Algorítmica

grado en ingeniería informática

Práctica 3

El viajante de comercio

Autores
María Jesús López Salmerón
Nazaret Román Guerrero
Laura Hernández Muñoz
José Baena Cobos
Carlos Sánchez Páez

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2017-2018

Índice

Índice de figuras

1. Descripción de la práctica

El objetivo de esta práctica es abarcar el problema del viajante de comercio (TSP, *Travel Salesman Problem*) mediante estrategias voraces. En concreto, seguiremos la heurística de *Branch and Bound*.

Tiene las siguientes características:

- **Conjunto de candidatos**. Ciudades a visitar.
- **Conjunto de seleccionados**. Aquellas ciudades que vayamos incorporando al circuito.
- **Función solución**. Todas las ciudades han sido visitadas y hemos vuelto a la primera.
- **Función de factibilidad**. La ciudad no ha sido visitada aún.
- **Función selección**. De entre todos los candidatos, elegimos aquella ciudad que incrementa menos el coste del circuito que llevamos hasta el momento.

2. Descripción del algoritmo

2.1. Datos y estructuras utilizadas

- **Distancia**. Comienza inicializada a $+\infty$.
- **Cota inferior**. Utilizamos una cota inferior optimista que inicializamos mediante un algoritmo greedy, aproximado al algoritmo *vecino más cercano*. La heurística que sigue el algoritmo es la siguiente:

$$\frac{1}{2}\sum_{i=0}^{n} coste_{entrada}(i) + coste_{salida}(i)$$

  En la sumatoria se acumulan progresivamente los costes de entrada y de salida de cada nodo que sean menores entre las aristas posibles de cada uno. Tras completar la sumatoria se divide a la mitad debido a que la salida de un nodo es la entrada del siguiente.
- **Solución parcial**. Es un vector que contiene la solución, pudiendo o no estar completa. En el caso de que esté completa pasa a ser la nueva cota.
- **Visitados**. Es un vector de booleanos que contiene *true* si la ciudad ya ha sido visitada y *false* en otro caso.

2.2. Procedimiento

El ajuste inicial que se lleva a cabo es el siguiente:

1. Se calcula la cota inferior inicial mediante el algoritmo greedy anteoriormente explicado.
2. Se toma la primera ciudad y se introduce en la solución parcial. La ciudad ya ha sido visitada, por tanto se pone a *true* en el vector de visitados.

3. Se llama entonces al método recursivo que lleva a cabo el algoritmo *Branch and Bound* propiamente dicho.

El seguimiento del algoritmo es el que sigue:

1. En el caso base se comprueba si hemos llegado a un nodo hoja del árbol. Si efectivamente estamos en un nodo hoja, cerramos el circuito y comprobamos si la solución parcial es mejor que la actual (la distancia es menor). En caso afirmativo, ésta se actualiza.

2. Si no estamos en el caso base, se siguen los siguientes pasos

   a) Recorremos todas las ciudades restantes.

   b) Comprobamos que no ha sido visitada y que no es la misma ciudad en la que estamos actualmente.

   c) Calculamos el coste que supone añadir la nueva ciudad al recorrido.

   d) Calculamos la cota de la rama actual. Se puede calcular de dos formas: si el nivel es el 1 la calculamos como la media entre el menor arco entrante de la última ciudad de la solución parcial y la nueva que queremos añadir. Si el nivel no es el 1, se calcula como la media entre el menor arco saliente de la última ciudad de la solución parcial y el menor arco entrante de la ciudad a añadir.

   e) Comprobamos si la suma entre la cota actual y el peso actual es menor que la distancia total, hemos encontrado una cota menor y por tanto exploramos los hijos mediante una llamada recursiva.

   f) Deshacemos los cambios que hemos realizado para que el siguiente hijo que evaluemos encuentre las variables en las mismas condiciones que el que se acaba de comprobar.

3. Resultados obtenidos

Número de ciudades	**Tiempo(s)**
6	$1{,}27 \cdot 10^{-5}$
7	$4{,}39 \cdot 10^{-5}$
8	0,0002036
9	0,0054381
10	0,0325048
11	0,381596
12	2,23487
13	8,90865
14	107,772 (2 minutos y 20 segundos)
15	1192,761 (19 minutos y 53 segundos)
16	(minutos y segundos)

4. Conclusiones

Lo más destacable de este algoritmo es que en términos de orden de eficiencia es pésimo pero, sin embargo, proporciona una solución muy óptima.

5. Anexo: código fuente