Algoritmos Greedy

Problema QAP

Daniel Bolaños Martínez

José María Borrás Serrano

Santiago de Diego de Diego

Fernando de la Hoz Moreno

1. Análisis del problema

El problema, P está basado en el Problema de Asignación Cuadrática, consiste en asignar a cada oficinista de un grupo de oficinistas, una habitación de un grupo de habitaciones de forma que se minimice el coste de asignar a cada habitación i el oficinista p(i). El Problema puede generalizarse a asignar N instalaciones a una cantidad N de locaciones en donde se considera un costo asociado a cada una de las asignaciones deseando que el coste en función del flujo sea mínimo. Matemáticamente, el problema puede definirse como:

$$p^* = min_p H(p) = min_p \sum_{i=0}^{N-1} \sum_{i=0}^{N-1} f_{p(i)p(j)} d_{ij}$$

donde f es la matriz de flujos de intersección desde el oficinista p(i) al p(j) y d es la matriz de distancias desde la habitación i a la j.

Por ejemplo, la matriz de distancias d puede ser:

0	7	14	20	3
4	0	10	17	49
51	1	0	43	71
7	3	10	0	20
90	101	47	3	0

y la matriz de flujos:

0	4	7	4	1
0	0	10	3	21
0	0	0	47	3
41	21	7	0	9
21	43	32	0	27

Además definimos un vector de distancias entre habitaciones $d_p(i) = \sum_{j=0}^{N-1} d_{ij}$. Cada componente del vector contiene la suma de las distancias desde la localización i al resto.

En nuestro caso dicho vector vendria dado como:

$$d_p = (44, 80, 166, 40, 241)$$

Si realizamos lo mismo con la matriz de flujos f, tendremos el vector :

$$f_p = \sum_{b=0}^{N-1} f_{ab}$$

que en nuestro ejemplo particular es:

$$f_p = (16, 24, 50, 78, 123)$$

2. Diseño de la solución empleando la metodología Greedy

En un algoritmo voraz los pasos que tenemos que seguir son:

- Diseñar una lista de candidatos
- Identificar una lista de candidatos ya utilizados
- Diseñar una función solución
- Diseñar un criterio de factibilidad
- Diseñar una función de selección del candidato más prometedor para formar parte de la solución
- Encontrar una función objetivo de minimización/maximización.
- Adaptar la estructura general del procedimiento Greedy al problema y resolverlo.

Nosotros hemos implementado los pasos de la siguiente forma:

- El **conjunto C de candidatos**, que son los oficinistas y las habitaciones.
- **Función solución.** Comprueba, en cada paso, si el subconjunto actual de candidatos elegidos forma una solución (no importa si es óptima o no lo es). En nuestro caso una vez el conjunto de candidatos esté vacío habremos llegado a la solución.
- **Función selección.** Informa cuál es el elemento más prometedor para completar la solución. Éste no puede haber sido escogido con anterioridad. Cada elemento es considerado una sola vez. Luego, puede ser rechazado o aceptado. Para nuestro algoritmo escogemos el oficinista con menor flujo de trabajo y le asignamos la habitación con mayor distancia.
- **Función de factibilidad.** Informa si a partir de un conjunto se puede llegar a una solución. En este caso la función de selección ya implementa la de factibilidad porque escogemos un elemento que es factible.

- **Función objetivo.** Es aquella que queremos maximizar o minimizar, el núcleo del problema. En nuestro problema queremos asignar a oficinistas con la máxima carga de trabajo a habitaciones con la mínima distancia.

3. Pseudocódigo que soluciona el problema

A continuación se presenta el pseudocódigo de la función Greedy que soluciona nuestro problema y que nos ha servido de esqueleto para programar la función definitiva:

```
S <- 0
Mientras (habitaciones != 0 && oficinistas != 0) hacer:
   x1 = Selección de la habitación con mínima
        distancia potencial entre las habitaciones.
   x2 = Selección del oficinista con máximo
        flujo potencial entre los oficinistas.
   habitaciones = habitaciones \ {x1}
   oficinistas = oficinista \ {x2}
   S[x1] = x2
Fin-Mientras
Devolver S
```

La función Greedy recorre las dos listas de candidatos (habitaciones y oficinistas en cada caso) hasta que estén vacíos y en cada iteración, obtiene el índice de la habitación con mínima distancia y el índice del oficinista con máximo flujo de trabajo asignando ambos datos en el vector solución.

4. Explicación del funcionamiento del algoritmo sobre el ejemplo

Para realizar el problema planteado se ha empleado un enfoque orientado a objetos empleando el lenguaje C++.

A la clase Problema le pasamos dos matrices cuadradas, una con las distancias entre las habitaciones y otra con los flujos de trabajo de los oficinistas.

La función Greedy genera los vectores potenciales asociados a las matrices que se pasan como parámetros en el Problema, llamando a la función candidatos.

Cuando tenemos los vectores potenciales que conforman ambas listas de candidatos, se llama a las funciones de selección las cuales devuelven el índice del máximo o mínimo valor

en cada caso; utilizaremos, dependiendo de la lista de candidatos, una u otra función de selección. En nuestro caso hemos optado por asignar a la habitación con distancia menor el mayor flujo de trabajo.

Una vez obtenidos los valores con la función de selección, asignamos los índices de ambos valores en su vector al vector solución y finalmente eliminamos los valores de las listas de candidatos.

5. Enunciado de un problema o caso real donde se pueda aplicar el algoritmo

El algorítmo de Asignación Cuadrática explicado es utilizado actualmente es muchas empresas para optimizar la ganancia de beneficios gastando el mínimo número de recursos.

Algunos de los ejemplos que hemos visto más destacables y que muestran el caso de forma explícita son:

Diseño de centros comerciales donde se quiere que el público recorra la menor cantidad de distancia para llegar a tiendas de mayor interés común.

Diseño de circuitos eléctricos, en donde es de relevante importancia dónde se ubican ciertas partes o chips con el fin de minimizar la distancia entre ellos, ya que las conexiones son de alto costo.

Para la resolución de ambos problemas, podríamos aplicar exactamente los mismos pasos utilizados en nuestro algoritmo para obtener una solución adecuada. Nos centraremos más en uno de los casos citados en las diapositivas de la práctica.

6. Cálculo del orden de eficiencia teórica del algoritmo

```
//Eficiencia: O(n^2)
S <- 0
Mientras (habitaciones != 0 && oficinistas != 0) hacer:
        //Eficiencia: O(n)
   x1 = Selección de la habitación con mínima
        distancia potencial entre las habitaciones.
        //Eficiencia: O(n)
   x2 = Selección del oficinista con máximo
        flujo potencial entre los oficinistas.
   habitaciones = habitaciones \ {x1}
```

```
   oficinistas = oficinista \ {x2}
   S[x1] = x2
Fin-Mientras
Devolver S
```

Si denotamos como n el número de filas y de columnas de la matriz (recordar que son matrices cuadradas), entonces se tiene que el orden de eficiencia del algoritmo de búsqueda de la solución es $O(n^2)$.

Esto es debido a que el algoritmo se compone de dos funciones de selección distintas, cada una de ellas recorre el vector de principio a fin, el cual tiene longitud n. Como las matrices son de tamaño $n \times n$, es necesario que el vector sea de longitud n para poder realizar la multiplicación correctamente. Por último, se engloban ambas funciones de selección dentro de un bucle *for* que de tamaño n, por lo que la eficiencia total viene dada, como apuntábamos, por:

$$\text{Eficiencia} = (2n)n = 2n^2 = O(n^2)$$

7. Instrucciones sobre como compilar y ejecutar el código de la práctica

Compilación: Ejecutar make en su terminal.

Ejecución:

1. Crear un archivo de texto con el siguiente formato.

   - Numero de filas/columnas.
   - Matriz de distancias a Habitaciones.
   - Matriz de flujo de trabajo de Oficinistas.

Formato similar al proporcionado en dat/matrices.dat

2. Ejecutar el programa que se guardará por defecto en la carpeta bin del proyecto ./bin/prueba *nombre_archivo*