Modelos de Investigación Operativa, Ingeniería Informática
Universidad de Valladolid

Práctica 6

Daniel González Alonso

1 de abril de 2017

Resumen

En este documento se describen los problemas y los resultados obtenidos de la práctica 6 del tema 3 de la asignatura Modelos de Investigación Operativa de Ingeniería Informática, Universidad de Valladolid.

1. Introducción

Esta práctica trataba de problemas de Set Covering. El modelo empleado para resolver los problemas de Set Covering es el siguiente:

$$\begin{aligned} &\text{Minimizar} && \sum_{j=1}^{n} f_j \cdot x_j && && (1)\\ &\text{Sujeto a} && \sum_{j \in N_i} a_{i,j} \cdot x_j \geq 1 && i = 1, \ldots, m \\ & && x_j \in \{0,1\} && j = 1, \ldots, n \end{aligned}$$

Donde m es el número de puntos de demanda, n es el número de posibles puntos de servicio, f_j representa el coste de instalar un punto de servicio en j y la matriz $a_{i,j}$ contiene solo los valores 1 si i está cubierto por j o 0 en caso contrario.

2. Ejercicios

2.1. Práctica 6.1

Para esta práctica se nos pide resolver el problema de los condados de Nueva York (conjunto de datos `data/matriz30x30.dat`) con el método greedy para valores de la distancia de cubrimiento entre 70 y 120, y comparar los resultados con los obtenidos de forma óptima con Xpress.

Este problema se encuentra resuelto mediante *Xpress Mosel* en el archivo `set_covering_6_1.mos`. Lo primero que hicimos en el archivo fue cargar los datos del fichero de datos, en la primera linea del archivo venía dado el valor de m, en la segunda el valor de n y en los siguientes la matriz de distancias $d_{i,j}$.

A continuación se creó un bucle que iteraba entre los valores de dc pedidos. En cada iteración lo primero que se hace es calcular la matriz $a_{i,j}$ mediante la distancia de cubrimiento dc actual y la matriz de distancias $d_{i,j}$. Lo siguiente es calcular la solución exacta mediante *Xpress* y el modelo presentado anteriormente, finalmente en cada iteración también se calcula la solución mediante el método greedy, el cual se describe a continuación:

1. Calcular el número de puntos de demanda no cubiertos que cubre cada posible punto de servicio que todavía no se encuentra en la solución. En mi caso lo almacené en un vector llamado d_j, y solo lo calculaba para todos aquellos posibles puntos de servicio que no se encontraran marcados en otro vector llamado `fijadas`. Además tengo otro vector llamado `cubiertas` para saber que punto de demanda ya está cubierto y quien no.

2. Buscamos el elemento que minimiza el criterio greedy $\frac{c_j}{d_j}$. Como para este problema los valores de coste c_j no se nos da, suponemos un valor 1 (minimizamos el número d instalaciones). A este elemento le llame en el código como `j_min`.

3. Añadimos `j_min` a la solución. Además en mi caso la añado al vector con elementos fijados.

4. Marcamos los puntos de demanda cubiertos por `j_min` como ya cubiertos.

5. En caso de que el número de puntos de servicio instalados se menor que m volvemos al paso 1, si no ya hemos alcanzado la solución Greedy.

La evolución de los resultados obtenidos con los distintos valores de dc se pueden observar en la siguiente gráfica, la cual compara los resultados de la solución exacta y la del algoritmo greedy:

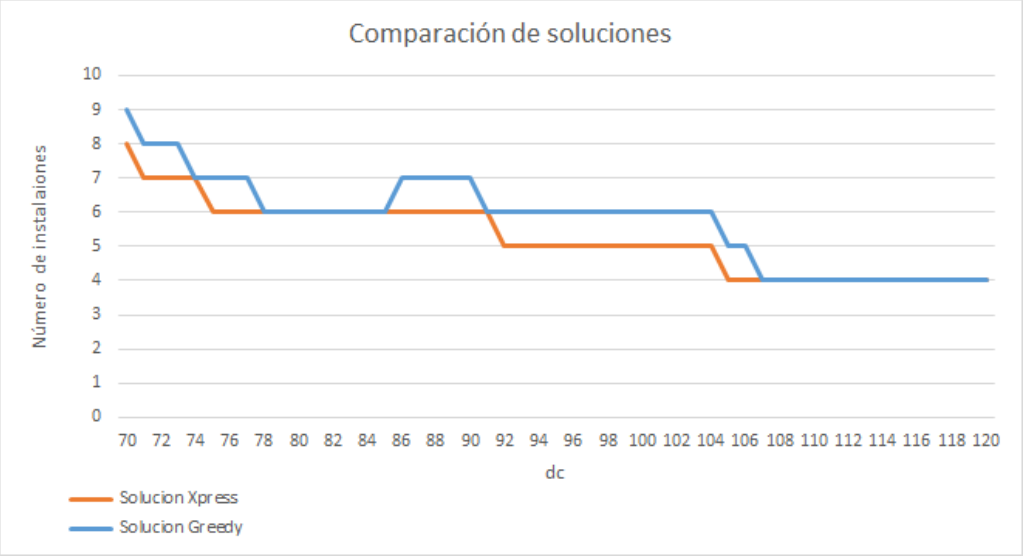

Figura 1: Comparación de los resultados de la solución de Xpress y el Método Greedy para la práctica 6.1

2.2. Práctica 6.2

Para la práctica 6.2 se nos pide resolver los problemas `aint1.dat` y `aint5.dat` para dc desde 250 hasta 400, con los métodos greedy, greedy aleatorizado (para $K = 5$ y $N = 100$), y comparar con el método exacto.

Esta práctica se encuentra resuelta mediante *Xpress Mosel* en los archivos `set_covering_6_2_aint1.mos` y `set_covering_6_2_aint5.mos`. En ambos ficheros los primero que se hizo fue cargar los datos. Los datos de los ficheros *aint* vienen dados de la siguiente forma:

- m — Número de puntos de demanda
- n — Número de puntos de servicio
- $h_i \quad i = 1, \ldots, m$ — Demandas, una por línea
- $d_{i,j} \quad i = 1, \ldots, m \quad j = 1, \ldots, n$ — Matriz $m \times n$ de distancias (Hectómetros)

A continuación se creó un bucle que iteraba entre los valores de dc pedidos. En cada iteración se obtiene el resultado para el método exacto, el método greedy y el método greedy aleatorizado. Como el método exacto y el greedy ya se ha explicado (la única diferencia en este problema tiene que ver con que para este problema si disponemos de los costes c_j), procederé a explicar el Método Greedy Aleatorizado.

En el método Greedy Aleatorizado a diferencia del Greedy normal disponemos de una lista *RCL* donde introduciremos los K indices (5 en nuestro caso) con los mejores valores para el criterio Greedy en la iteración actual, y de entre ellos se selecciona uno aleatoriamente para así obtener el elemento `j_min`. En mi caso para hacer esta parte cree un vector para la lista *RCL* de tamaño K y otro llamado `marcadas` que sirve para indicar si un elemento ya se encuentra en la lista *RCL* o no el cual reiniciaba en cada iteración. Este cálculo se hace de forma idéntica a la del algoritmo Greedy con la única diferencia de que

hay que tener en cuenta si un elemento ya se encuentra en la lista o no. Por otro lado, otra de las diferencias de este método es que estos pasos anteriores tienen que ejecutarse N veces (en nuestro caso 100), y en cada una de las iteraciones se comprueba si la solución obtenida es mejor que la de las iteraciones anteriores.

Las soluciones obtenidas para los ficheros de datos `aint1.dat` y `aint5.dat` se muestran a continuación:

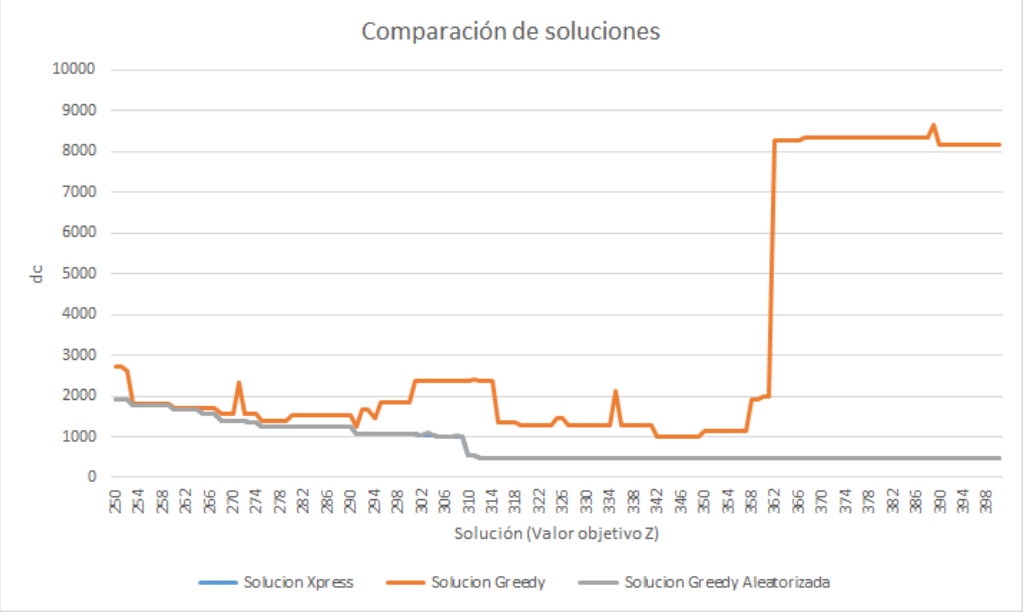

Figura 2: Comparación de los resultados para el archivo de datos `aint1`

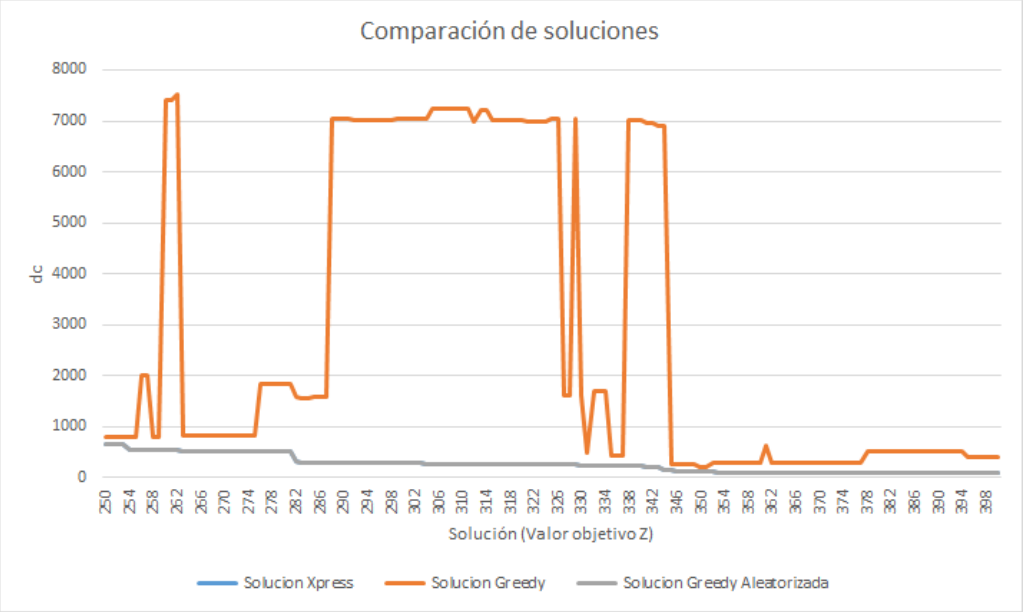

Figura 3: Comparación de los resultados para el archivo de datos `aint5`