

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

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 estudiante realizará el problema de las reparaciones de forma individual y cada equipo de estudiantes deberá resolver el problema del viajante de comercio, siguiendo las indicaciones descritas en la sección 3.

2 Problemas

2.1 Reparaciones

Un electricista necesita hacer n reparaciones urgentes, y sabe de antemano el tiempo que le va a llevar cada una de ellas: en la tarea i -ésima tardará t_i minutos.

Como en su empresa le pagan dependiendo de la satisfacción del cliente y esta es inversamente proporcional al tiempo que tardan en atenderles, necesita decidir el orden en el que atenderá los avisos para minimizar el tiempo medio de atención de los clientes (desde el inicio hasta que su reparación es efectuada).

- Diseñar un algoritmo greedy para resolver esta tarea. Demostrar que el algoritmo obtiene la solución óptima.
- Modificar el algoritmo anterior para el caso de una empresa en la que se disponga de los servicios de más de un electricista.

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 incluida 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 de 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 aquella 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 adaptados de la librería TSPLIB¹. El formato de los ficheros es el siguiente:

```
DIMENSION: 51
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
```

¹<http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/>

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 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
```