Modelos de Investigación Operativa, Ingeniería Informática
Universidad de Valladolid

Práctica 7

Daniel González Alonso

27 de marzo de 2017

Resumen

En este documento se describen los problemas y los resultados obtenidos de la práctica 7 del tema 3 de la asignatura Modelos de Investigación Operativa de Ingeniería Informática, Universidad de Valladolid.

1. Introducción

Esta práctica trata de problemas de cubrimiento máximo. El modelo empleado para resolver estos problemas es el siguiente:

$$
\begin{array}{lll}
\text{Maximizar} & \displaystyle\sum_{i=1}^{m} h_i \cdot z_i & \\
\text{Sujeto a} & \displaystyle\sum_{j \in N_i} x_j \geq z_i & i = 1, \ldots, m \\
 & \displaystyle\sum_{j=1}^{n} x_j \leq P & \\
 & x_j \in \{0,1\} & j = 1, \ldots, n \\
 & z_i \in \{0,1\} & i = 1, \ldots, m
\end{array}
\tag{1}
$$

Donde x_j representa si se abre una instalación en j (1) o no (0), z_i nos dice si la demanda de i queda cubierta (1) o no (0), h_i es la demanda de i y P es el número de instalaciones que se van a abrir. El objetivo del modelo como se puede observar es maximizar la demanda que queda cubierta para un cierto número de instalaciones.

2. Desarrollo

Para esta práctica se nos pide resolver el problema de cubrimiento máximo para los problemas `data/aint1.dat` y `data/aint5.dat` mediante el método exacto, el método greedy y el método greedy aleatorizado ($K = 5$ y $N = 100$) para valores de p entre 1 y 6 y una distancia de cubrimiento $dc = 200$ (equivalente a 20 Km). Al final se nos pide que para cada método construyamos las tablas o gráficas con la demanda cubierta y el porcentaje de demanda cubierta para cada valor de p.

Esta práctica se encuentra resuelta mediante *Xpress Mossel* en los archivos `max_covering_aint1.mos` y `max_covering_aint5.mos` (el nombre indica el fichero de datos empleado). En cada uno de los ficheros se resuelve el problema con los métodos pedidos.

Para resolver esta práctica en cada uno de los ficheros lo primero que se hizo fue cargar los datos. El formato de los datos *aint* es el siguiente:

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

Una vez cargados los datos, la siguiente parte consiste en obtener la resolución exacta. La solución exacta obtenida con Xpress no tiene mayor dificultad que introducir el modelo anterior, lo único destacable de esta parte fue el tiempo. La solución de Xpress nos dio 0.516 segundos para los datos `aint1` y 0.838 segundos para `aint5` después de ejecutarlo con todos los valores de P.

En la siguiente parte se hizo la solución greedy. El algoritmo se describe a continuación:

1. Buscamos el indice del nodo que cubre la mayor cantidad de demanda y que no esté fijado todavía. Como detalles de implementación cabe destacar que en mi caso, para este paso simplemente calculaba la suma de toda la demanda que cubre un punto (los puntos que cubre cada nodo de servicio están precalculados en una matriz llamada `cubrimientos`), y con ello buscaba el máximo, cuyo indice es llamado `j_max`. Además en caso de que en un nodo de servicio la suma de demanda que puede cubrir diese 0, lo fijo y añado a la solución con un 0.
2. Fijamos el nodo anterior y los añadimos a la solución.
3. Eliminamos la demanda ya cubierta. En mi caso este paso lo hacía simplemente poniendo un 0 en el vector de demandas (llamado *demandas*) en todos aquellos nodos de de demanda que se encuentren cubiertos por `j_max`.
4. Si todavía no hemos cubierto toda la demanda y el número de instalaciones es inferior a p volvemos al paso 1.

La solución greedy obtenida es en este caso la más rápida, con 0.031 segundos para los datos `aint1` y también de 0.031 segundos para `aint5` después de ejecutarlo con todos los valores de P.

Por último se hizo la versión greedy aleatorizada, la cual se diferencia del algoritmo greedy "normal" en que para el primer paso se dispone de una lista RCL con los K indices (5 en nuestro caso) de los mejores valores en la iteración actual, y de entre ellos se selecciona uno aleatoriamente para así obtener j_max. En mi caso para hacer esta parte cree un vector para la lista RCL de tamaño K y otro con con banderas para indicar si un elemento ya se encontraba en la lista RCL o no el cual reiniciaba en cada iteración. El calculo de cada elemento de la lista RCL se hace de forma idéntica al algoritmo greedy sin RCL, solo que hay que tener en cuenta si un elemento ya está marcado (en la RCL) o no. Además de esta lista RCL hay que destacar en el algoritmo greedy aleatorizado el bucle anterior ejecutarse N veces (en nuestro caso 100), y en cada una de las iteraciones se comprueba si la solución es obtenida es mejor que la de las iteraciones anteriores.

Sobre el tiempo de ejecución, el algoritmo greedy aleatorizado es sin duda alguna el más lento, tardando 14.529 segundos para `aint1` y 12.519 segundos para `aint5`.

3. Resultados

- Resultados obtenidos para `aint1`:

Valor de P	Método Exacto		Algoritmo Greedy		Algoritmo Greedy Aleatorizado	
	Demanda Cubierta	Porcentaje de la demanda Cubierta	Demanda Cubierta	Porcentaje de la demanda Cubierta	Demanda Cubierta	Porcentaje de la demanda Cubierta
1	15397	83.1506 %	15397	83.1506 %	15397	83.1506 %
2	17055	92.1046 %	16746	90.4358 %	16898	91.2567 %
3	17912	96.7327 %	17802	96.1387 %	17912	96.7327 %
4	18411	99.4276 %	18230	98.4501 %	18320	98.9361 %
5	18517	100 %	18427	99.514 %	18517	100 %
6	18517	100 %	18517	100 %	18517	100 %

- Resultados obtenidos para `aint5`:

Valor de P	Método Exacto		Algoritmo Greedy		Algoritmo Greedy Aleatorizado	
	Demanda Cubierta	Porcentaje de la demanda Cubierta	Demanda Cubierta	Porcentaje de la demanda Cubierta	Demanda Cubierta	Porcentaje de la demanda Cubierta
1	8405	88.8854 %	8405	88.8854 %	8405	88.8854 %
2	9134	96.5948 %	9097	96.2035 %	9117	96.415 %
3	9365	99.0376 %	9312	98.4772 %	9337	98.7415 %
4	9456	100 %	9404	99.4501 %	9429	99.7145 %
5	9456	100 %	9456	100 %	9456	100 %
6	9456	100 %	9456	100 %	9456	100 %