



ugr

Universidad
de Granada

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



DECSAI

Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

CURSO 2017-2018

Índice

1. Descripción de la práctica	1
2. Vecino más cercano	1
3. Inserción más económica	1
4. Derivado de Kruskal	1
5. Comparación de estrategias	2
6. Anexo: código fuente	10

Índice de figuras

1. Comparación de resultados.	2
2. <i>ulysses16.tsp</i>	3
3. <i>gr96.tsp</i>	3
4. <i>a280.tsp</i>	4
5. <i>att48.tsp</i>	5
6. <i>pa561.tsp</i>	6
7. <i>lin105.tsp</i>	6
8. <i>pr76.tsp</i>	7
9. <i>tsp225.tsp</i>	7
10. <i>ch130.tsp</i>	8
11. <i>berlin52.tsp</i>	8
12. <i>rd100.tsp</i>	9
13. <i>st70.tsp</i>	9
14. Programa que calcula el orden según las distintas heurísticas	17
15. Programa que calcula la distancia del circuito a partir de un fichero con coordenadas y una lista ordenada de ciudades con sus respectivas coordenadas.	21
16. Script que genera archivos de coordenadas a partir de la mejor opción dada como lista.	21
17. Script que genera archivos de coordenadas a partir de las tres heurísticas desarrolladas.	22
18. Script que genera las gráficas de los problemas a partir de archivos de coordenadas.	25
19. Script genera todo lo necesario para la práctica.	26

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 tres heurísticas diferentes:

1. **Vecino más cercano.**
2. **Inserción más económica.**
3. **Derivado de Kruskal.**

Todas las heurísticas desarrolladas tienen varias características en común:

- **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.

Por tanto, será la función de selección la que diferencie una de otra.

2. Vecino más cercano

- **Función de selección.** Seleccionaremos aquella ciudad cuya distancia euclídea sea menor con respecto a la última ciudad añadida al conjunto de seleccionados.

Para abarcar más posibilidades, ejecutaremos el algoritmo voraz teniendo en cuenta todas las posibles ciudades de inicio, quedándonos con la distancia total más pequeña.

3. Inserción más económica

- **Función de selección.** Seleccionamos la ciudad que incremente mínimamente la distancia total del circuito.

El conjunto de seleccionados comienza inicializado por el circuito formado por las ciudades más al norte, más al este y más al oeste.

4. Derivado de Kruskal

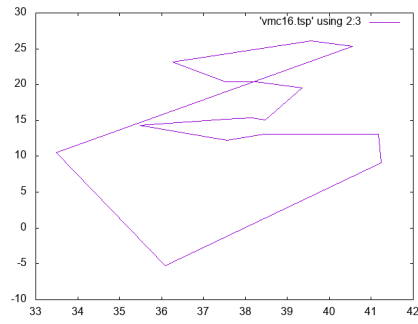
- **Función de selección.** Elegiremos aquella arista cuyo coste sea menor y cuyas ciudades no hayan sido visitadas aún.

Si el número de ciudades es impar, forzosamente tendremos que añadir la ciudad faltante al final, justo antes de cerrar el circuito.

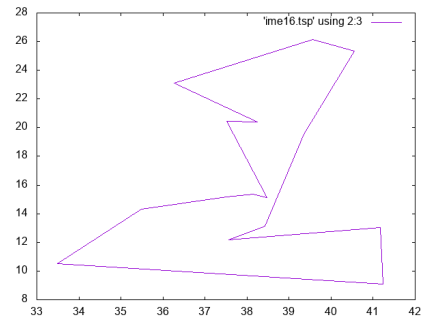
5. Comparación de estrategias

<i>ulysses16.tsp</i>		<i>gr96.tsp</i>	
Vecino más cercano	77.1269	Vecino más cercano	603.302
Inserción más económica	51.9733	Inserción más económica	620.367
Derivado de Kruskal		Derivado de Kruskal	
Solución más óptima	74.1087	Solución más óptima	512.309
<i>a280.tsp</i>		<i>att48.tsp</i>	
Vecino más cercano	3094.28	Vecino más cercano	39236.9
Inserción más económica	3192.42	Inserción más económica	40341.7
Derivado de Kruskal		Derivado de Kruskal	
Solución más óptima	2586.77	Solución más óptima	33523.7
<i>pa561.tsp</i>		<i>lin105.tsp</i>	
Vecino más cercano	18347	Vecino más cercano	16939.4
Inserción más económica	17688.4	Inserción más económica	19063.7
Derivado de Kruskal		Derivado de Kruskal	
Solución más óptima	19330.8	Solución más óptima	14383
<i>pr76.tsp</i>		<i>tsp225.tsp</i>	
Vecino más cercano	130921	Vecino más cercano	4633.2
Inserción más económica	136636	Inserción más económica	4734.51
Derivado de Kruskal		Derivado de Kruskal	
Solución más óptima	108159	Solución más óptima	3859
<i>ch130.tsp</i>		<i>berlin52.tsp</i>	
Vecino más cercano	7198.74	Vecino más cercano	8182.19
Inserción más económica	7455.72	Inserción más económica	9064.04
Derivado de Kruskal		Derivado de Kruskal	
Solución más óptima	6110.86	Solución más óptima	7544.37
<i>rd100.tsp</i>		<i>st70.tsp</i>	
Vecino más cercano	9427.33	Vecino más cercano	761.689
Inserción más económica	9655.6	Inserción más económica	824.228
Derivado de Kruskal		Derivado de Kruskal	
Solución más óptima	9724.75	Solución más óptima	678.597

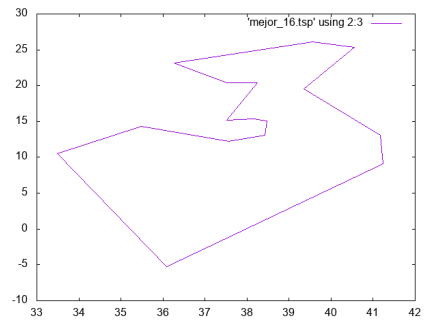
Figura 1: Comparación de resultados.



(a) Vecino más cercano



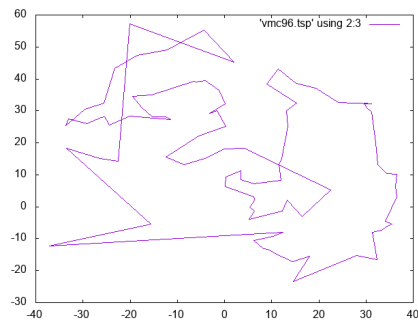
(b) Inserción más económica



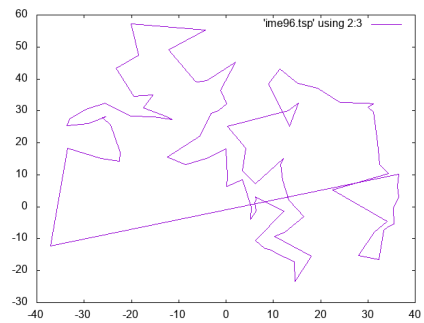
(c) Derivado de Kruskal

(d) Versión óptima

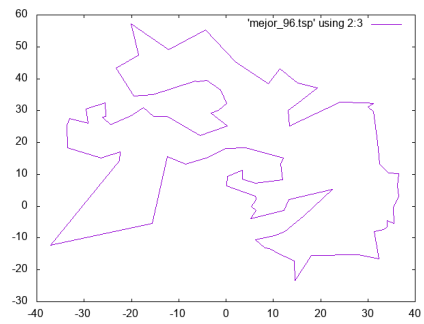
Figura 2: *ulysses16.tsp*



(a) Vecino más cercano



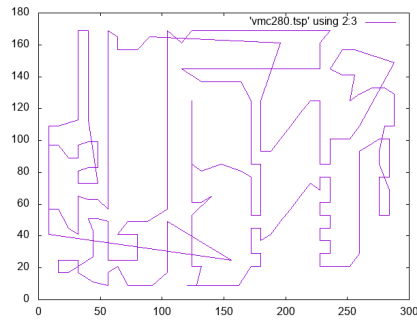
(b) Inserción más económica



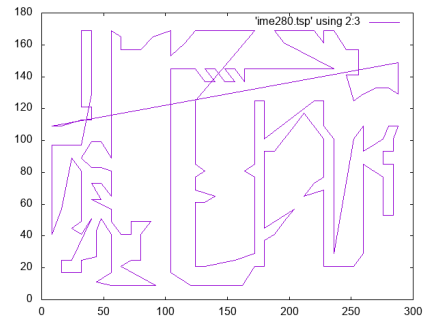
(c) Derivado de Kruskal

(d) Versión óptima

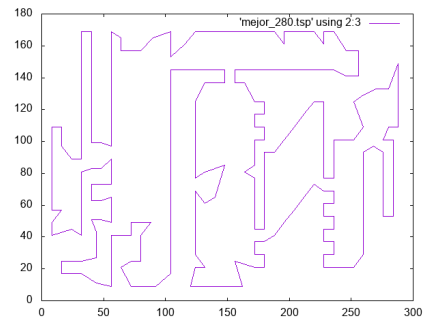
Figura 3: *gr96.tsp*



(a) Vecino más cercano



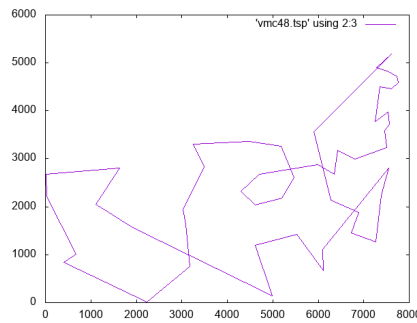
(b) Inserción más económica



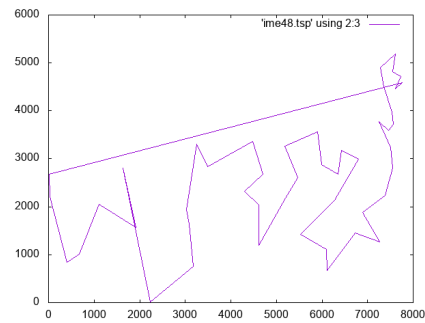
(c) Derivado de Kruskal

(d) Versión óptima

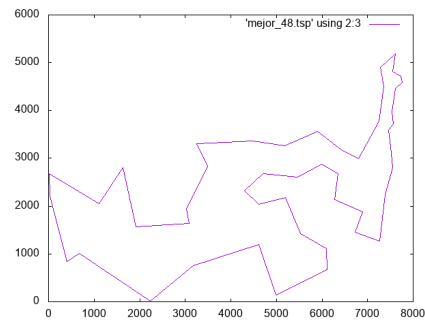
Figura 4: *a280.tsp*



(a) Vecino más cercano



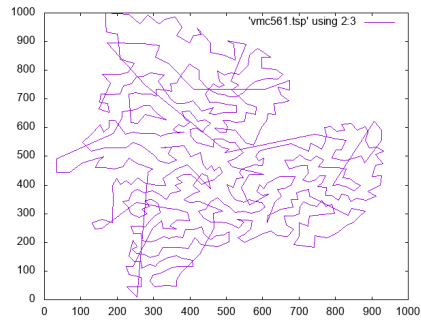
(b) Inserción más económica



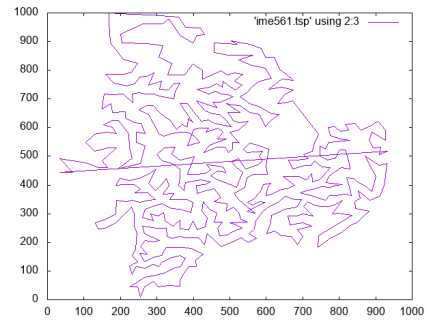
(c) Derivado de Kruskal

(d) Versión óptima

Figura 5: *att48.tsp*

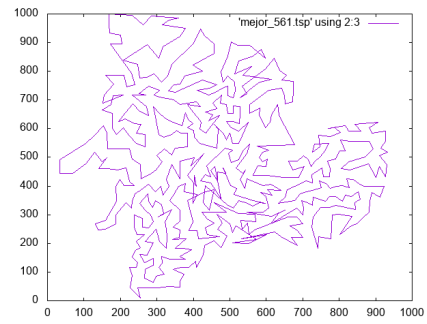


(a) Vecino más cercano



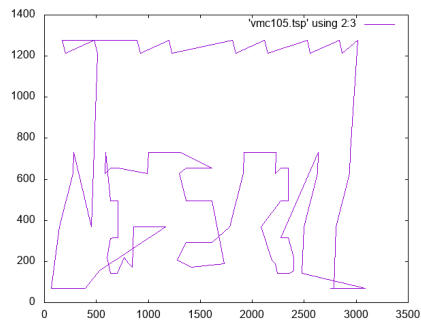
(b) Inserción más económica

(c) Derivado de Kruskal

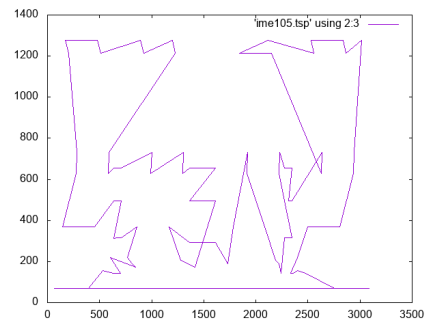


(d) Versión óptima

Figura 6: *pa561.tsp*

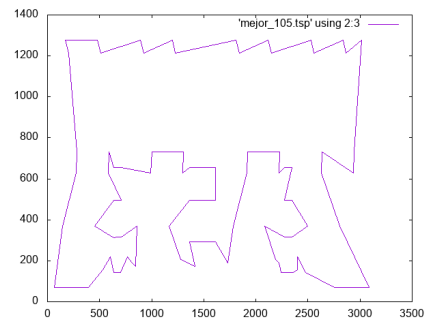


(a) Vecino más cercano



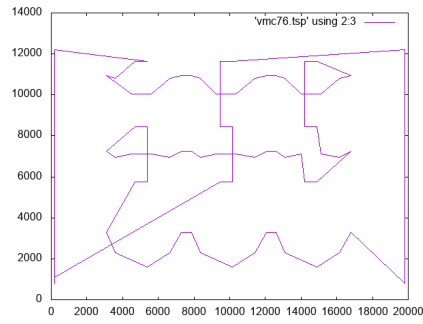
(b) Inserción más económica

(c) Derivado de Kruskal

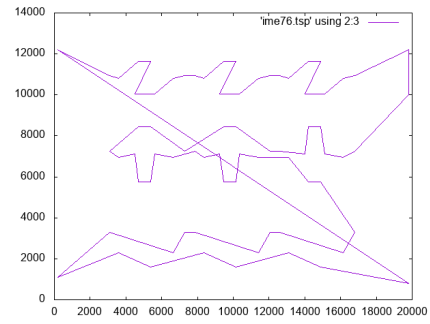


(d) Versión óptima

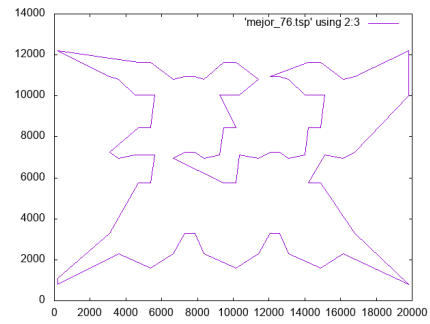
Figura 7: *lin105.tsp*



(a) Vecino más cercano



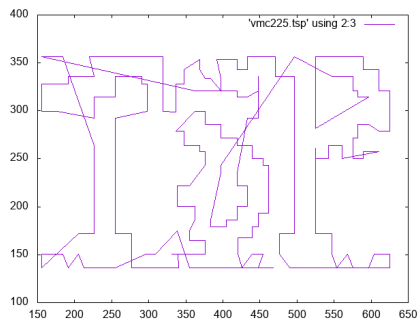
(b) Inserción más económica



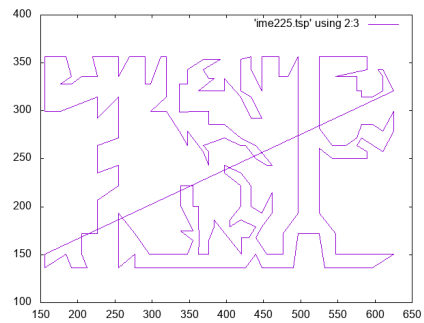
(c) Derivado de Kruskal

(d) Versión óptima

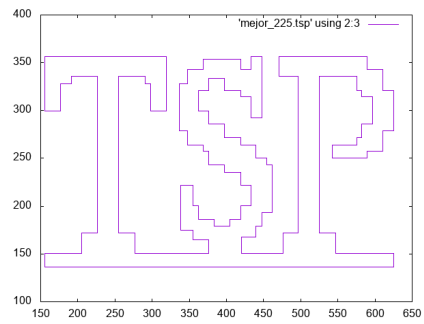
Figura 8: *pr76.tsp*



(a) Vecino más cercano



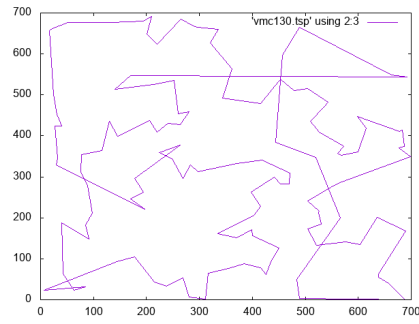
(b) Inserción más económica



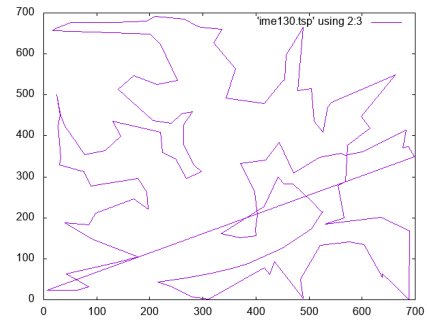
(c) Derivado de Kruskal

(d) Versión óptima

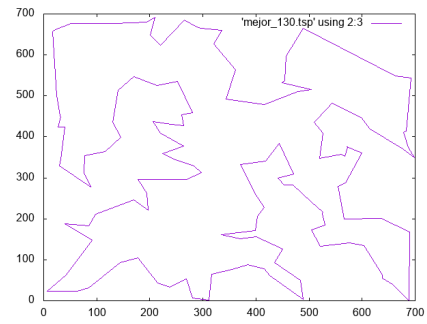
Figura 9: *tsp225.tsp*



(a) Vecino más cercano



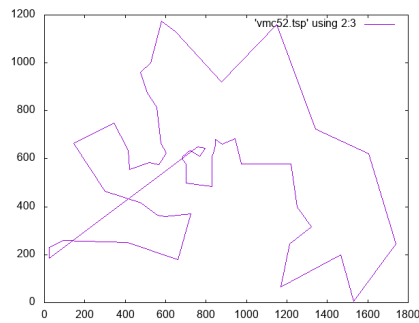
(b) Inserción más económica



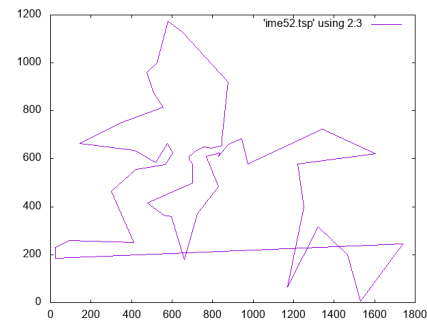
(c) Derivado de Kruskal

(d) Versión óptima

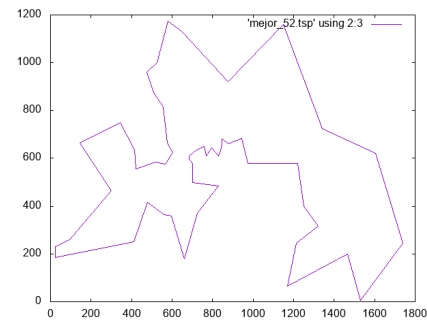
Figura 10: *ch130.tsp*



(a) Vecino más cercano



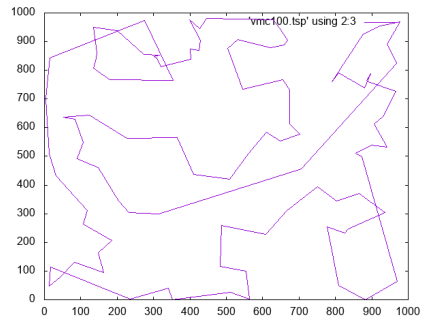
(b) Inserción más económica



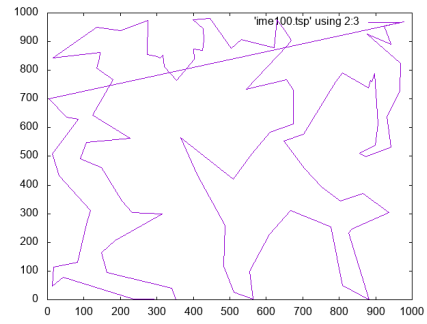
(c) Derivado de Kruskal

(d) Versión óptima

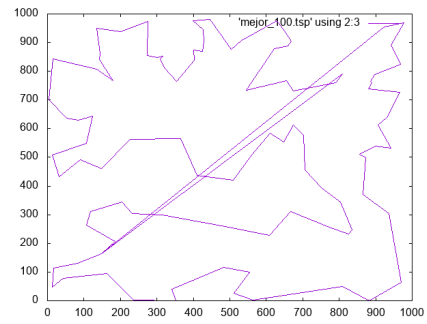
Figura 11: *berlin52.tsp*



(a) Vecino más cercano



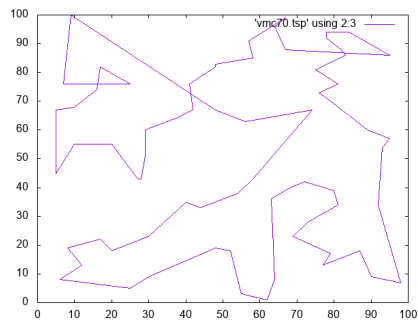
(b) Inserción más económica



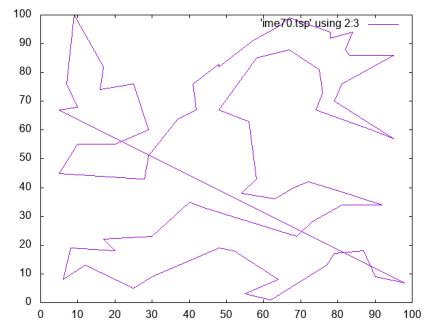
(c) Derivado de Kruskal

(d) Versión óptima

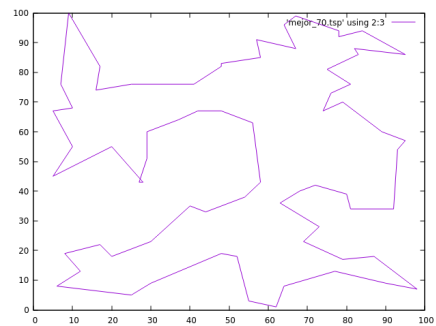
Figura 12: *rd100.tsp*



(a) Vecino más cercano



(b) Inserción más económica



(c) Derivado de Kruskal

(d) Versión óptima

Figura 13: *st70.tsp*

6. Anexo: código fuente

```
1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <climits>
5  #include <cmath>
6  #include <string>
7  #include <limits>
8  using namespace std;
9
10 #define TEST 0
11
12 struct ciudad {
13     int n;
14     double x;
15     double y;
16 };
17
18 bool operator==(const ciudad &una, const ciudad &otra) {
19     return una.x == otra.x && una.y == otra.y;
20 }
21 bool operator!=(const ciudad &una, const ciudad &otra) {
22     return !(una == otra);
23 }
24
25 ostream& operator<<(ostream &flujo, const vector<ciudad> &v) {
26     for (auto it = v.begin(); it != v.end(); ++it) {
27         if (it != v.begin())
28             flujo << "->";
29         flujo << it->n ;
30     }
31     return flujo;
32 }
33
34 class TSP {
35
36 private:
37     vector<ciudad> ciudades;
38     double distancia_total;
39     vector<ciudad> camino;
40     vector<vector<double>> matriz_distancias;
41     vector<bool> visitados;
42
43     double calcularDistanciaCamino(const vector<ciudad> &path) {
44         double distancia = 0;
45         for (int i = 0, j = 1; j < path.size(); i++ , j++)
46             distancia += distanciaEuclidea(path[i], path[j]);
```

```

47     return distancia;
48 }
49
50 double distanciaEuclidea(const ciudad &una, const ciudad &otra) {
51     double resultado;
52     if (una == otra)
53         resultado = 0;
54     else
55         resultado = sqrt(pow(una.x - otra.x, 2) + pow(una.y - otra.y, 2));
56     return resultado;
57 }
58
59 int CiudadMasCercana(ciudad actual) {
60     double distancia_minima = INFINITY;
61     int ciudad = -1;
62     for (int i = 0; i < ciudades.size(); i++) {
63         if (matriz_distancias[actual.n][i] < distancia_minima &&
64             ↪ !visitados[i]) {
65             distancia_minima = matriz_distancias[actual.n][i];
66             ciudad = i;
67         }
68     }
69     return ciudad;
70 }
71
72 pair<vector<ciudad>, double> VecinoMasCercanoParcial(int inicial) {
73     vector<ciudad> resultado;
74     ciudad actual = ciudades[inicial];
75     ciudad siguiente;
76     bool fin = false;
77     resultado.push_back(actual);
78     while (!fin) {
79         visitados[actual.n] = true;           //Pongo como visitados
80         int indice_siguiente = CiudadMasCercana(actual); //Busco el indice
81         ↪ de la siguiente ciudad
82         if (indice_siguiente != -1)           // Si he recorrido todas
83         ↪ las ciudades, añado la primera.
84             siguiente = ciudades[indice_siguiente];
85         else {
86             fin = true;
87             siguiente = resultado[0]; //Volvemos al inicio
88         }
89         resultado.push_back(siguiente);
90         actual = siguiente;
91     }
92     double distancia = calcularDistanciaCamino(resultado);
93     pair<vector<ciudad>, double> par;
94     par.first = resultado;

```

```

92     par.second = distancia;
93     return par;
94 }
95
96 void InicializarMatrizDistancias() {
97     for (int i = 0; i < ciudades.size(); i++)
98         for (int j = 0; j < ciudades.size(); j++)
99             matriz_distancias[i][j] = distanciaEuclidea(ciudades[i],
100                 ↪ ciudades[j]);
101 }
102
103 void Reservar(int n) {
104     visitados.resize(n);
105     matriz_distancias.resize(n);
106     for (int i = 0; i < n; i++)
107         matriz_distancias[i].resize(n);
108 }
109
110 void ResetVisitados() {
111     for (auto it = visitados.begin(); it != visitados.end(); ++it)
112         *it = false;
113 }
114
115 void Triangularizar(ciudad & norte, ciudad & este,
116     ciudad & oeste) {
117
118     double x_max, x_min, y_max;
119     x_max = numeric_limits<double>::min();
120     y_max = numeric_limits<double>::min();
121     x_min = numeric_limits<double>::max();
122
123     for (int i = 0; i < ciudades.size(); i++) {
124         if (ciudades[i].x > x_max) { //Más al este
125             x_max = ciudades[i].x;
126             este = ciudades[i];
127         }
128         if (ciudades[i].x < x_min) { //Más al oeste
129             x_min = ciudades[i].x;
130             oeste = ciudades[i];
131         }
132         if (ciudades[i].y > y_max) { //Más al norte
133             y_max = ciudades[i].y;
134             norte = ciudades[i];
135         }
136     }
137 }
138
139 void seleccionarNuevaCiudad() {

```

```

139     ciudad actual;
140     ciudad a_insertar;
141     int indice;
142     double dist_minima = numeric_limits<double>::max();
143     double d_aux;
144     for (int i = 0; i < ciudades.size(); i++) {    //Itero por todas las
        ↪ posibles ciudades
145         actual = ciudades[i];
146         if (!visitados[actual.n]) {                //Si no la he visitado
147             for (int j = 1; j < camino.size() - 1; j++) {    //Veo en que
                ↪ posición podría insertarla
148                 vector<ciudad> aux = camino;
149                 aux.insert(aux.begin() + j, actual);
150                 d_aux = calcularDistanciaCamino(aux);
151                 if (d_aux < dist_minima) {    //Me quedo con la que menos
                    ↪ incrementa la distancia
152                     dist_minima = d_aux;
153                     a_insertar = actual;
154                     indice = j;
155                 }
156             }
157         }
158     }
159     camino.insert(camino.begin() + indice, a_insertar);
160     visitados[a_insertar.n] = true;
161 }
162
163 public:
164     TSP() {
165         distancia_total = 0;
166         ResetVisitados();
167     }
168     TSP(char *archivo) {
169         CargarDatos(archivo);
170         distancia_total = 0;
171         ResetVisitados();
172     }
173     ~TSP() {
174
175     }
176
177     void VecinoMasCercano() {
178         pair<vector<ciudad>, double> minimo, temp;
179         minimo = VecinoMasCercanoParcial(0);    //Calculo el vecino más
            ↪ cercano comenzando por el primero
180         for (int i = 0; i < ciudades.size(); i++) {
181             ResetVisitados();
182             temp = VecinoMasCercanoParcial(i);

```

```

183 #if TEST
184     cout << temp.first << endl;
185     cout << "Distancia " << temp.second << endl;
186 #endif
187     if (temp.second < minimo.second)
188         minimo = temp;
189 }
190 camino = minimo.first;
191 distancia_total = minimo.second;
192 }
193
194
195 int GetTamanio() {
196     return ciudades.size();
197 }
198
199 void DerivadoKruskal() {
200
201 }
202
203 void CargarDatos(char *archivo) {
204     ifstream datos;
205     string s;
206     int n;
207     ciudad aux;
208     datos.open(archivo);
209     if (datos.is_open()) {
210         datos >> s; //Leo DIMENSIÓN (cabecera)
211         datos >> n; //Leo NÚMERO de ciudades y reservo espacio en
212             ↪ matrices y vector.
213         Reservar(n);
214
215         for (int i = 0; i < n; i++) {
216             datos >> aux.n; // Leo número de ciudad
217             aux.n--; //Decremento el número: los índices del archivo
218                 ↪ comienzan en 1. Los del vector en 0.
219             datos >> aux.x >> aux.y; //Leo coordenadas
220             ciudades.push_back(aux);
221         }
222         datos.close();
223     }
224     else
225         cout << "Error al leer " << archivo << endl;
226
227     InicializarMatrizDistancias();
228 }
229
230 void imprimirResultado() {

```

```

229     cout << endl << "Mejor solución:" << endl;
230     cout << camino << endl;
231     cout << "Distancia: " << distancia_total << endl;
232 }
233
234 void Exportar(const char *name) {
235     ofstream salida;
236     salida.open(name);
237     if (salida.is_open()) {
238         salida << "DIMENSION: ";
239         salida << ciudades.size() << endl;
240         salida << "DISTANCIA: " << distancia_total << endl;
241         for (auto it = camino.begin(); it != camino.end(); ++it) {
242             salida << it->n + 1 << " " << it->x << " " << it->y << endl;
243         }
244         salida.close();
245     }
246     else
247         cout << "Error al exportar." << endl;
248 }
249
250 void InsercionMasEconomica() {
251     camino.clear();
252     ciudad norte, este, oeste;
253     //Hallo triángulo inicial
254     Triangularizar(norte, este, oeste);
255     //Añado al camino
256     camino.push_back(oeste);
257     camino.push_back(norte);
258     camino.push_back(este);
259     camino.push_back(oeste);
260
261     visitados[norte.n] = true;
262     visitados[este.n] = true;
263     visitados[oeste.n] = true;
264
265     cout << "Circuito Parcial:" << endl;
266     imprimirResultado();
267
268
269     //Voy eligiendo la siguiente
270     while (camino.size() != ciudades.size()) //Mientras no recorramos
271         ↪ todas las ciudades
272         seleccionarNuevaCiudad();
273     //Añado el inicio
274     //camino.push_back(camino[0]);
275     distancia_total = calcularDistanciaCamino(camino);

```



```

276     }
277
278 };
279
280
281 int main(int argc, char **argv) {
282
283     if (argc != 2) {
284         cerr << "Error de formato: " << argv[0] << " <fichero>." << endl;
285         exit(-1);
286     }
287
288     string nombre = "";
289
290     /****** Vecino más cercano******/
291     cout << "*****" << endl;
292
293     TSP vecino_mas_cercano(argv[1]);
294     cout << "Heurística del Vecino más cercano." << endl;
295     vecino_mas_cercano.VecinoMasCercano();
296     vecino_mas_cercano.imprimirResultado();
297
298     nombre = "vmc";
299     nombre += to_string(vecino_mas_cercano.GetTamano());
300     nombre += ".tsp";
301     vecino_mas_cercano.Exportar(nombre.c_str());
302     cout << "Exportado archivo " << nombre << endl;
303
304     cout << "*****" << endl;
305
306
307     /****** Inserción más económica******/
308
309     cout << "*****" << endl;
310
311
312     TSP insercion_mas_economica(argv[1]);
313
314     cout << "Heurística de la inserción más económica." << endl;
315     insercion_mas_economica.InsercionMasEconomica();
316     insercion_mas_economica.imprimirResultado();
317
318     nombre = "ime";
319     nombre += to_string(insercion_mas_economica.GetTamano());
320     nombre += ".tsp";
321     insercion_mas_economica.Exportar(nombre.c_str());
322     cout << "Exportado archivo " << nombre << endl;
323

```

```

324     cout << "*****" << endl;
325
326
327
328     /****** Derivado de Kruskal******/
329
330     cout << "*****" << endl;
331
332
333     TSP derivado_kruskal(argv[1]);
334
335     cout << "Heurística derivada de Kruskal." << endl;
336     derivado_kruskal.DerivadoKruskal();
337     derivado_kruskal.imprimirResultado();
338
339     nombre = "kruskal";
340     nombre += to_string(derivado_kruskal.GetTamanio());
341     nombre += ".tsp";
342     derivado_kruskal.Exportar(nombre.c_str());
343     cout << "Exportado archivo " << nombre << endl;
344
345     cout << "*****" << endl;
346
347
348 }

```

Figura 14: Programa que calcula el orden según las distintas heurísticas

```

1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <climits>
5  #include <cmath>
6  #include <string>
7
8  using namespace std;
9
10
11 struct ciudad {
12     int n;
13     double x;
14     double y;
15 };
16
17 bool operator==(const ciudad &una, const ciudad &otra) {
18     return una.x == otra.x && una.y == otra.y;
19 }
20 bool operator!=(const ciudad &una, const ciudad &otra) {

```

```

21     return !(una == otra);
22 }
23
24 ostream& operator<<(ostream &flujo, const vector<ciudad> &v) {
25     for (auto it = v.begin(); it != v.end(); ++it) {
26         if (it != v.begin())
27             flujo << "->";
28         flujo << it->n ;
29     }
30     return flujo;
31 }
32
33
34 class TSP {
35
36 private:
37     vector<ciudad> ciudades;
38     double distancia_total;
39     vector<ciudad> camino;
40
41     double calcularDistanciaCamino(const vector<ciudad> &path) {
42         double distancia = 0;
43         for (int i = 0, j = 1; j < path.size(); i++ , j++)
44             distancia += distanciaEuclidea(path[i], path[j]);
45         return distancia;
46     }
47
48     double distanciaEuclidea(const ciudad &una, const ciudad &otra) {
49         double resultado;
50         if (una == otra)
51             resultado = 0;
52         else
53             resultado = sqrt(pow(una.x - otra.x, 2) + pow(una.y - otra.y, 2));
54         return resultado;
55     }
56
57
58 public:
59     TSP() {
60         distancia_total = 0;
61     }
62     TSP(char *archivo) {
63         CargarDatos(archivo);
64         distancia_total = 0;
65     }
66     ~TSP() {
67
68     }

```

```

69
70     int GetTamanio() {
71         return ciudades.size();
72     }
73
74
75     void SegunOrden(char *archivo) {
76         camino.clear();
77         ifstream datos;
78         string s;
79         int n;
80         int auxiliar;
81         datos.open(archivo);
82         if (datos.is_open()) {
83             datos >> s; //Leo cabecera
84             datos >> n; //Leo número ciudades
85             for (int i = 0; i < ciudades.size(); i++) {
86                 datos >> auxiliar; //Leo número de ciudad
87                 auxiliar--;
88                 bool fin = false;
89                 for (auto it = ciudades.begin(); it != ciudades.end() && !fin;
90                     ↪ ++it) { //Busco la ciudad y la inserto en el camino
91                     if (it->n == auxiliar) {
92                         camino.push_back(*it);
93                         fin = true;
94                     }
95                 }
96                 camino.push_back(camino[0]);
97                 distancia_total = calcularDistanciaCamino(camino);
98             }
99             else
100                 cout << "Error al leer" << archivo << endl;
101         }
102
103     int getDistancia(){
104         return distancia_total;
105     }
106
107     void CargarDatos(char *archivo) {
108         ifstream datos;
109         string s;
110         int n;
111         ciudad aux;
112         datos.open(archivo);
113         if (datos.is_open()) {
114             datos >> s; //Leo DIMENSIÓN (cabecera)
115             datos >> n; //Leo NÚMERO de ciudades .

```

```

116
117     for (int i = 0; i < n; i++) {
118         datos >> aux.n; // Leo número de ciudad
119         aux.n--; //Decremento el número: los índices del archivo
120                 ↪ comienzan en 1. Los del vector en 0.
121         datos >> aux.x >> aux.y; //Leo coordenadas
122         ciudades.push_back(aux);
123     }
124     datos.close();
125 }
126 else
127     cout << "Error al leer " << archivo << endl;
128 }
129
130 void Exportar(const char *name) {
131     ofstream salida;
132     salida.open(name);
133     if (salida.is_open()) {
134         salida << "DIMENSION: ";
135         salida << ciudades.size() << endl;
136         salida << "DISTANCIA: " << distancia_total << endl;
137         for (auto it = camino.begin(); it != camino.end(); ++it) {
138             salida << it->n + 1 << " " << it->x << " " << it->y << endl;
139         }
140         salida.close();
141     }
142     else
143         cout << "Error al exportar." << endl;
144 }
145 };
146
147 // Modo de uso: ./programa coordenadas orden
148 int main(int argc, char **argv) {
149
150     if (argc != 3) {
151         cerr << "Error de formato: " << argv[0] << " <coordenadas> <orden>."
152             ↪ << endl;
153         exit(-1);
154     }
155
156     string nombre = "";
157
158     /***** Vecino más cercano*****/
159     TSP instancia(argv[1]);
160
161

```

```

162
163     instancia.SegunOrden(argv[2]);
164
165     nombre = "mejor_";
166     nombre += to_string(instancia.GetTamano());
167     nombre += ".tsp";
168     instancia.Exportar(nombre.c_str());
169     cout << "Exportado archivo " << nombre << endl;
170
171     cout<<"DISTANCIA " <<instancia.getDistancia()<<endl;
172
173
174 }

```

Figura 15: Programa que calcula la distancia del circuito a partir de un fichero con coordenadas y una lista ordenada de ciudades con sus respectivas coordenadas.

```

1  #!/bin/bash
2
3  ./calcular_distancia ../datosTSP/a280.tsp ../datosTSP/a280.opt.tour
4  ./calcular_distancia ../datosTSP/att48.tsp ../datosTSP/att48.opt.tour
5  ./calcular_distancia ../datosTSP/berlin52.tsp
   ↪  ../datosTSP/berlin52.opt.tour
6  ./calcular_distancia ../datosTSP/ch130.tsp ../datosTSP/ch130.opt.tour
7  ./calcular_distancia ../datosTSP/gr96.tsp ../datosTSP/gr96.opt.tour
8  ./calcular_distancia ../datosTSP/lin105.tsp ../datosTSP/lin105.opt.tour
9  ./calcular_distancia ../datosTSP/pa561.tsp ../datosTSP/pa561.opt.tour
10 ./calcular_distancia ../datosTSP/st70.tsp ../datosTSP/st70.opt.tour
11 ./calcular_distancia ../datosTSP/pr76.tsp ../datosTSP/pr76.opt.tour
12 ./calcular_distancia ../datosTSP/rd100.tsp ../datosTSP/rd100.opt.tour
13 ./calcular_distancia ../datosTSP/tsp225.tsp ../datosTSP/tsp225.opt.tour
14 ./calcular_distancia ../datosTSP/ulysses16.tsp
   ↪  ../datosTSP/ulysses16.opt.tour

```

Figura 16: Script que genera archivos de coordenadas a partir de la mejor opción dada como lista.

```

1  #!/bin/bash
2
3  ./tsp ../datosTSP/a280.tsp
4  ./tsp ../datosTSP/att48.tsp
5  ./tsp ../datosTSP/berlin52.tsp
6  ./tsp ../datosTSP/ch130.tsp
7  ./tsp ../datosTSP/gr96.tsp
8  ./tsp ../datosTSP/lin105.tsp
9  ./tsp ../datosTSP/pa561.tsp
10 ./tsp ../datosTSP/st70.tsp
11 ./tsp ../datosTSP/pr76.tsp

```

```

12 ./tsp ../datosTSP/rd100.tsp
13 ./tsp ../datosTSP/tsp225.tsp
14 ./tsp ../datosTSP/ulysses16.tsp

```

Figura 17: Script que genera archivos de coordenadas a partir de las tres heurísticas desarrolladas.

```

1  #!/usr/bin/gnuplot
2
3
4  set terminal png size 640,480
5
6  set output 'a280_mejor.png'
7  plot 'mejor_280.tsp' using 2:3 with lines
8
9  set output 'a280_vmc.png'
10 plot 'vmc280.tsp' using 2:3 with lines
11
12 set output 'a280_ime.png'
13 plot 'ime280.tsp' using 2:3 with lines
14
15 #set output 'a280_kruskal.png'
16 #plot 'kruskal280.tsp' using 2:3 with lines
17
18
19
20 set output 'att48_mejor.png'
21 plot 'mejor_48.tsp' using 2:3 with lines
22
23 set output 'att48_vmc.png'
24 plot 'vmc48.tsp' using 2:3 with lines
25
26
27 set output 'att48_ime.png'
28 plot 'ime48.tsp' using 2:3 with lines
29
30 #set output 'att48_kruskal.png'
31 #plot 'kruskal48.tsp' using 2:3 with lines
32
33
34
35 set output 'berlin52_mejor.png'
36 plot 'mejor_52.tsp' using 2:3 with lines
37
38 set output 'berlin52_vmc.png'
39 plot 'vmc52.tsp' using 2:3 with lines
40
41 set output 'berlin52_ime.png'

```

```

42 plot 'ime52.tsp' using 2:3 with lines
43
44 #set output 'berlin52_kruskal.png'
45 #plot 'kruskal52.tsp' using 2:3 with lines
46
47
48
49 set output 'ch130_mejor.png'
50 plot 'mejor_130.tsp' using 2:3 with lines
51
52 set output 'ch130_vmc.png'
53 plot 'vmc130.tsp' using 2:3 with lines
54
55 set output 'ch130_ime.png'
56 plot 'ime130.tsp' using 2:3 with lines
57
58 #set output 'ch130_kruskal.png'
59 #plot 'kruskal130.tsp' using 2:3 with lines
60
61
62
63 set output 'gr96_mejor.png'
64 plot 'mejor_96.tsp' using 2:3 with lines
65
66 set output 'gr96_vmc.png'
67 plot 'vmc96.tsp' using 2:3 with lines
68
69 set output 'gr96_ime.png'
70 plot 'ime96.tsp' using 2:3 with lines
71
72 #set output 'gr96_kruskal.png'
73 #plot 'kruskal96.tsp' using 2:3 with lines
74
75
76
77
78 set output 'lin105_mejor.png'
79 plot 'mejor_105.tsp' using 2:3 with lines
80
81 set output 'lin105_vmc.png'
82 plot 'vmc105.tsp' using 2:3 with lines
83
84 set output 'lin105_ime.png'
85 plot 'ime105.tsp' using 2:3 with lines
86
87 #set output 'lin105_kruskal.png'
88 #plot 'kruskal105.tsp' using 2:3 with lines
89

```



```

90
91
92
93 set output 'pa561_mejor.png'
94 plot 'mejor_561.tsp' using 2:3 with lines
95
96 set output 'pa561_vmc.png'
97 plot 'vmc561.tsp' using 2:3 with lines
98
99
100 set output 'pa561_ime.png'
101 plot 'ime561.tsp' using 2:3 with lines
102
103 #set output 'pa561_kruskal.png'
104 #plot 'kruskal561.tsp' using 2:3 with lines
105
106
107
108
109 #set output 'st70_mejor.png'
110 #plot 'mejor_70.tsp' using 2:3 with lines
111
112 #set output 'st70_vmc.png'
113 #plot 'vmc70.tsp' using 2:3 with lines
114
115 #set output 'st70_ime.png'
116 #plot 'ime70.tsp' using 2:3 with lines
117
118 #set output 'st70_kruskal.png'
119 #plot 'kruskal70.tsp' using 2:3 with lines
120
121
122
123 set output 'pr76_mejor.png'
124 plot 'mejor_76.tsp' using 2:3 with lines
125
126 set output 'pr76_vmc.png'
127 plot 'vmc76.tsp' using 2:3 with lines
128
129 set output 'pr76_ime.png'
130 plot 'ime76.tsp' using 2:3 with lines
131
132 #set output 'pr76_kruskal.png'
133 #plot 'kruskal76.tsp' using 2:3 with lines
134
135
136
137 set output 'rd100_mejor.png'

```

```

138 plot 'mejor_100.tsp' using 2:3 with lines
139
140 set output 'rd100_vmc.png'
141 plot 'vmc100.tsp' using 2:3 with lines
142
143 set output 'rd100_ime.png'
144 plot 'ime100.tsp' using 2:3 with lines
145
146 #set output 'rd100_kruskal.png'
147 #plot 'kruskal100.tsp' using 2:3 with lines
148
149
150
151 set output 'tsp225_mejor.png'
152 plot 'mejor_225.tsp' using 2:3 with lines
153
154 set output 'tsp225_vmc.png'
155 plot 'vmc225.tsp' using 2:3 with lines
156
157 set output 'tsp225_ime.png'
158 plot 'ime225.tsp' using 2:3 with lines
159
160 #set output 'tsp225_kruskal.png'
161 #plot 'kruskal225.tsp' using 2:3 with lines
162
163
164
165
166 set output 'ulysses16_mejor.png'
167 plot 'mejor_16.tsp' using 2:3 with lines
168
169 set output 'ulysses16_vmc.png'
170 plot 'vmc16.tsp' using 2:3 with lines
171
172 set output 'ulysses16_ime.png'
173 plot 'ime16.tsp' using 2:3 with lines
174
175 #set output 'ulysses16_kruskal.png'
176 #plot 'kruskal16.tsp' using 2:3 with lines

```

Figura 18: Script que genera las gráficas de los problemas a partir de archivos de coordenadas.

```

1 #!/bin/bash
2
3 echo "Ejecutando tres heurísticas..."
4 ./ejecucion_tres_heuristicas.sh
5

```

```
6 echo "Ejecutando mejor opción..."
7 ./ejecucion_mejor_opcion.sh
8
9 echo "Ejecutando gnuplot..."
10 ./gnuplot.sh
```

Figura 19: Script genera todo lo necesario para la práctica.