.4.2 Caso de prueba 2

El segundo fichero, *alumnos2.txt*, tiene a 11 alumnos. 7 de ellos son hermanos.

El alumno 2 y el 5 son hermanos pero están en distinto ciclo. Como el alumno 2 es de movilidad reducida no es necesario que se sienten juntos pero deben ir en el mismo ciclo.

Los alumnos 3 y 4 son hermanos pero están en distinto ciclo. El programa los debe colocar en el ciclo 1 con el hermano menor(3) en la ventana y el 4 en el pasillo.

Los alumnos 6 y 8 son hermanos de distinto ciclo también y ambos son conflictivos, por lo que el programa los debe colocar juntos pese a que sean conflictivos.

Los alumnos 10 y 11 son hermanos del mismo ciclo pero 11 es de movilidad reducida.

El programa debe colocar a los alumnos en función de las restricciones

Esta es una de las soluciones que muestra el programa.

```
{'9CR': 20, '10XR': 2, '2XR': 3, '7XR': 18, '1CX': 32, '6CX': 11, '8CX': 12, '11XX': 9, '5XX': 10, '3XX': 13, '4XX': 14}
```

Coloca al **alumno 1** en el asiento 32, que corresponde al ciclo2. Al **alumno 2** le coloca en el asiento 3 y a su **hermano 5** en el 10, pese a que el alumno pertenece al ciclo 2, aunque su hermano menor es de movilidad reducida y no se puede sentar a su lado le sienta en el asiento 10 del primer ciclo. El **alumno 3 y el 4** utilizan asientos contiguos en el primer ciclo.

El **6 y el 8** se sientan en la parte delantera estando el 6 en el pasillo y sin alumnos alrededor. El **7** se sienta en el ciclo 2 sin alumnos en los asientos

contiguos debido a que tiene movilidad reducida. el **9** se sienta en el 20 sin alumnos alrededor debido a que posee movilidad reducida y es conflictivo. Por último el **10 y 11** son hermanos del mismo ciclo pero no se simental alado porque les ocurre lo mismo que la pareja de hermanos 2 y 5. Que uno de ellos tiene movilidad reducido.

2.4.3 Caso de prueba 3

Este fichero *alumnos3noSolucion.txt* tiene 8 alumnos, pero la mayoría son conflictivos y son hermanos entre sí de distintos ciclos. Por lo que todos deben ir en la parte delantera del autobús (1-16). El problema es que no hay suficientes asientos para todos los alumnos. Por lo que el programa devuelve la solución vacía.

El fichero *alumnos3.txt* está formado por 7 alumnos. Tiene dos parejas de hermanos, 2 y 3, hermanos conflictivos de distintos ciclos que los sentarán juntos en el ciclo1 y 5 y 6, hermanos con movilidad reducida de distinto ciclo.
Por otro lado el alumno 1, de primer ciclo sin movilidad reducida, tampoco es conflictivo, por lo que no tiene ninguna limitación a la hora de colocarse.
El alumno 4 es de movilidad reducida por lo que nadie podrá sentarse a su lado.

```
{'7CR': 20, '4XR': 18, '5XR': 14, '6CX': 12, '2CX': 6, '3CX': 5, '1XX': 2}
```

Observamos que la solución se muestra de manera correcta. Pese a que 5 y 6 son hermanos, los coloca en asientos separados porque tienen movilidad reducida

2.4.4 Caso de prueba 4

El fichero *alumnos4.txt* está formado por 10 alumnos. La peculiaridad de este caso es que 5 y 6 son hermanos de distinto ciclo, uno conflictivo y otro R, se sientan en el mismo ciclo pero no alado.

```
{'7CR': 20, '5XR': 2, '10CX': 8, '4CX': 22, '6CX': 6, '2CX': 14, '3CX': 13, '1XX': 21, '8XX': 9, '9XX': 10}
```

2.4.5 Caso de prueba 5

El último caso de prueba implementado es *alumnos5.txt*, este fichero tiene dos alumnos que no son hermanos. Ambos alumnos pertenecen al ciclo 1.
Gracias a este ejemplo comprobamos que nuestro programa asigna los asientos adecuados a los alumnos. No les asigna asientos fuera del dominio del ciclo 1, es decir les sienta del 1 al 16. A demás de pertenecer al mismo ciclo, el 1 es conflictivo y el 2 tiene movilidad reducida. Podemos observar que no se sientan juntos.

```
{'2XR': 16, '1CX': 2}
```

3. DESCRIPCIÓN MODELO PARTE 2

3.1 CARACTERÍSTICAS DEL PROBLEMA

El objetivo de este problema es encontrar la forma más óptima para que los alumnos suban al autobús. Antes de subirse forman una fila en la puerta. Hay que tener en cuenta que los alumnos conflictivos duplican el tiempo a aquellos alumnos que se sientan detrás y que los alumnos de movilidad reducida necesitan ayuda para subir y hacen que aquellos que los ayuden tarden el mismo tiempo que ellos en subir pero ese tiempo no aumentaría el total de la cola ya que suben a la vez.

3.2 MODELADO DEL PROBLEMA

Operadores:

Empleamos un operador para expandir los nodos, nodo_nuevo.

Tenemos otro operador que se encarga de incluir en la listaAbierta los nodos expandidos y otro para meter la closedList a los alumnos.

Otro operador que se encarga de meter los nombres(id de los alumnos) en el orden que deben subir al autobús en una lista.

Contador para los nodos generados.

Estados:

Un estado puede representarse como una lista a la que van entrando los alumnos que deben estar en el autobús.

Estado inicial: como estado inicial tenemos una lista vacía de alumnos a la que llamaremos closedList, ya que todavía no hay ningún alumno en el autobús.

Estado Final: el estado final se encuentra cuando todos los alumnos que estaban en la openList (lista con todos los alumnos que deben entrar en el bus) están en la closedList, es decir, cuando la lista cerrada está llena.

Precondiciones

Los alumnos con **movilidad reducida**

tardan tres veces mas en montar en el autobús que el resto de alumnos.

El alumno que se encuentre en la cola justo detrás de un alumno con movilidad reducida deberá ayudarle a subir al autobús.

No puede haber dos alumnos con movilidad reducida uno a continuación del otro.

No puede haber un alumno de movilidad reducida ocupando la última posición.

Cuando un alumno ayuda a subir a otro con movilidad reducida el tiempo

que tardan los dos es el tiempo que tarda el alumno con movilidad reducida puesto que suben a la vez.

Un alumno **conflictivo:**

Duplica el tiempo necesario para subir al autobús tanto del compañero que se encuentre justo delante de él como del compañero que se encuentre justo detrás.
Si un alumno conflictivo ayuda a un alumno con movilidad reducida a subir al autobús, el tiempo empleado por los dos se duplica.
Un alumno conflictivo duplicara el tiempo invertido para subir al autobús de todos aquellos alumnos que, encontrándose en la cola después de él, tengan asignados asientos en el autobús con una numeración superior a la suya.

Heurísticas:

Para la implementación de la Heurística se han planteado dos opciones.

- **Heurística 1:** es la más informada. Tiene en cuenta que los alumnos tardan un tiempo diferente en subirse al autobús dependiendo de su tipo, es decir, si tienen Movilidad Reducida (R), si son conflictivos(C), o si son “normales” (X).
  Para plantearla se tiene en cuenta que cada vez que se expande un nodo se reduce el coste heurístico respecto al nodo padre. Si su padre(el nodo que ya ha sido expandido) es de tipo conflictivo, el coste heurístico del nodo actual será 3. Si el nodo padre es de tipo movilidad reducida el coste será 2 ya que al ayudar a su padre a subir tardan el mismo tiempo los dos (tiempo del anterior, el de movilidad reducida + tiempo del nodo actual). Si el nodo padre era de tipo X, el coste actual será 1. Por último si su padre era tanto conflictivo como movilidad reducida su coste heurístico actual será de 5.

- **Heurística 2:** es poco informada dado que considera que todos los alumnos tardan el mismo tiempo en en subir al autobús con un coste = 1.
  Cada vez que se expande un nodo el coste heurístico se reduce en uno respecto al de su nodo padre. Es decir, calcula el coste/número de alumnos que faltan por entrar en el autobús.

3.3 CODIFICACIÓN DEL PROBLEMA EN PYTHON

Para realizar el problema en python. Se ha decidido convertir los datos del fichero de entrada en una lista de tuplas. De tal forma que cada tupla está formada por el alumno, si tiene o no conflictivo, si tiene movilidad reducida o no, y por último el asiento que ocupa. Tendremos una tupla por alumno que esté en el fichero de entrada (id_alumno, X,X,asiento).

El programa está formado por:

- La clase nodo se utiliza para almacenar los datos de cada alumnos que se expande, datos como el nombre, el nodo padre(alumnos que va delante en la cola),coste, coste de heurística, coste individual del alumno y sus tipos(movilidad reducida y conflictivos).
- Una función llamada def heurística, que recibe el nodo del que se quiere obtener la heurística y el número de la heurística que se quiere aplicar, en este caso solo aceptará recibir 1 o 2, si no salta un error y para el programa.
- Una función llamada Astar que recibe la lista de tuplas y el número de la heurística que se desea implementar. Esta función es una función de búsqueda basada en el algoritmo A* que encuentra la mejor disposición de los alumnos en la fila del autobús. Parte de un nodo Start, que se crea llamando a la clase nodo. A continuación va expandiendo los nodos de la openList y metiéndolos en la closedList. También cuenta el número total de nodos expandidos. Una vez recorridos los nodos, devuelve la solución en un archivo.output y en otro muestra las estadísticas.

3.4 CASOS DE PRUEBA Y ANÁLISIS COMPARATIVO DE LAS HEURÍSTICAS

Para probar nuestro programa se han usado tres archivos de entrada diferentes.

Tras ejecutar todos los archivos, se observa que aquellos ejecutados con la primera heurística implementada han expandido más nodos que sus equivalentes empleando la segunda Heurística(la menos informada). Como consecuencia el tiempo de ejecución del programa en los casos que se emplea la heurística 2 es menor que los que se emplea la 1.

Caso de prueba 2 alumnos2.prob

HEURÍSTICA 1

```
INICIAL:{'3XX': 11, '1CX': 12, '6XX': 15, '5XX': 16, '4CR': 20, '2XX': 31, '7CX': 32}
FINAL:{'4CR': 20', '3XX': 11, '1CX': 12, '6XX': 15, '5XX': 16, '2XX': 31, '7CX': 32}
```

HEURÍSTICA 2

```
INICIAL:{'3XX': 11, '1CX': 12, '6XX': 15, '5XX': 16, '4CR': 20, '2XX': 31, '7CX': 32}
FINAL:{3XX': 11, '6XX': 15, '5XX': 16, '2XX': 31, '1CX': 12, '4CR': 20, '7CX': 32}
```

Observamos que no coloca al alumno con movilidad reducida el último ya que necesita ayuda para subir al autobús.

5. CONCLUSIONES

El trabajo nos ha ayudado principalmente para aprender a resolver problemas complejos haciendo uso de la heurística y a familiarizarnos con los entornos en que se utiliza. Mientras lo hacíamos hemos perfeccionado el uso de la búsqueda heurística informada que nos parecía muy compleja en un principio y las formas de optimizar el proceso de búsqueda.

Además, ahora tenemos claro cómo aplicar distintas restricciones y dominios a las variables de un problema gracias a la parte de csp. Esta primera parte también nos ha servido para volver a familiarizarnos con Python y a aumentar nuestros conocimientos sobre este lenguaje de programación en lo que respecta a la aplicación de la heurística con el uso de la librería python-constraint.