Universidad Carlos III

Heurística y Optimización

Curso 2023-24

Práctica 2

Alumnos:

Carlos Perez Gomez/ Jorge Viñas Blasco

NIAs:

100451025/100472262

Grupo:

84

Correo de todos los alumnos:

100451025@alumnos.uc3m.es

100472262@alumnos.uc3m.es

Índice:

1.Introducción

Nuestra práctica consta de dos partes en las que hemos utilizado los conocimientos adquiridos en clase sobre CSP y Búsqueda Heurística.

En la primera parte modelamos el problema como un CSP, identificando las variables de decisión, sus dominios, así cómo generando sus restricciones que posteriormente serán limitadas aplicando camino-consistencia. Para gestionar todo esto utilizaremos la librería “constraint” de python que nos ayudará a añadir las diferentes limitaciones que debe cumplir nuestro programa.

En la segunda parte trataremos un problema sobre búsqueda en el que tendremos que encontrar el camino más corto dentro de un mapa con estados finales múltiples. Para lograr encontrar de manera estos caminos hemos hemos desarrollado un algoritmo A* al que iremos guiando según su estado actual y el estado final objetivo que tenga en cada momento, ya que al no tener un sólo estado final nuestro algoritmo sino varios deberemos estar calculando constantemente de los estados finales posibles y sus respectivas rutas optimizadas cuál de esas rutas priorizamos.

En base al código realizado y los resultados obtenidos hemos realizado el siguiente trabajo.

2.Descripción de los modelos

En la 1ª parte se pedirá la organización de un parking de ambulancias y en la 2ª parte la gestión de la ruta de una de estas ambulancias, a continuación desarrollaremos ampliamente cada una de las partes.

2.1. Modelo Parte1

Para poder empezar a modelar primero tenemos que definir las variables que tendrá nuestro mapa, las cuales vendrán determinadas por el archivo .txt que se le introduzca al programa. En el que se definirá: las dimensiones del parking (filas y columnas), sus plazas electrificadas y por último las ambulancias que debe contener.Guardaremos cada uno de estos datos en una variable y llamaremos a la función que resuelve el programa. Cómo breve resumen en términos de CSP, las ambulancias serán nuestras variables y a priori todos las plazas del parking serán su dominio. A continuación desarrollaremos las restricciones aplicadas para poder hallar las diferentes distribuciones solución.
Toda la restricciones irán dentro de un bucle para que se compruebe cada ambulancia con todas las otras ambulancias del parking.
Para empezar asignaremos a cada ambulancia que su dominio son todas las plazas del parking.

1ªConstraint->Ambulancias con congelador sólo pueden estar en plazas con conexión

> Esta restricción se cumplirá mediante un filtrado en el que los vehículos con congelador simplemente serán delimitados a que su dominio sean sólo aquellas plazas con conexión a corriente.

2ªConstraint->Ambulancias prioritarias deberán estar por delante que las que no lo sean

Esta restricción será analizada mediante un bucle anidado que revise que la ambulancia del primer bucle en el caso de ser prioritario tenga una plaza más cercana a la salida que cualquier otra ambulancia que no sea prioritaria. El caso contrario será analizado en una posterior iteración del bucle.

3ªConstraint->Las ambulancias deben tener un hueco a la izquierda o a la derecha

Esta restricción será analizada dentro de un 3er bucle anidado que tendrá en cuenta un 3er vehículo y llamará a una función externa que divide los casos posibles en 3:

El vehículo 1 está en la primera fila del parking, el vehículo 1 está en la última fila del parking y por último en cualquier otro caso.

En el caso de que esté en la primera fila la función devolverá “False” en el caso de que el vehículo 2 o vehículo 3 estén situados a su derecha, puesto que a la izquierda tendrá la pared.
El caso de que el vehículo 1 esté en la última fila será similar al anterior sólo que comprobando que los vehículos 2 y 3 no estén situados a la izquierda del vehículo 1, de nuevo no hará falta comprobar la derecha puesto que se encontrará la pared.
Por último si el vehículo 1 se encuentra en cualquier punto intermedio se valorará si el vehículo 2 está a la derecha y el vehículo 3 a la izquierda del vehículo 1 se devuelve False. No será necesario añadir otra restricción if, pues sería redundante, ya que en próximas iteraciones del bucle se comprobará con los valores de vehículo 2 y vehículo 3 invertidos.
En cualquier otro caso se devolverá “True” lo que querrá decir que no estamos en ninguno de los casos anteriores.

4ªConstraint->Cada ambulancia deberá ocupar una ubicación diferentºe

Cómo es lógico 2 ambulancias no podrán estar en la misma plaza de garaje. Añadiremos esta restricción fuera de los bucles.

Soluciones:

Serán guardadas en archivos .csv. Los resultados obtenidos serán explicados en el punto “Análisis de resultados”

2.2 Modelo Parte 2

En la 2ª parte llevaremos a cabo la implementación de un programa en el que se pide hallar el camino mínimo para recoger a varios pacientes, llevarlos a sus respectivos centros sanitarios y volver a nuestro parking mediante A* minimizando la gasolina gastada en el trayecto.

Nuestro programa recibe 2 argumentos, el primero es el mapa y el segundo la heurística que debemos utilizar.
El mapa será procesado para guardar todos sus datos útiles cómo su tamaño, localizaciones de pacientes, gasolina restante, posición actual entre otras.

La heurística será posteriormente utilizada para guiar a nuestro A*.

Nuestras Heurísticas:

En el caso de que reciba un 1 utiliza la “heurística_pacientes_sin_trasladar”, esta heurística tendrá en cuenta el nº de pacientes que quedan por ser trasladados, multiplicando este nº por 0,5 y guiando de esta manera el recorrido.

En el caso de que reciba 2 utilizará una heurística basada en la “Minimum Spanning Tree (MST) ponderado k-árboles ”, en el caso 3 será el máximo de la 2 y la 1, en cualquier otro caso el problema se realizará mediante fuerza bruta.
Para gestionar el programa dispondremos de una clase “Estado” que contendrá los datos básicos de cada iteración del problema, cómo son la ubicación de la ambulancia, los pacientes que lleva la ambulancia, tanto su tipo cómo su casilla de origen. Y por último las ubicaciones de los pacientes que faltan por ser trasladados.
A parte en esta clase definiremos algunos métodos tales como __lt__, __hash__, __eq__ y __str__ que nos ayudarán al desarrollo del programa.

Pasando a la ejecución de nuestro programa, en el main llamaremos a la función “a_estrella” a la que le pasaremos el mapa, el estado inicial, la dirección del archivo de salida y la heurística que debe utilizar.
Este A* entrará en un bucle en el cual revisará si ha llegado al estado objetivo;
Definido cómo que la lista de pacientes está vacía, la ambulancia está en el parking y tiene el depósito de gasolina lleno.
En el caso de que no se haya alcanzado el estado_objetivo se pasará a expandir posibles sucesores, para cada nodo habrá un máximo de 4 sucesores, que serán aquellos pertenecientes a las casillas adyacentes del primero. Esta función trabajará de la siguiente manera:

Generar sucesores:

Tendremos un bucle de 4 iteraciones (1 por cada posible movimiento), asignaremos como nueva posición este movimiento y revisaremos que nuestra ambulancia tenga gasolina, que nuestra casilla pertenezca al mapa y que esta nueva posición no sea una ‘X’ las cuales serán casillas intransitables. Si algo de esto no se cumple la función devolverá una lista vacía a la función A*.
De no ser así se comprobará de qué tipo es la casilla a la que se quiere transitar.
Si fuera tipo ‘N’ o ‘C’ y fuera un paciente aún por recoger se comprobaría que se cumplen con las restricciones establecidas por el programa: máximo 10 pacientes en la ambulancia, máximo 2 de tipo ‘C’, en el caso de no haber ningún tipo ‘C’ se pueden llenar sus plazas con tipo ‘N’, pero si hubiese 9 tipo ‘N’ no puedo meter ningún tipo ‘C’ y si hay tipos ‘C’ dentro no puedo recoger ningún otro tipo ‘N’. Si se cumplen las restricciones se eliminará de la lista de pacientes sin trasladar a este elemento y se añadirá a la ambulancia.
En el caso de que la casilla fuera tipo ‘CC’ y la ambulancia contuviese algún paciente tipo ‘C’ se descargara este paciente liberando dicho hueco en la ambulancia.

Por último si la casilla fuera tipo ‘CN’ y la ambulancia no contuviese pacientes tipo ‘C’ se descargaron todos los pacientes tipo ‘N’ de la ambulancia.
A continuación pasaremos a llamar a la función “costo_entre_estados”. Se actualizarán todos los atributos del nuevo estado y añadiría a la lista de posibles sucesores.
Tras ejecutar esto una vez por cada posible movimiento se devolverá la lista de sucesores al A* para que seleccione a uno de ellos y vuelva a repetir el procedimiento.

Costo entre estados:

Esta función devolverá 1 en el caso de que la casilla contenga cualquiera de las letras representativas de nuestro problema, a excepción de la X, que cómo sabemos nunca será tenida en cuenta. Y devolverá el valor entero de la casilla en cualquier otro caso.
Esta función será utilizada en la anterior mencionada “generar_sucesores” para calcular la gasolina restante que tendrá el nuevo estado generado.

Impresion valores:

El archivo .output lo generamos de golpe al final del programa cuándo hayamos llegado al estado final, pues tenemos guardado el camino, con él llamaremos a la función imprimir_output accederemos a los valores en el mapa para poder imprimir el tipo de la casilla así cómo ir haciendo un seguimiento del uso de gasolina.
El archivo .stat lo generamos pasándole el número de nodos generados, que hemos contado en el A*, el tiempo que ha tardado en ejecutar y las direcciones de los archivos de salida, con todo ello procederemos a la impresión cómo se especifica en el enunciado.

3.Análisis de los resultados

Parte1:

Caso de prueba (parking01.txt); modelo del pdf:
Nos da 2175288 soluciones posibles, cómo hemos contrastado con varios compañeros se trata del nº correcto.
Caso de prueba (parking02.txt); ambulancias con congelador> plazas electrificadas:
Cómo es previsible el resultados de nº de soluciones será 0
Caso de prueba (parking03.txt); nºambulancias > espacio parking:
De nuevo nuestro resultado serán 0 soluciones.

Caso de prueba (parking04.txt); sin ambulancias con congelador ni plazas electrificadas:
Se solucionará correctamente el problema dando un total de 4 soluciones.

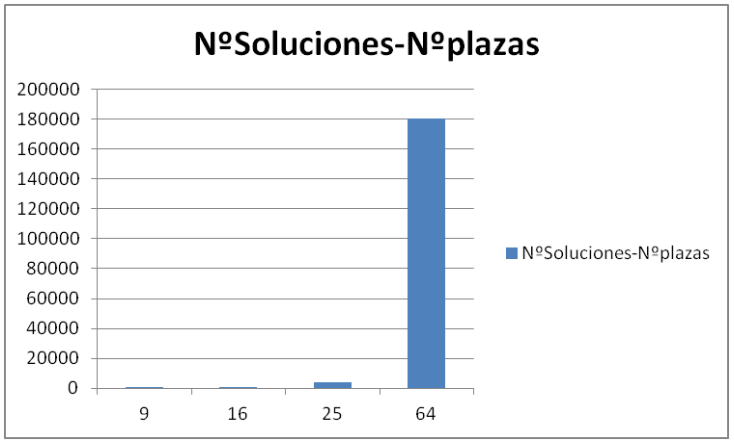
Caso de prueba (parking05.txt); todas las plazas son electrificadas y todas las ambulancias son sin congelador:

Asignará correctamente las plazas a las ambulancias sin importar que nuestras ambulancias no tengan congelador dando un total de 8 soluciones.

Caso de prueba (parking06.txt); no hay plazas electrificadas, pero si ambulancia con congelador:
El resultado será que hay 0 posibles soluciones.

Por último hemos querido realizar un análisis de la evolución según la amplitud del parking, con los casos parking07.txt->parking10.txt, evaluando a un mismo nº de ambulancias cuánto varía la cantidad de soluciones desde el 3*3 al 8*8.
Mostrando un notable crecimiento exponencial.

NºCasillas	9	16	25	64
NºSoluciones	10	512	3970	180672



Parte2:

mapa-1.csv, mapa-3.csv, mapa-4.csv, mapa5.csv:

```
1;N;1;C;1
N;N;N;2;2
CC;C;P;1;2
N;1;X;X;X
3;1;1;CN;1
```

Caso de prueba mapa-5.csv con fuerza bruta:
Cómo podemos observar en esta ejecución se expandirá gran cantidad de nodos

```
Tiempo total: 21.443876266479492
Coste total: 22
Longitud del plan: 23
Nodos expandidos: 364349
```

```
(3,3):P:50
(2,3):N:49
(2,2):N:48
(2,1):N:47
(1,1):1:46
(1,2):N:45
(1,3):1:44
(1,4):C:43
(1,3):1:42
(1,2):N:41
(2,2):N:40
(3,2):C:39
(3,1):CC:38
(4,1):N:37
(4,2):1:36
(5,2):1:35
(5,3):1:34
(5,4):CN:33
(5,3):1:32
(5,2):1:31
(4,2):1:30
(3,2):C:29
(3,3):P:50
```

Caso de prueba mapa-1.csv con heurística 1:
Utilizaremos la heurística “heuristica_pacientes_sin_trasladar”, heurística informada que prioriza aquellos nodos en los que quedan menos pacientes por recoger. En este caso se expanden muchos menos nodos y encuentra mucho antes la solución

```
Tiempo total: 3.1553244590759277
Coste total: 22
Longitud del plan: 23
Nodos expandidos: 80822
```

```
(3,3):P:50
(2,3):N:49
(2,2):N:48
(2,1):N:47
(1,1):1:46
(1,2):N:45
(1,3):1:44
(1,4):C:43
```

```
(1,3):1:42
(2,3):N:41
(3,3):P:50
(3,2):C:49
(3,1):CC:48
(4,1):N:47
(4,2):1:46
(5,2):1:45
(5,3):1:44
(5,4):CN:43
(5,3):1:42
(5,2):1:41
(4,2):1:40
(3,2):C:39
(3,3):P:50
```

Caso de prueba mapa-3.csv con heurística 2:
“heuristica_mst_k” basada en el Minimum Spanning Tree (MST) ponderado k-árboles (k-Trees) como una heurística para estimar el costo mínimo necesario para recoger a todos los pacientes restantes y dejarlos en sus centros de atención.
El MST garantiza que todos los nodos estén conectados de manera mínima, lo que significa que la heurística proporcionará una estimación eficiente del costo mínimo necesario.
Se puede ajustar el valor de k para controlar cuántos MST se calculan, lo que permite adaptar la heurística según sea necesario.
El número de nodos expandidos es menor aún que en la h1, aunque el tiempo es un poco mayor.

```
Tiempo total: 4.850266695022583
Coste total: 22
Longitud del plan: 23
Nodos expandidos: 71892
```

```
(3,3):P:50
(2,3):N:49
(2,2):N:48
(2,1):N:47
(1,1):1:46
(1,2):N:45
(1,3):1:44
(1,4):C:43
(1,3):1:42
(2,3):N:41
(3,3):P:50
(3,2):C:49
(3,1):CC:48
(4,1):N:47
(4,2):1:46
(5,2):1:45
(5,3):1:44
(5,4):CN:43
(5,3):1:42
(5,2):1:41
(4,2):1:40
(3,2):C:39
(3,3):P:50
```

Caso de prueba mapa4.csv con heurística 3:
“max(heuristica_mst_k(sucesor,mapa), heuristica_pacientes_sin_trasladar(sucesor))” que realiza un max de las dos heurísticas anteriores escogieron la heurística más relevante en cada caso.
En este caso el tiempo y los nodos expandidos son los mismos que en el h2 probablemente porque esté informando más .

```
Tiempo total: 5.151045799255371
Coste total: 22
Longitud del plan: 23
Nodos expandidos: 71892
```

```
(3,3):P:50
(2,3):N:49
(2,2):N:48
(2,1):N:47
(1,1):1:46
(1,2):N:45
(1,3):1:44
(1,4):C:43
(1,3):1:42
(2,3):N:41
(3,3):P:50
(3,2):C:49
(3,1):CC:48
(4,1):N:47
(4,2):1:46
(5,2):1:45
(5,3):1:44
(5,4):CN:43
(5,3):1:42
(5,2):1:41
(4,2):1:40
(3,2):C:39
(3,3):P:50
```

mapa2.csv

```
1;N;1;C;1
N;N;X;2;2
CC;X;P;X;2
N;1;X;X;X
3;1;1;CN;1
```

Con cualquiera de las heurísticas este será el resultado, pues este mapa no tendrá solución al estar rodeado el parking de X.

```
Tiempo total: 0.0
Nodos expandidos: 1
```

4.Conclusiones de la práctica

En cuánto a la primera parte nos ha parecido muy intuitiva y relativamente fácil de implementar salvo la última restricción que dio algunos problemas. Nuestros problemas han venido en la 2ª parte en la que ante la frustración de que no nos corriese el mapa de ejemplo decidimos cambiar totalmente la implementación de manera errónea buscando algo que si funcionase. Conseguimos hacer un programa que resolvía de manera muy interesante y rápida el problema propuesto pero no era un a_estrella completo.No fue hasta hace pocos días cuando nos dimos cuenta que no éramos los únicos que no podíamos correr ese mapa 10x10 y que debíamos seguir insistiendo en esa dirección, por ello abandonamos nuestro modelo para volver al inicial, el cual tras muchas horas de trabajo conseguimos completar. La verdad es que nos gustaría que en ciertas cosas la práctica fuera más específica y que al poner un mapa de ejemplo se pusiera uno que no se resuelva con mucha dificultad. Aún con todo hemos aprendido muchas cosas sobre la heurística atribuyendo esta asignatura a problemas reales y comunes.


UNIVERSIDAD • CARLOS III • DE MADRID
homo
homini
sacra
res