UGR

Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones
Grado en Ingeniería Informática

Asignatura: Algorítmica

Práctica 4 (Segunda parte): TSP

Autores:
Míriam Mengíbar Rodríguez
Néstor Rodríguez Vico
Ángel Píñar Rivas

Granada, 2 de junio de 2016

Índice:

1. Enunciado del problema.

Construir un programa que utilice la técnica de ramificación y acotación para resolver el problema del viajante de comercio en las condiciones descritas anteriormente, empleando la función de acotación comentada, o alguna otra.

2. Descripción del algoritmo.

Utilizamos nodos que representan ciudades. Dichos nodos contienen la ruta que hay hasta alcanzar dicha ciudad y el índice de la misma, así como el coste.

Los elementos de nuestro algoritmo son:

-Solución Parcial: mejorNodo (en concreto, el vector ruta que lleva dentro).

-Función de Poda: El coste del nodo actual mas el coste restante es mayor que el coste de la solución parcial.

-Restricciones Explícitas: Que las coordenadas de cada ciudad sean válidas.

-Restricciones Implícitas: No se pueden repetir ciudades y se deben añadir todas.

En nuestro algoritmo, que es iterativo, almacenamos los nodos vivos en una cola con prioridad, que está ordenada de la siguiente manera: el primer nodo será aquel cuya distancia desde el inicio de la ruta hasta dicho nodo (la distancia recorrida actual con dicho nodo incluido) más la distancia del nodo a su ciudad más cercana de las ciudades que faltan por añadir.

Escogemos un primer nodo (ciudad), la número 1, y la incluimos en la solución. A continuación, mientras que en la cola de nodos vivos aun haya nodos, sacamos el top y lo usamos para hacer comprobaciones:

-Si el nodo extraído es el ultimo, se calcula la distancia de la ruta hasta incluir dicha ciudad, y si es mejor que la ciudad mejor escogida en ese momento, se establece como la mejor.

-En caso contrario:

-Se comprueba si el nodo extraído es mejor candidato que el mejor nodo hasta el momento. Entonces, se comprueba si se produce algún cruce dentro de la ruta del nodo extraído. Si NO hay cruce, actualizamos la ruta del nodo extraído que podría encontrarse con un cruce para que lo evite escogiendo una nueva ruta, y una vez actualizada esta información además de las nuevas distancias o costes, se introduce de nuevo en la cola de nodos vivos.

-Si no es así, no hacemos nada, es en este caso cuando se realiza la poda.

Una parte a mencionar es que durante el algoritmo hemos introducido algunas partes para extraer información adicional, concretamente el número de nodos expandidos y el número de podas realizadas (el resto de datos que nos interesan se extraen en otras partes del código).

Pseudocódigo:

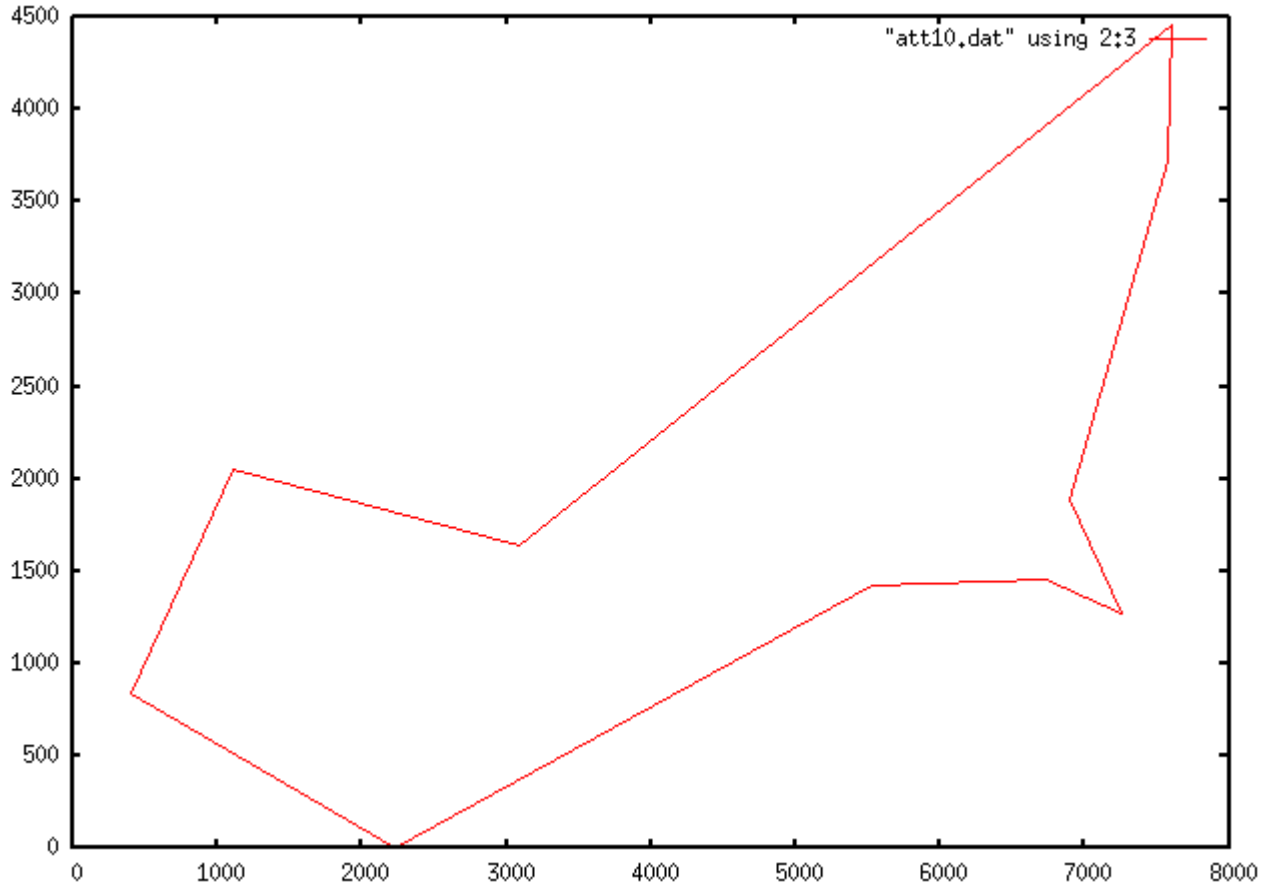
```
def TSP {
  nodosVivos; mejorNodo; min_distancias[];
  mejorNodo.ruta.añadirTodasCiudades;
  min_distancias = MinimosNodos();
  mejorNodo.mejores_aristas = sumar(min_distancias);
  nodosVivos.push_back(primerNodo);
  while(nodosVivos no esta vacio){
    actual = nodosVivos.top();  //LC: Priority Queue ordenada por un functor
    nodosVivos.pop();
    if(actual es el ultimo){
```

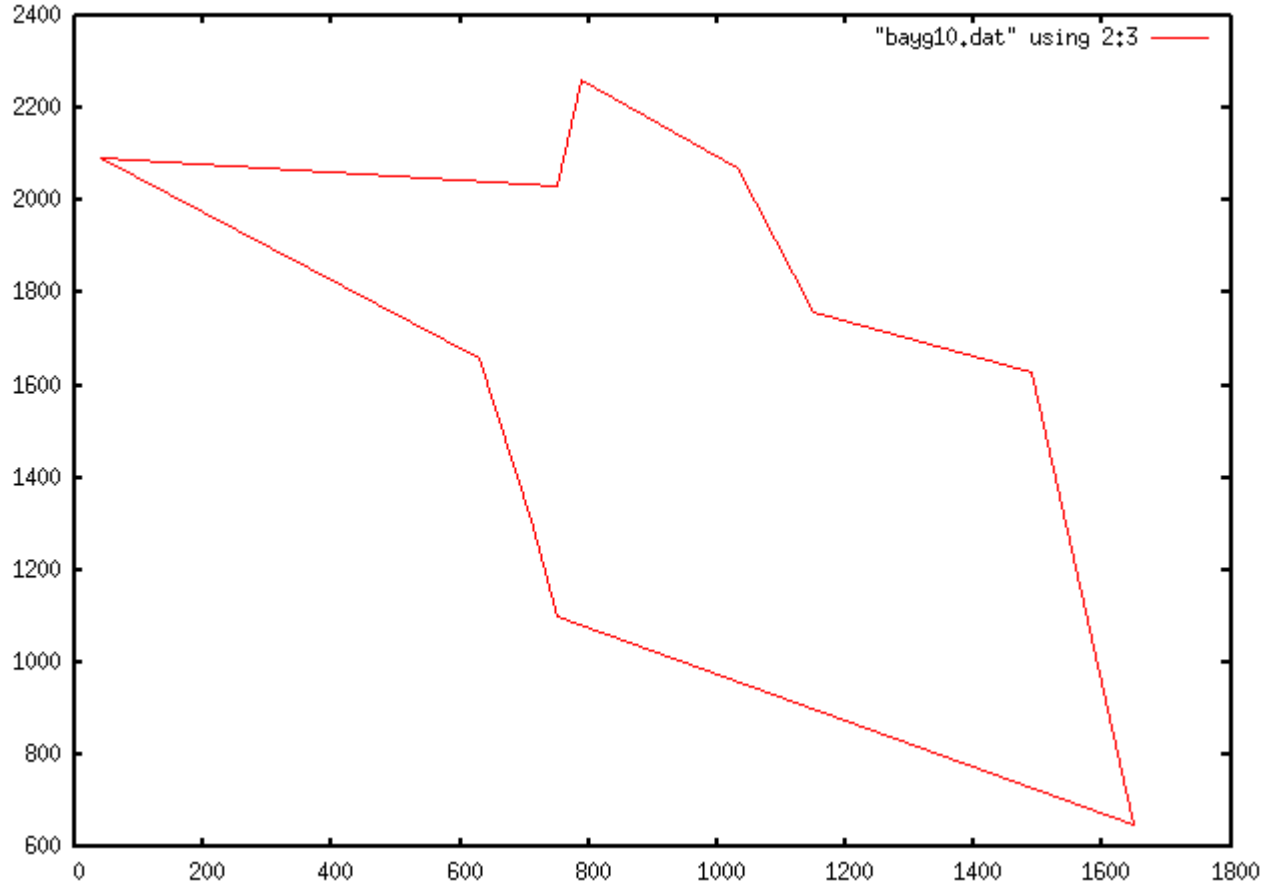
```
      coste = actual.coste + coste de cerrar el ciclo;
      if(coste < mejorNodo.coste){
        mejorNodo.ruta = actual.ruta; mejorNodo.coste = coste;
      }
    }
    else if(actual.costeTotal < mejorNodo.coste){
      para cada nodo [i] restante (desde actual hasta el final){
        para cada ciudad [j] en nodo.ruta{
          cruce = comprobarCruce(actual, ciudad)
        }

        if(!cruce){
          hijo = actual;
          swap(ciudad[i] de hijo, indice de mejor hijo);  // Selecciona el mejor hijo

          hijo.coste += coste de añadir el hijo;
          hijo.mejores_aristas -= min_distancias[hijo.indice - 1];  // Resta distancia minima
.                                                                   //porque es optima
          hijo.indice++;

          nodosVivos.push(hijo);
        }
      }
    }
    else{
      PODA (no toma ese camino)
    }
  }
  return mejorNodo;
}
```

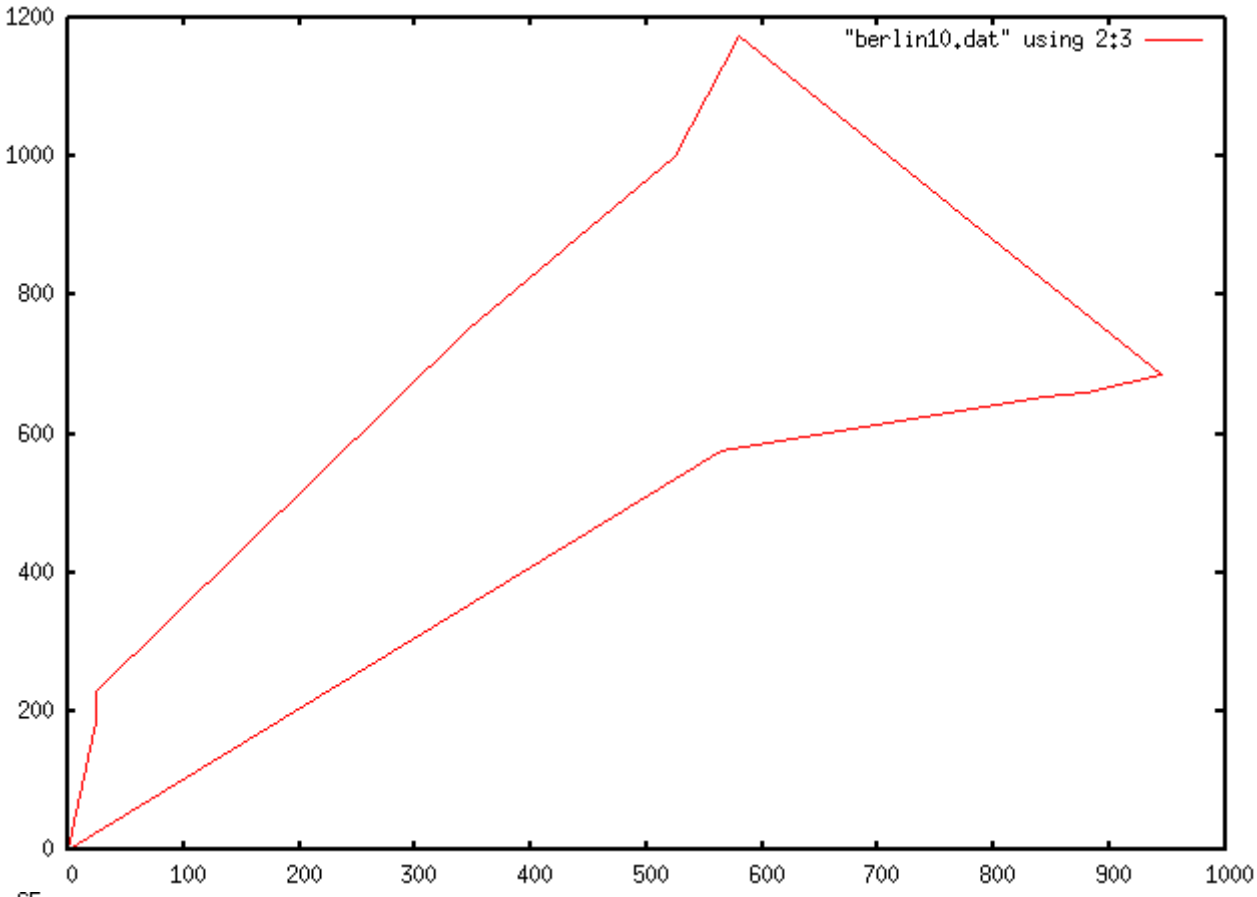
3. Datos obtenidos para distintos conjuntos de ciudades.

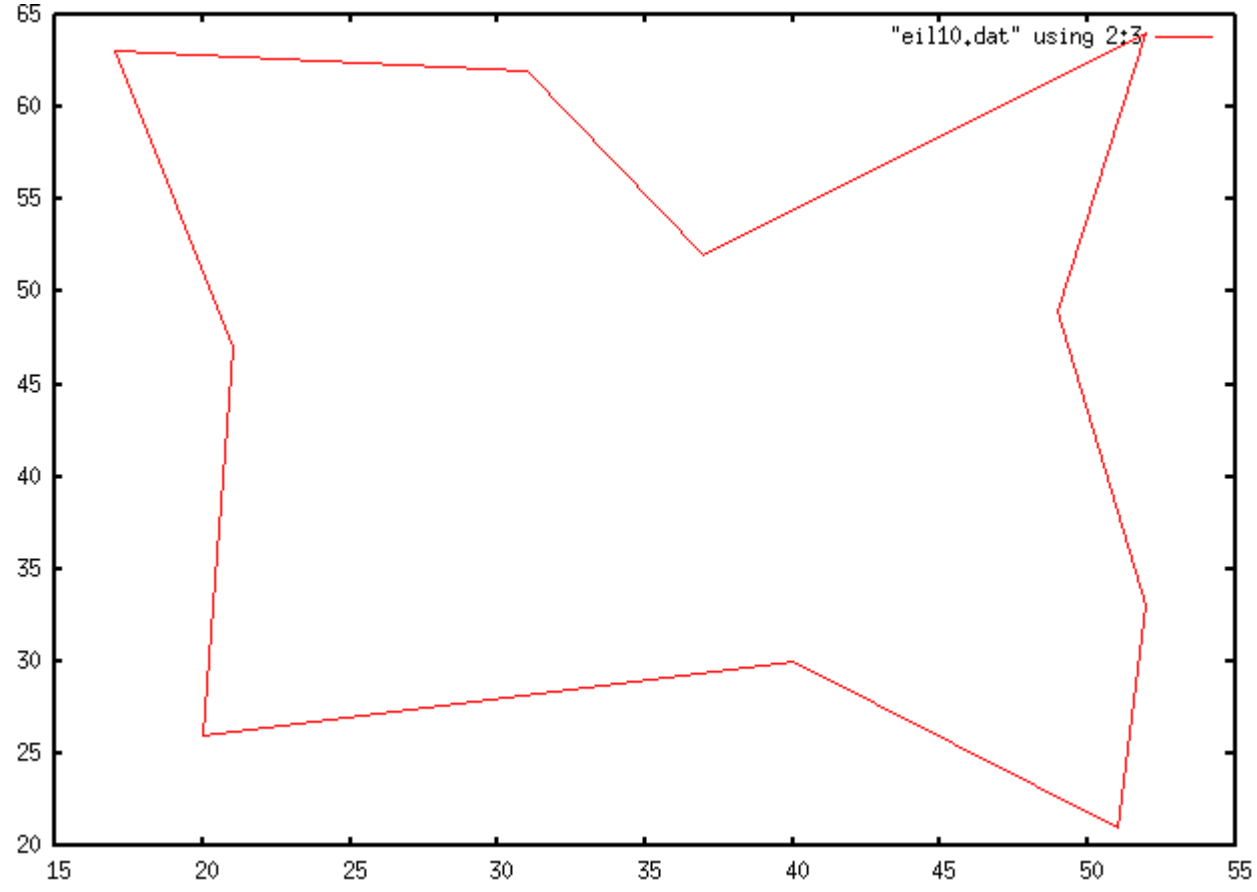
	att10	bayg10	berlin10	eil10	ulyses10
Coste	19520.3	5250.58	3147.09	160.649	46.5519
Nodos expandidos	2677	2396	2477	1669	2771
Tamaño máximo cola	18	41	22	24	18
Numero de podas	170	446	349	286	181
Tiempo	0.0256737	0.0163229	0.0145086	0.00877693	0.00990882

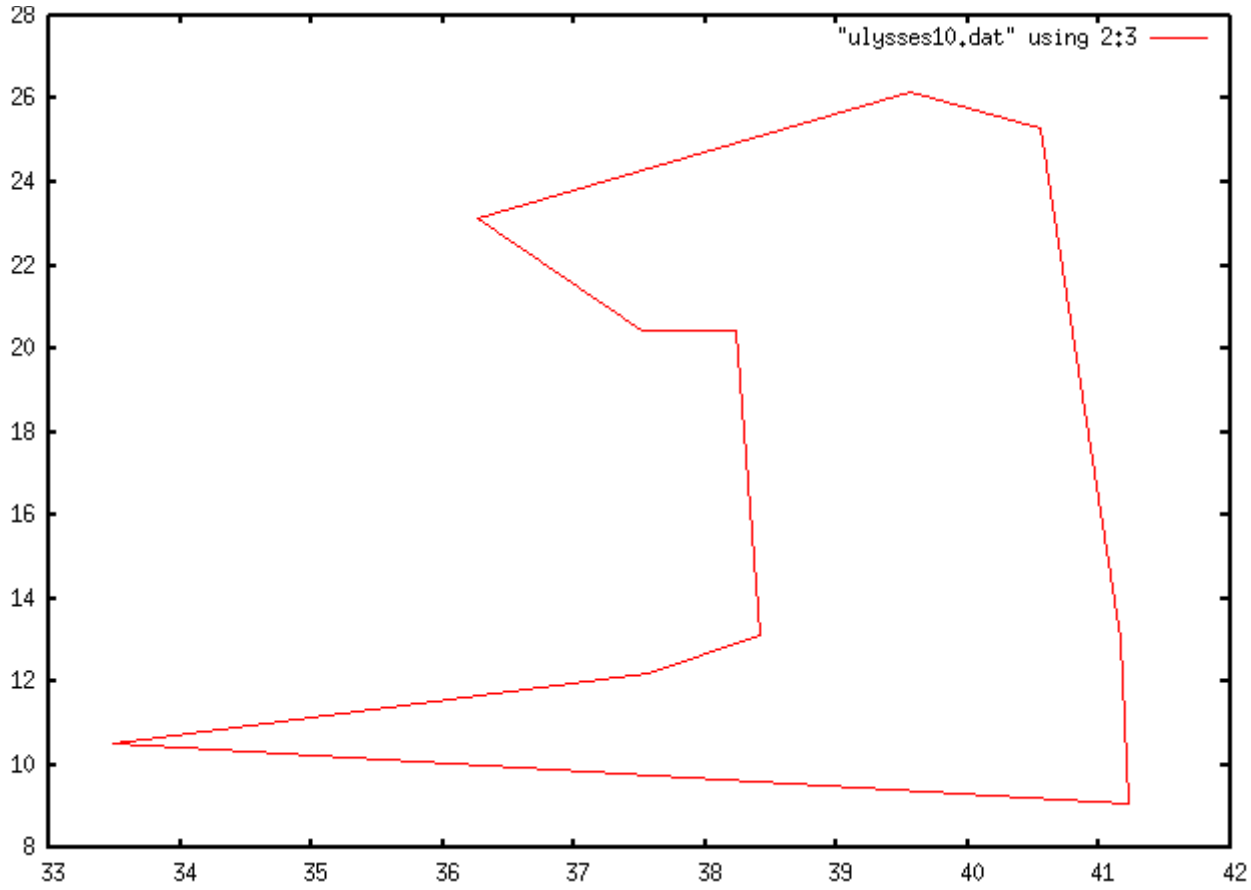
4. Gráficos de las rutas obtenidas.






"berlin10.dat" using 2:3
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
800
900
1000


"eil10.dat" using 2:3
65
60
55
50
45
40
35
30
25
20
15
20
25
30
35
40
45
50
55


"ulysses10.dat" using 2:3
28
26
24
22
20
18
16
14
12
10
8
33
34
35
36
37
38
39
40
41
42