UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación
e Inteligencia Artificial

Algorítmica

Práctica 3

Algoritmos greedy

Curso 2013-2014

Doble Grado en Ingeniería Informática y Matemáticas

Práctica 3

Algoritmos greedy

El objetivo de esta práctica es que el estudiante aprecie la utilidad de los métodos voraces (greedy) para resolver problemas de forma muy eficiente. Para ello, cada equipo deberá resolver los problemas que se describen en este guión.

En la resolución de los problemas de esta práctica, deberá tener en cuenta que los algoritmos voraces en algunos casos proporcionan la solución óptima del problema y, en otros casos, sólo aproximaciones. Por tanto, deberá ser capaz de identificar ambos casos y tener en cuenta este aspecto a la hora de exponer y defender sus soluciones en clase.

También deberá implementar algoritmos voraces para resolver el problema de la asignación cuadrática, siguiendo las indicaciones que se recogen en la sección 7 de este guión de prácticas.

1 Terminales de venta

Nuestra empresa desarrolla software para terminales de puntos de venta [TPV] y desea que resolvamos los siguientes problemas:

- En su versión para máquinas expendedoras, se pretende minimizar el número de monedas empleado para darle el cambio al cliente. Diseñe un algoritmo greedy que devuelva un número mínimo de monedas (de 0.01, 0.02, 0.05, 0.10, 0.20, 0.50 y 1 euro).

- Nuestra empresa también distribuye máquinas expendedoras de sellos de correos (de 0.54, 0.32, 0.17 y 0.01 euros) y decidimos reutilizar nuestro algoritmo greedy para decidir qué sellos ha de devolver la máquina. Demuestre que el algoritmo greedy no proporciona necesariamente una solución óptima, aun disponiendo de un suministro inagotable de sellos de cada valor.

2 Red de comunicaciones

Supongamos que el coste de tender una red de fibra óptica entre dos ciudades es proporcional a la distancia euclídea entre ellas.

- Diseñe un algoritmo que permita interconectar un conjunto de ciudades minimizando el coste de la red de interconexión.
- Busque un ejemplo en el que se demuestre que puede resultar más económico instalar una centralita entre ciudades que utilizar solamente conexiones directas entre ciudades

3 Segmentación de clientes

El departamento de marketing de nuestra empresa desea segmentar nuestro conjunto de clientes en K grupos para diseñar campañas de marketing dirigidas a cada grupo por separado.

Si suponemos que podemos caracterizar a cada cliente mediante un vector de características $(c_1, .., c_n)$ y medir la similitud entre dos clientes como la inversa de la distancia entre sus vectores de características, diseñe un algoritmo greedy que nos permita agrupar a nuestros clientes en K grupos diferentes de tal forma que se maximice la distancia entre clientes de distintos grupos.

4 Asignación de trabajos

Supongamos que disponemos de n trabajadores y n tareas. Sea $c_{ij} > 0$ el coste de asignarle el trabajo j al trabajador i. Una asignación válida es aquélla en la que a cada trabajador le corresponde una tarea y cada tarea la realiza un trabajador diferente. Dada una asignación válida, definimos el coste de dicha asignación como la suma total de los costes individuales.

- Diseñe distintas estrategias greedy para asignar tareas.
- Encuentre contrajemplos que demuestren que ninguna de ellas nos permite encontrar la solución óptima con un algoritmo greedy.

5 Asignación de aulas

Deseamos optimizar el uso de las aulas de un centro educativo. Dados un conjunto de aulas y un conjunto de clases con un horario preestablecido (la clase i empieza a la hora s_i y termina a la hora f_i), diseñe un algoritmo greedy que minimice el número de aulas necesario para impartir toda la docencia del centro.

6 Memorias caché

Disponemos de una memoria caché con capacidad para almacenar k datos. Cuando la caché recibe una solicitud d_i, puede que el dato ya esté en la caché [hit] o que tengamos que buscarlo en memoria [miss], en cuyo caso reemplazaremos alguno de los datos ya almacenados en la caché por el dato que acabamos de obtener.

Si conocemos de antemano el conjunto de solicitudes que deberemos atender $d_1..d_m$, diseñe un algoritmo greedy que minimice el número de fallos de caché.

7 El problema de la asignación cuadrática

El problema de la asignación cuadrática (QAP, Quadratic assignment problem) es un problema fundamental de optimización combinatoria con numerosas aplicaciones. El problema se puede describir de la siguiente forma:

Supongamos que queremos decidir dónde construir n instalaciones (p.ej. fábricas) y tenemos n posibles localizaciones en las que podemos construir dichas instalaciones. Conocemos las distancias que hay entre cada par de instalaciones y también el flujo de materiales que ha de existir entre las distintas instalaciones (p.ej. la cantidad de suministros que deben transportarse de una fábrica a otra). El problema consiste en decidir dónde construir cada instalación de forma que se minimize el coste de transporte de materiales.

Formalmente, si llamamos $d(i, j)$ a la distancia de la localización i a la localización j y $w(i, j)$ al peso asociado al flujo de materiales que ha de transportarse de la instalación i a la instalación j, hemos de encontrar la asignación de instalaciones a localizaciones que minimiza la función de coste

$$\sum_{i,j} w(i,j)d(p(i),p(j))$$

donde p() define una permutación sobre el conjunto de instalaciones.

Igual que en el problema del viajante de comercio, que se puede considerar un caso particular del QAP, una solución para el problema es una permutación del conjunto de instalaciones que indica dónde se debe construir cada una.

El problema de la asignación cuadrática es un problema habitual en investigación operativa y, además de utilizarse para decidir la ubicación de plantas de producción, también se utiliza como modelo para colocar los componentes electrónicos de un circuito sobre una placa impresa o los módulos de un circuito integrado en la superficie de un microchip.

Por su interés teórico y práctico, existe una variedad muy amplia de algoritmos que abordan la resolución del problema de la asignación cuadrática. Al ser un problema NP-completo, el diseño y aplicación de algoritmos exactos para su resolución no es viable cuando n es grande. Nos centraremos, por tanto, en el diseño de algoritmos aproximados de tipo greedy y evaluaremos su rendimiento sobre instancias concretas del problema.

7.1 Heurísticas

Para diseñar algoritmos greedy que resuelvan el problema, probaremos con distintas estrategias heurísticas para poder evaluar cuáles funcionan mejor en la práctica:

- Estrategias constructivas.
- Estrategias de transposición.

Para el primer tipo de estrategia, utilizaremos una variante de la heurística del *vecino más cercano*, cuyo funcionamiento es extremadamente simple: dada la localización $p(i)$ de una instalación i, a continuación se escoge la instalación j con la i tiene un mayor flujo asociado $w(i, j)$ y se construye en la localización $p(j)$ que esté más cerca del lugar $p(i)$ donde hemos situado la instalación i. En este tipo de algoritmos, podemos considerar, una por una, distintas combinaciones de punto de partida y devolver el mejor resultado obtenido en las distintas ejecuciones realizadas.

En las estrategias de transposición, la idea es comenzar con una permutación inicial e ir intercambiando las posiciones de dos instalaciones mediante algún criterio de tipo greedy. Para poder implementar este tipo de estrategia, han de definirse dos elementos:

1. Cómo se construye la permutación inicial.
2. Qué transposiciones se consideran en cada momento, p.ej. 2-opt ó 3-opt.
3. Cuántas transposiciones realizamos antes de devolver un resultado.

La permutación inicial se puede construir al azar en tiempo lineal o, directamente, utilizando la heurística del vecino más cercano, lo que tiene un coste mayor.

Si, al realizar un intercambio de posición, disminuye el coste asociado a la solución, entonces mantenemos el intercambio. Si no, el intercambio no se efectúa y seguimos con la solución que ya teníamos.

En pseudocódigo, un algoritmo greedy basado en 2-opt para el problema QAP podría tener el siguiente aspecto:

```
1  S = candidato inicial con coste c(S)
2
3  do {
4
5    mejor = S
6
7    for i=1..n
8        for j=i+1..n
9            T = S tras intercambiar i con j
10           if c(T) < c(S)
11              S = T
12
13 } while (S != mejor)
```

NOTA: El algoritmo 2-opt asociado al problema consideraría $n(n-1)/2$ intercambios antes de modificar la mejor solución actual S y, en cada iteración, realizaría un único intercambio (aquél con el que se obtuviese un coste menor de entre todas las transposiciones). El algoritmo greedy hace el intercambio permanente en cuanto encuentra una mejora, lo que le permite ser más eficiente a costa de encontrar peores soluciones que el algoritmo 2-opt.

7.2 Datos de prueba

Los casos de prueba para comprobar el funcionamiento de las distintas heurísticas greedy están disponibles en la plataforma de docencia de la asignatura y se han obtenido de la biblioteca QAPLIB (`http://www.seas.upenn.edu/qaplib/`).El formato de los ficheros de datos es el siguiente:

```
n
A
B
```

donde n es el tamaño del problema, mientras que A y B son las matrices de flujos y distancias, indistintamente (el problema es simétrico, por lo que no influye en el resultado cuál es cuál).

7.3 Implementación y análisis de resultados

- Implemente un programa que resuelva el problema de la asignación cuadrática empleando las dos heurísticas descritas y una tercera heurística que deberá diseñar cada equipo de prácticas. Su implementación debe proporcionar la permutacióno obtenida y el coste asociado a la misma.
- Realice un estudio comparativo de las tres técnicas heurísticas empleando para ello el conjunto de datos de prueba que se proporciona con este guión.
- Analice la eficiencia de los algoritmos greedy resultantes de utilizar cada una de las tres heurísticas (las dos propuestas en el guión y la tercera diseñada por su equipo de prácticas).