

Práctica 3- Segunda Parte: Travel Salesman Problem

Francisco Carrillo Pérez, Borja Cañavate Bordons,
Miguel Porcel Jiménez, Jose Manuel Rejón Santiago, Jose Arcos Aneas

3 de mayo de 2016

Índice

1. Introducción	3
2. Primera versión	3
2.1. Diseño del algoritmo	3
2.2. Pseudocódigo	3
3. Segunda versión	3
3.1. Diseño del algoritmo	4
3.2. Pseudocódigo	4
4. Tercera versión	4
4.1. Diseño del algoritmo	4
4.2. Pseudocódigo	5
5. Comparación	5

Índice de figuras

5.1. Primera versión	6
5.2. Segunda versión	6
5.3. Tercera versión	7

Índice de tablas

1. Introducción

- El objetivo de esta práctica es diseñar varios algoritmo Greedy resolviendo el problema del viajante de comercio o TSP

2. Primera versión

Aplicar algoritmo usando Greedy simple, buscaremos en cada momento la ciudad más cercana y esa ciudad la añadiremos a nuestro conjunto de soluciones.

2.1. Diseño del algoritmo

- **Conjunto de candidatos:** Conjunto de ciudades. (Conjunto **C**)
- **Conjunto de seleccionados:** Conjunto de ciudades ordenado. (Conjunto **S**)
- **Función solución:** Cuando el conjunto de candidatos esté vacío.
- **Función factibilidad:** En principio no hay función de factibilidad.
- **Función selección:** Se seleccionará el día más cercano no visitado.
- **Función objetivo:** Lista con las ciudades en el orden en el que hay que visitarlas.

2.2. Pseudocodigo

Require: Conjunto de ciudades C,S

```
x=0
S=C[0]
for i = 1 to len(C) do
    if C[i] MenorDistancia then
        x = C[i]
        S.add(x)
        C[i].erase
    end if
end for
return S
```

3. Segunda versión

Aplicar algoritmo usando Greedy con selección parcial, estableceremos un recorrido inicial seleccionando las 3 ciudades más alejadas (oeste,norte y este). A partir de ahí añadiremos las ciudades con respecto a los intervalos creados.

3.1. Diseño del algoritmo

- **Conjunto de candidatos:** Conjunto de ciudades. (Conjunto **C**)
- **Conjunto de seleccionados:** Conjunto de ciudades ordenado. (Conjunto **S**)
- **Función solución:** Cuando el conjunto de candidatos esté vacío.
- **Función factibilidad:** En principio no hay función de factibilidad.
- **Función selección:** Se seleccionará una ciudad y comprobaremos para cada una de las posiciones del circuito solución mitocondria el mejor lugar para insertarlo (la que ofrece menor distancia) .
- **Función objetivo:** Lista con las ciudades en el orden en el que hay que visitarlas.

3.2. Pseudocódigo

Require: Conjunto de ciudades C,S

```
x=0
S=CalcularRecorridoInicial()
for i=0 to len(C) do
    x = C[i]
    for j=0 to len(S) do
        x = C[i]
        MejorPosicion(S[j],x)
        C[i].erase
    end for
end for
return S
```

4. Tercera versión

Implementación del algoritmo Greedy 2-Opt. Calcularemos una primera aproximación de la solución utilizando el algoritmo de la primera versión. Después, iremos para cada ciudad intercambiando con las siguientes, viendo si alguno de esos intercambios mejora la primera solución obtenida, así hasta intentar generar una solución mejor que la primera

4.1. Diseño del algoritmo

- **Conjunto de candidatos:** Conjunto de ciudades. (Conjunto **C**)
- **Conjunto de seleccionados:** Conjunto de ciudades ordenado. (Conjunto **S**)
- **Función solución:** Cuando el conjunto de candidatos esté vacío.
- **Función factibilidad:** En principio no hay función de factibilidad.

- **Función selección:** Dada la solución con la primera versión del algoritmo, comprobaremos si intercambiando una por una las ciudades se mejora la solución inicial.
- **Función objetivo:** Lista con las ciudades en el orden en el que hay que visitarlas.

4.2. Pseudocódigo

Require: Conjunto de ciudades C,S,AUX

S=PrimeraVersión();Z=Distancia(S);

x,y,i = 0; j = 1;

while i menor o igual **tam -2** and **j** menor o igual a **tam - 1** **do**

Sp=Swap(S,i,j);Zp=Distancia(Sp);

if Zp mayor o igual que **z** and **j** menor que **tam-1** **then**

j++;

end if

if zp mayor o igual que **z** and **j** == **tam - 1** **then**

j += 1; i += 1

end if

if zp menor que **z** **then**

S = Sp; Z = Zp;

i += i; j = i + 1;

end if

end while

return S

5. Comparación

En esta sección veremos los distintos resultados obtenidos de cada una de las ejecuciones de los algoritmos.

Mapa	Versión I	Versión II	Versión III	Versión óptima
Berlin52	8314.81	9622.17	7883.92	7542
Ch130	7022.12	7499.39	6738.84	6110
Gr666	25345.7	27646.1	24163.5	21282

En esta sección veremos los distintos grafos obtenidos de cada uno de los algoritmos para el mapa Berlin52

En esta sección veremos los distintos grafos obtenidos de cada uno de los algoritmos para el mapa Berlin52

En esta sección veremos los distintos grafos obtenidos de cada uno de los algoritmos para el mapa Berlin52

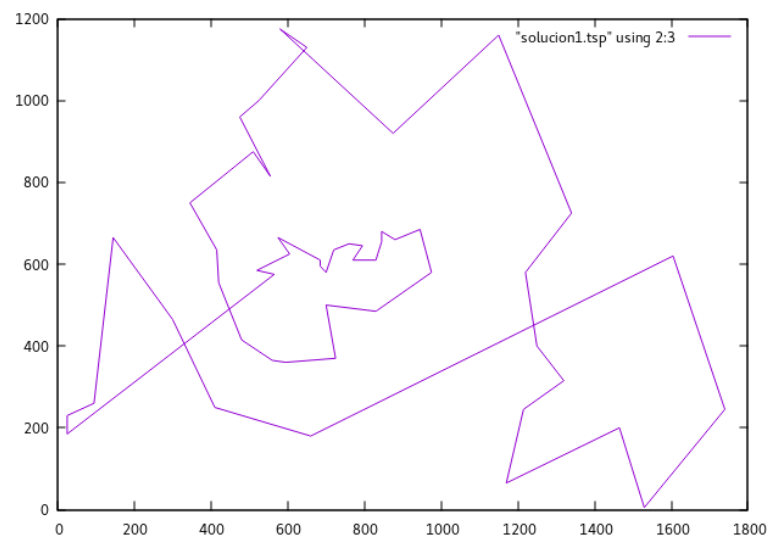


Figura 5.1: Primera versión

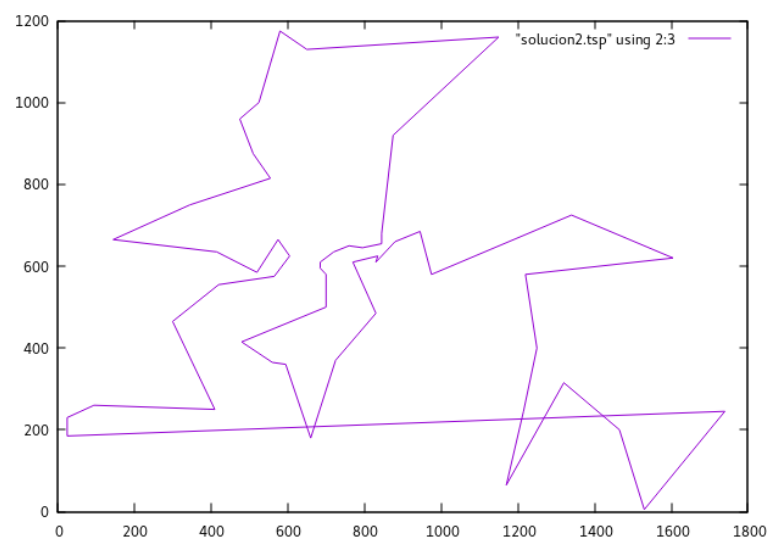


Figura 5.2: Segunda versión

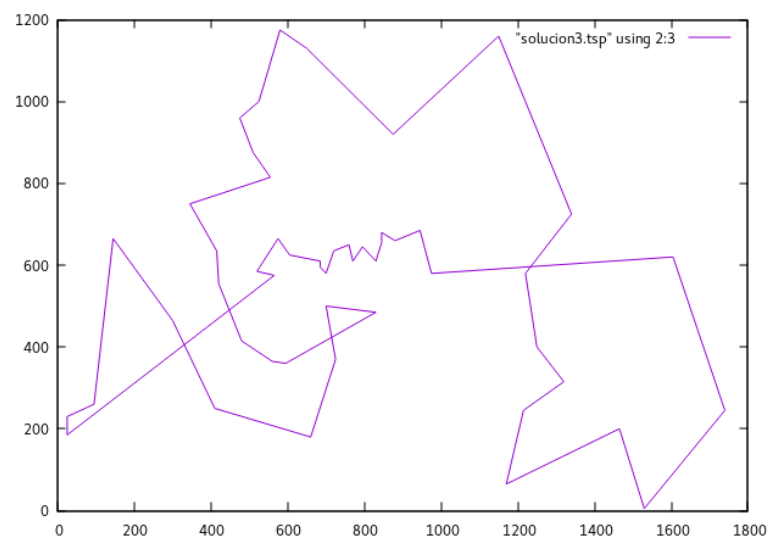


Figura 5.3: Tercera versión