UNIVERSIDAD DE GRANADA

E.T.S. DE INGENIERÍAS INFORMÁTICA y DE TELECOMUNICACIÓN

Departamento de Ciencias de la Computación e Inteligencia Artificial

Algorítmica

Guión de Prácticas

Práctica 3: Algoritmos Voraces (Greedy)

Curso 2017-2018

Grado en Informática

1. Objetivo

El objetivo de esta práctica es que el estudiante aprecie la utilidad de los métodos voraces (greedy) para resolver problemas de forma muy eficiente, en algunos casos obteniendo soluciones óptimas y en otros aproximaciones. Para ello cada equipo de estudiantes deberá resolver uno de los problemas (asignado al azar) que se describen en la sección 2, así como exponer y defender su propuesta en clase. Adicionalmente, todos los equipos deberán implementar algoritmos voraces para resolver el problema del viajante de comercio, siguiendo las indicaciones descritas en la sección 3.

2. Problemas

2.1. Contenedores en un barco

Se tiene un buque mercante cuya capacidad de carga es de K toneladas y un conjunto de contenedores $c_1, \ldots, c_n$ cuyos pesos respectivos son $p_1, \ldots, p_n$ (expresados también en toneladas) y para los que se obtiene un beneficio al ser transportados de $b_1, \ldots, b_n$ euros, respectivamente. Teniendo en cuenta que la capacidad del buque es menor que la suma total de los pesos de los contenedores:

- Diseñe un algoritmo que maximice el número de contenedores cargados, y demuestre su optimalidad.
- Diseñe un algoritmo que intente maximizar el beneficio obtenido.

2.2. Supercomputador

Una empresa necesita realizar un sofisticado y costoso proceso informático cada cierto tiempo. La empresa dispone de un potente supercomputador así como de gran número (ilimitado, a efectos prácticos) de ordenadores personales de gama alta. El proceso se puede dividir en n subprocesos que pueden realizarse de forma completamente independiente. Cada subproceso consta de dos etapas. La primera debe realizarse en el supercomputador, y después la segunda se realiza en uno de los PCs. Se conoce que el subproceso i-ésimo necesita $p(i)$ segundos de tiempo en el supercomputador, seguidos de $f(i)$ segundos en un PC.

Puesto que hay al menos n PCs disponibles, la segunda parte de cada subproceso puede realizarse en paralelo (con todos los subprocesos ejecutándose simultáneamente). Sin embargo, el supercomputador solo puede trabajar con un subproceso cada vez, de modo que el gestor del sistema necesita decidir el orden en que los subprocesos son procesados en el supercomputador. El proceso global termina cuando todos los subprocesos lo han hecho (es decir el tiempo de proceso global es el máximo de los tiempos de finalización de los subprocesos). Se desea encontrar el orden en que procesar los subprocesos en el supercomputador que minimice el tiempo de proceso global.

Diseñar un algoritmo greedy para realizar esta tarea y demostrar su optimalidad.

2.3. Minimizando el tiempo medio de acceso

Sean n programas $P_1, P_2, \ldots, P_n$ que hay que almacenar en una cinta. El programa P_i requiere s_i kilobytes de espacio y la cinta es suficientemente larga para almacenar todos los programas.

Se sabe con qué frecuencia se utiliza cada programa: una fracción π_i de las solicitudes afecta al programa P_i (y por tanto $\sum_{i=1}^{n} \pi_i = 1$). Los datos se almacenan en la cinta con densidad constante y la velocidad de la cinta también es constante. Una vez que se carga el programa, la cinta se rebobina hasta el principio.

Si los programas se almacenan por orden $i_1, i_2, \ldots, i_n$ el tiempo medio requerido para cargar un programa es, por tanto:

$$\hat{T} = c \sum_{j=1}^{n} \left[\pi_{i_j} \sum_{k=1}^{j} s_{i_k} \right]$$

donde la constante c depende de la densidad de grabación y de la velocidad de la cinta.

Se desea minimizar $\hat{T}$ empleando un algoritmo voraz. Demuestre la optimalidad del algoritmo o encuentre un contraejemplo que muestre que el algoritmo no es óptimo para los siguientes criterios de selección:

- Programas en orden no decreciente de s_i.
- Programas en orden no creciente de π_i.
- Programas en orden no creciente de π_i/s_i.

2.4. Bar de tapas

Un restaurante de tapas ofrece un amplio surtido de n tapas, todas al mismo precio. De cada tapa t_i se conoce el número de calorías c_i que contiene. Se desea elegir como menú un subconjunto de tapas (sin repetir ninguna) que garantice un consumo de un mínimo de M calorías, pagando el menor precio posible. Diseñad un algoritmo voraz para resolver el problema y demostrad su optimalidad.

2.5. Recubrimiento de puntos

Dado un conjunto de puntos de la recta real $\{x_1, x_2, \ldots, x_n\}$, con $x_i \leq x_{i+1}$, $i = 1, \ldots, n-1$, se pretende determinar el menor conjunto de intervalos cerrados de longitud 1 que contienen a todos los puntos.

Diseñar un algoritmo greedy para resolver esta tarea. Demostrar que el algoritmo obtiene la solución óptima.

3. El problema del viajante de comercio

En su formulación más sencilla, el problema del viajante de comercio (TSP, por Traveling Salesman Problem) se define como sigue: dado un conjunto de ciudades y una matriz con las distancias entre todas ellas, un viajante debe recorrer todas las ciudades exactamente una

vez, regresando al punto de partida, de forma tal que la distancia recorrida sea mínima. Mas formalmente, dado un grafo G, conexo y ponderado, se trata de hallar el ciclo hamiltoniano de mínimo peso de ese grafo.

Una solución para TSP es una permutación del conjunto de ciudades que indica el orden en que se deben recorrer. Para el cálculo de la longitud del ciclo no debemos olvidar sumar la distancia que existe entre la última ciudad y la primera (hay que cerrar el ciclo).

Por su interés teórico y práctico, existe una variedad muy amplia de algoritmos para abordar la solución del TSP y sus variantes (siendo un problema NP-Completo, el diseño y aplicación de algoritmos exactos para su resolución no es factible en problemas de cierto tamaño). Nos centraremos en una serie de algoritmos aproximados de tipo greedy y evaluaremos su rendimiento en un conjunto de instancias del TSP. Para el diseño de estos algoritmos, utilizaremos dos enfoques diferentes: a) estrategias basadas en alguna noción de cercanía, y b) estrategias de inserción.

En el primer caso emplearemos la heurística del *vecino más cercano*, cuyo funcionamiento es extremadamente simple: dada una ciudad inicial v_0, se agrega como ciudad siguiente aquella v_i (no incluída en el circuito) que se encuentre más cercana a v_0. El procedimiento se repite hasta que todas las ciudades se hayan visitado.

En las estrategias de inserción, la idea es comenzar con un recorrido parcial, que incluya algunas de las ciudades, y luego extender este recorrido insertando las ciudades restantes mediante algún criterio de tipo greedy. Para poder implementar este tipo de estrategia, deben definirse tres elementos:

1. Cómo se construye el recorrido parcial inicial.

2. Cuál es el nodo siguiente a insertar en el recorrido parcial.

3. Dónde se inserta el nodo seleccionado.

El recorrido inicial se puede construir a partir de las tres ciudades que formen un triángulo lo más grande posible: por ejemplo, eligiendo la ciudad que está más a Este, la que está más al Oeste, y la que está más al norte.

Cuando se haya seleccionado una ciudad, ésta se ubicará en el punto del circuito que provoque el menor incremento de su longitud total. Es decir, hemos que comprobar, para cada posible posición, la longitud del circuito resultante y quedarnos con la mejor alternativa.

Por último, para decidir cuál es la ciudad que añadiremos a nuestro circuito, podemos aplicar el siguiente criterio, denominado *inserción más económica*: de entre todas las ciudades no visitadas, elegimos aquélla que provoque el menor incremento en la longitud total del circuito. En otras palabras, cada ciudad debemos insertarla en cada una de las soluciones posibles y quedarnos con la ciudad (y posición) que nos permita obtener un circuito de menor longitud. Seleccionaremos aquella ciudad que nos proporcione el mínimo de los mínimos calculados para cada una de las ciudades.

3.1. Tareas a realizar

- Implementar un programa que proporcione soluciones para el problema del viajante de comercio empleando las dos heurísticas descritas anteriormente, así como otra adicional

propuesta por el propio equipo. El programa debe proporcionar el recorrido obtenido y la longitud de dicho recorrido.

- Realizar un estudio comparativo de las tres estrategias empleando un conjunto de datos de prueba.

3.2. Datos de prueba

Los casos de prueba para comprobar el funcionamiento de los algoritmos están disponibles en la plataforma de docencia de la asignatura, y se han obtenido y adaptado de la librería TSPLIB[1]. El formato de los ficheros es el siguiente:

```
DIMENSION: 52
1 565.0 575.0
2 25.0 185.0
3 345.0 750.0
.............
50 595.0 360.0
51 1340.0 725.0
```

La primera fila indica la cantidad de ciudades en el fichero. A continuación, aparece una fila por cada ciudad conteniendo tres valores: el número de ciudad, la coordenada X y la coordenada Y.

En cada ejecución del algoritmo, deberá calcular la matriz de distancias entre todas las ciudades teniendo en cuenta lo siguiente: sean (x_i, y_i), (x_j, y_j) las coordenadas en el plano de las ciudades c_i, c_j . En primer lugar se calcula la distancia euclídea d entre ambos puntos como

$$d = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

Después la distancia d debe redondearse al entero más próximo.

3.3. Visualización

Si se desean visualizar los diferentes problemas, se puede utilizar gnuplot. Simplemente, si `fichero.tsp` es el fichero que contiene las coordenadas de las ciudades (en el formato anterior), basta con utilizar el comando siguiente (para indicarle a gnuplot que utilice los datos de la columna 2 del fichero como abscisas y los datos de la columna 3 como ordenadas en la representación gráfica):

```
gnuplot> plot "fichero.tsp" using 2:3 with points
```

Si las diferentes ciudades (filas del fichero) se reordenan de acuerdo al orden de un circuito concreto (por ejemplo el generado por nuestro algoritmo, o el óptimo) entonces es posible hacer también con gnuplot una representación gráfica del circuito. Simplemente, si el fichero de coordenadas reordenado es `ficheroreord.tsp`, basta con ejecutar el comando siguiente (al usar la opción `with lines` se unen mediante una línea recta los puntos consecutivos, que al

[1] `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/`

estar en el orden del tour, generan una representación visual del recorrido, a falta de cerrar el ciclo)):

```
gnuplot> plot "ficheroreord.tsp" using 2:3 with lines
```