UGR

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

Asignatura: Algorítmica

Práctica 3 (Segunda parte): TSP

Autores:
Míriam Mengíbar Rodríguez
Néstor Rodríguez Vico
Ángel Píñar Rivas

Granada, 30 de abril de 2016

Índice:

1. Heurística del vecino más cercano. 2

2. Heurística de la mejor inserción. 2

3. Heurística de la mejor arista. 2

4. Estudio empírico. 3

    4.1. Heurística del vecino más cercano. 3

    4.2. Heurística de la mejor inserción. 4

    4.3. Heurística de la mejor arista. 5

    4.4. Datos obtenidos. 6

    4.5. Ejemplo: “*berlin52”*. 6

1. Heurística del vecino más cercano.

El 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.

Los elementos de nuestro algoritmo Greedy son:

- Conjunto de candidatos: Ciudades sin visitar.
- Conjunto de seleccionados: Ciudad más cercana.
- Función solución: Se verifica que el numero de ciudades visitadas coincide con el numero de ciudades iniciales.
- Función selección: Función que, dada una ciudad, devuelve la ciudad más cercana.
- Función objetivo: Distancia total del recorrido que queremos optimizar.

2. Heurística de la mejor 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 esta heurística, los elementos del algoritmo coinciden con los descritos anteriormente, excepto la función de selección.
Para esta implementación del algoritmo que resuelve el problema TSP la función de selección es distinta. Dada una solución parcial, S_P, la función de selección obtiene una nueva ciudad, C, del conjunto de candidatas no usados, las ciudades sin visitar, y una nueva secuencia de ciudades visitadas, que inserta C en una posición de S_P, de forma que esta nueva secuencia minimiza la distancia del recorrido parcial.

Nuestra solución parcial inicial es una lista formada por 3 ciudades, las situadas más al norte, al sur y al oeste.

3. Heurística de la mejor arista.

La idea es hacer igual que en el algoritmo de Kruskal, el cual calcula un camino minimal entre dos nodos, pero garantizando que se forme un ciclo al final del proceso. Y de cada nodo no salgan más de dos aristas.

Los elementos de nuestro algoritmo Greedy son:

- Conjunto de candidatos: Aristas sin seleccionar.
- Conjunto de seleccionados: Arista de menor peso.
- Función solución: Se verifica que el numero de ciudades visitadas coincide con el numero de ciudades iniciales.
- Función selección: Función que devuelve la arista de menor peso.
- Función objetivo: Distancia total del recorrido que queremos optimizar.

4. Estudio empírico.

4.1. Heurística del vecino más cercano.

Comenzamos seleccionando la ciudad inicial como la 1. Recorremos la fila 1 y seleccionamos la ciudad a menor distancia (columnas). Cuando hacemos esto saltamos a la fila de la ciudad seleccionada y volvemos a realizar este procedimiento. Tenemos en cuenta que no podemos introducir ciudades repetidas.
Cuando haya introducido de la lista todas las ciudades sin repeticiones, añadimos a la distancia final el camino entre la última ciudad y la primera, e insertamos de nuevo la ciudad 1 (única que se repite).

```
vector<int> TSP(vector< vector<int> > matriz, int & distancia) {
   vector<int> recorrido;
   int ciudad, min, pos, j;
   ciudad = 0;
   min = 0;
   pos = 0;
   j = 0;

   recorrido.push_back(1);
   for(int i = 1; i < matriz.size(); i++) {
      j = 1;
      while(matriz[ciudad][j] == -1)
         j++;

      min = matriz[ciudad][j];
      pos = j;
      for(j; j < matriz[ciudad].size(); j++)
         if(matriz[ciudad][j] != -1 && min > matriz[ciudad][j]) {
            min = matriz[ciudad][j];
            pos = j;
         }

      ciudad = pos;
      recorrido.push_back(ciudad+1);
      InvalidarColumna(matriz,ciudad);
      distancia += min;
   }

   distancia += matriz[ciudad][0];
   recorrido.push_back(1);
   return recorrido;
}
```

4.2. Heurística de la mejor inserción.

Primero creamos una matriz con la distancias entre las ciudades, utilizando el algoritmo de Euclides. Comenzamos seleccionando la ciudad inicial como la 1. Recorremos la fila 1 y seleccionamos la ciudad a menor distancia (columnas). Cuando hacemos esto saltamos a la fila de la ciudad seleccionada y volvemos a realizar este procedimiento. Tenemos en cuenta que no podemos introducir ciudades repetidas. Cuando haya introducido de la lista todas las ciudades sin repeticiones, añadimos a la distancia final el camino entre la última ciudad y la primera, e insertamos de nuevo la ciudad 1 (única que se repite).

```
list<int> TSP(const vector<vector<int> > &matriz, const map<int, pair<double,
double> >  &m, int &distancia_final) {
    int n=1, s=1, w=1;

    list<int> salida;
    map<int,pair<double,double> >::const_iterator it = m.begin();
    list<int> ciudades_restantes;
    ciudades_restantes.push_back(1);

    pair<double,double> pn=(*it).second, ps=(*it).second, pw=(*it).second;
    it++;

    for(it; it != m.end(); it++) {
        ciudades_restantes.push_back((*it).first);
        pair<double,double> p=(*it).second;
        int ciudad_actual=(*it).first;

        if(p.first<pw.first) { //Elige la de mas al oeste
            pw=p;
            w=ciudad_actual;
        }
        else if(p.second>pn.second || (n==w && n==1)) { //La de mas al norte
            pn=p;
            n=ciudad_actual;
        }
        else if(p.second<ps.second || (s==w && s==1)) { //La de mas al sur
            ps=p;
            s=ciudad_actual;
        }
    }

    salida.push_back(n);
    salida.push_back(s);
    salida.push_back(w);

    ciudades_restantes.remove(n);
    ciudades_restantes.remove(s);
    ciudades_restantes.remove(w);

    while(ciudades_restantes.size() > 0) {
        pair<int,list<int>::iterator> pres;
        pres = ElegirCiudad(salida, matriz, ciudades_restantes);
        InsertarCiudad(salida, pres.first, pres.second);
    }
    salida.push_back(*(salida.begin())); //volvemos a la primera ciudad
    distancia_final = DistanciaTotal(salida, matriz);

    return salida;
}
```

4.3. Heurística de la mejor arista.

Creamos un map formado por la distancia entre dos ciudades y un par con esas dos ciudades. Vamos seleccionando las distancias menores entre dos ciudades y añadiendo estas al conjunto de seleccionados. Para poder insertar una nueva ciudad, esta no puede tener más de 2 aristas ni crear un ciclo. Cómo el camino resultante de este algoritmo no es cerrado, tenemos que recorrer el camino y añadir la distancia entre las 2 ciudades que únicamente tienen una arista.

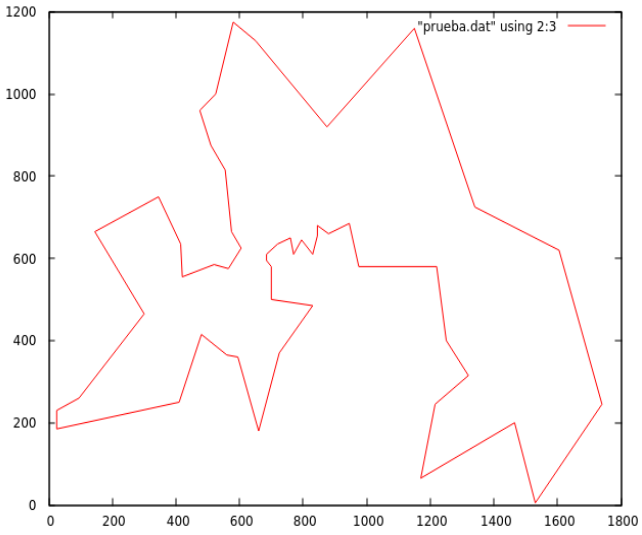
```
vector<int> TSP(multimap<int, pair<int,int> > &
aristas_par_ciudades, int & distancia) {

    vector<int> salida;
    list<pair<int,int> > aux;
    multimap<int , pair<int,int> >::iterator it =
aristas_par_ciudades.begin();

    aux.push_back((*it).second);
    distancia+=(*it).first;
    it++;
    while(it != aristas_par_ciudades.end()) {

       if(CiudadFactible(aux, (*it).second) && !ExistenCiclos(aux,
(*it).second)) {
            aux.push_back((*it).second);
            distancia += (*it).first;
        }
        it++;
    }

    CerrarCiclo(aux, aristas_par_ciudades, distancia);

    salida = TransformaACamino(aux);

    return salida;
}
```

4.4. Datos obtenidos.

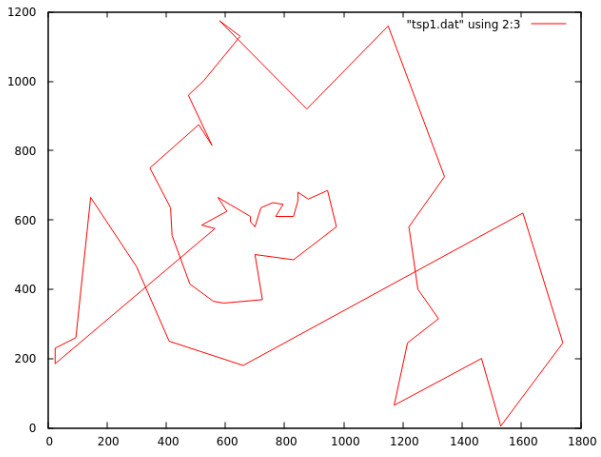
	tsp1	tsp2	tsp3
a280.tsp	3203	6525	3096
att48.tsp	40503	61949	40138
bayg29.tsp	10200	16450	9874
ch130.tsp	7511	13414	7127
eil51.tsp	497	581	480
gr96.tsp	615	1110	534
kroA100.tsp	27772	61605	24158
lin105.tsp	20341	37761	16748
pa561.tsp	18654	30809	17978
pcb442.tsp	61926	142878	61024

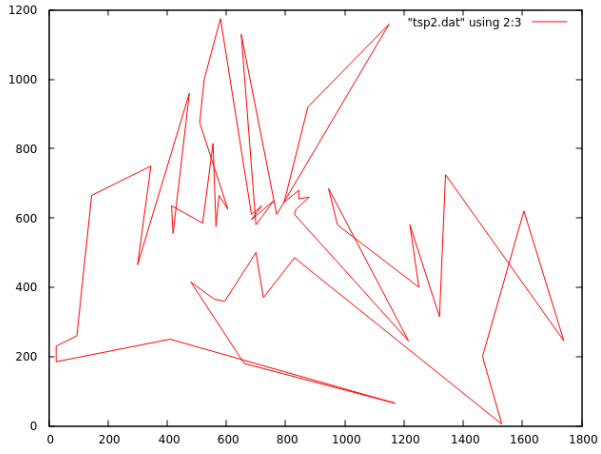
4.5. Ejemplo “*berlin52*”.

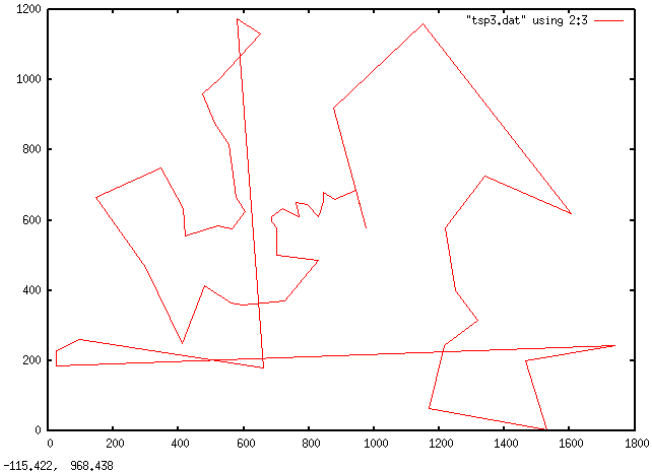
Ejecutamos el programa “*tsp*” y obtenemos el siguiente gráfico:



Y nuestros algoritmos generan los siguientes gráficos:







Como podemos observar, para el caso de "*berlin52*", el algoritmo más óptimo es el tsp1, es decir, el vecino más cercano, lo vemos al comparar los gráficos de ambos algoritmos. La distancia obtenida para tsp1, tsp2, y tsp3 es 8962, 14258 y 9937 respectivamente.