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 2018-2019

Grado en Ingeniería Informática

1 Normativa

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 deberá realizar lo siguiente:

1. **Problema común**: Un problema a resolver por todos los equipos.
2. **Problema asignado**: Cada equipo tendrá que resolver un problema que se asignará de forma aleatoria cuando se presente la Práctica 3.
3. La generación de datos para cada problema la debe realizar cada equipo.

Cada equipo tendrá que entregar:

1. Una memoria en la que se describa la solución propuesta para cada uno de los problemas (común y asignado). No puede obviarse en la descripción las componentes del método voraz diseñado y el pseudocódigo del problema. Además se debe detallar el proceso de generación del problema, y varios escenarios de ejecución junto con sus resultados.
2. El código elaborado para resolver ambos problemas. El código se debe entregar indicando cómo debe ser compilado. Se aconseja entregar el Makefile.

2 Problema común

En su formulación más sencilla, el problema del viajante de comercio (TSP, por sus siglas en inglés *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.[1] Nos centraremos en una serie de algoritmos aproximados de tipo voraz y evaluaremos su rendimiento en un conjunto de instancias del TSP. Para el diseño de estos algoritmos, utilizaremos dos enfoques diferentes: (1) basado en cercanía, y (2) basado en inserción.

Basado en cercanía. Estrategia basada en el *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.

Basado en 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 voraz. Para poder implementar este tipo de estrategia, deben definirse tres elementos:

1. Cómo se construye el recorrido parcial.
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 al 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

[1] El problema TSP es un problema NP-Completo, el diseño y aplicación de algoritmos exactos para su resolución no es factible en problemas de cierto tamaño

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.

Tareas. Se debe realizar lo siguiente:

1. Implementar un programa que resuelva el problema TSP siguiendo la estrategía de cercanía e inserción.
2. Diseñar e implementar una nueva estrategia de resolución del problema TSP.
3. Realizar un estudio comparativo de las tres estrategias usando un conjunto de datos de prueba.

Datos de prueba. Los datos que se tienen que usar para evaluar esta tarea están disponibles en PRADO. Los datos se han obtenido de la librería TSPLIB.[2] Cada problema está descrito por dos ficheros:

Fichero `.tsp`. Fichero en el que se indica tanto el número de ciudades (`DIMENSION`) como sus coordenadas x e y. Los datos de cada ciudad están en una nueva línea.

Fichero `.opt.tour`. Fichero en el se muestra el resultado que debe devolver el algoritmo.

Las distancias entre las ciudadades se debe calcular mediante la distancia euclídea, es decir, dadas las coordenadas de dos ciudades $((x_1, y_1), (x_2, y_2))$ la distancia entre ellas se calcularía de la siguiente manera:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

La distancia d debe redondearse al entero más próximo.

Visualización. Se puede usar cualquier programa para la visualización de los ciudades (puntos) en el espacio. Si se usa `gnuplot`, tomando los ficheros de datos que se proporcionan se puede visualizar las ciudades ejecutando el siguiente comando:

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

Una vez que se haya obtenido la solución y se genera el fichero con las ciudades y los puntos se puede generar el recorrido del viajante de comercio usando `gnuplot` de la siguiente manera:

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

3 Problema a asignar

3.1 Ahorro en gasolina

Un camionero conduce desde una ciudad A a otra ciudad B siguiendo una ruta dada y llevando un camión que le permite, con el tanque de gasolina lleno, recorrer n kilómetros sin parar. El camionero dispone de un mapa de carreteras que le indica las distancias entre las gasolineras que hay en su ruta. Como va con prisa, el camionero desea pararse a repostar el menor número de veces posible. Deseamos diseñar un algoritmo voraz para determinar en qué gasolineras tiene que parar, minimizando el número de paradas.

[2] `https://wwwproxy.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/`

3.2 Asignación de tareas

Tenemos que completar un conjunto de n tareas con plazos límite. Cada una de las tareas consume la misma cantidad de tiempo (una unidad) y, en un instante determinado, podemos realizar únicamente una tarea. La tarea i tiene como plazo límite d_i y produce un beneficio g_i ($g_i > 0$) sólo si la tarea se realiza en un instante de tiempo $t \leq d_i$. Diseñe un algoritmo voraz que nos permita seleccionar el conjunto de tareas que nos asegure el mayor beneficio posible.

3.3 Recubrimiento de un grafo no dirigido

Consideremos un grafo no dirigido $G = (V, E)$. Un conjunto U se dice que es un recubrimiento de G si $U \subseteq V$ y cada arista en E incide en, al menos, un vértice o nodo de U, es decir $\forall (x, y) \in E$, bien $x \in U$ o $y \in U$. Un conjunto de nodos es un recubrimiento minimal de G si es un recubrimiento con el menor número posible de nodos.

1. Diseñar un algoritmo voraz para intentar obtener un recubrimiento minimal de G. Demostrar que el algoritmo es correcto, o dar un contraejemplo.

2. Diseñar un algoritmo voraz que obtenga un recubrimiento minimal para el caso particular de grafos que sean árboles.

3. Opcionalmente, realizar un estudio experimental de las diferencias entre los dos algoritmos anteriores cuando ambos se aplican a árboles.

3.4 Trabajadores y Tareas

Supongamos que disponemos de n trabajadores y n tareas. Sea $c_{i,j} > 0$ el coste de asignarle la tarea j al trabajador i. Una asignación válida es aquella 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 un algoritmo voraz para obtener una asignación de tareas a trabajadores óptima.

3.5 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). Teniendo en cuenta que la capacidad del buque es menor que la suma total de los pesos de los contenedores:

1. Diseñe un algoritmo que maximice el número de contenedores cargados, y demuestre su optimalidad.
2. Diseñe un algoritmo que intente maximizar el número de toneladas cargadas.