



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

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 tres heurísticas diferentes:

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

2. Vecino más cercano

- **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 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

- **Conjunto de candidatos.** Ciudades por visitar.
- **Conjunto de seleccionados.** Inicialmente está formado por el triángulo de las ciudades más al norte, este y oeste.
- **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 de selección.** Seleccionamos la ciudad que incrementa minimamente la distancia total del circuito.

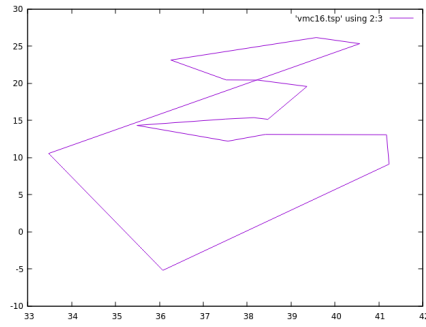
4. Derivado de Kruskal

- Conjunto de candidatos.
- Conjunto de seleccionados.
- Función solución.
- Función de factibilidad.
- Función de selección.

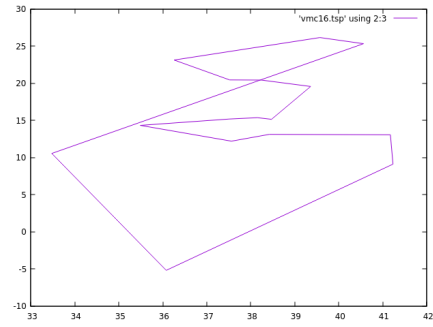
5. Comparación de estrategias

<i>ulysses16.tsp</i>		<i>gr96.tsp</i>	
Vecino más cercano	77.1269	Vecino más cercano	603.302
Inserción más económica		Inserción más económica	
Derivado de Kruskal		Derivado de Kruskal	
Solución más óptima	74	Solución más óptima	512
<i>a280.tsp</i>		<i>att48.tsp</i>	
Vecino más cercano	3094.28	Vecino más cercano	39236.9
Inserción más económica		Inserción más económica	
Derivado de Kruskal		Derivado de Kruskal	
Solución más óptima	2586	Solución más óptima	33523
<i>pa561.tsp</i>		<i>lin105.tsp</i>	
Vecino más cercano	18347	Vecino más cercano	16939.4
Inserción más económica		Inserción más económica	
Derivado de Kruskal		Derivado de Kruskal	
Solución más óptima	19330	Solución más óptima	14382
<i>pr76.tsp</i>		<i>tsp225.tsp</i>	
Vecino más cercano	130921	Vecino más cercano	4633.2
Inserción más económica		Inserción más económica	
Derivado de Kruskal		Derivado de Kruskal	
Solución más óptima	108159	Solución más óptima	14806
<i>ch130.tsp</i>		<i>berlin52.tsp</i>	
Vecino más cercano	7198.74	Vecino más cercano	8182.19
Inserción más económica		Inserción más económica	
Derivado de Kruskal		Derivado de Kruskal	
Solución más óptima	6110	Solución más óptima	7544
<i>rd100.tsp</i>		<i>pr2392.tsp</i>	
Vecino más cercano	9427.33	Vecino más cercano	458790
Inserción más económica		Inserción más económica	
Derivado de Kruskal		Derivado de Kruskal	
Solución más óptima	9724	Solución más óptima	50560

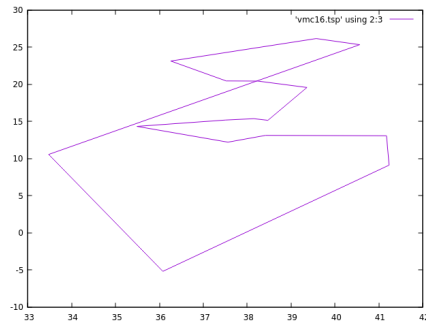
Figura 1: Comparación de resultados.



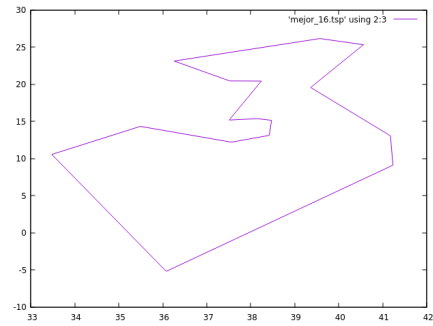
(a) Vecino más cercano



(b) Inserción más económica

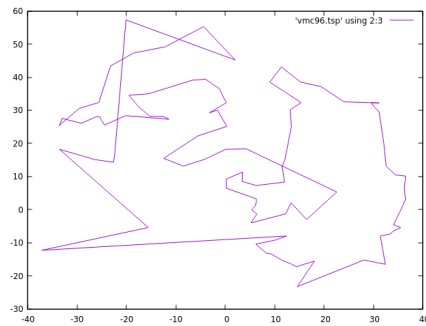


(c) Derivado de Kruskal

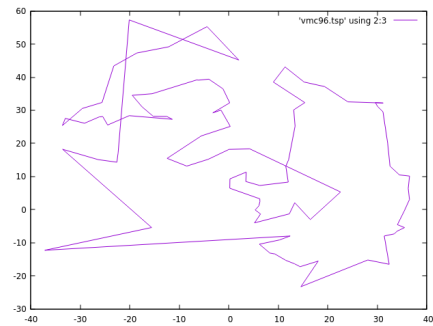


(d) Versión óptima

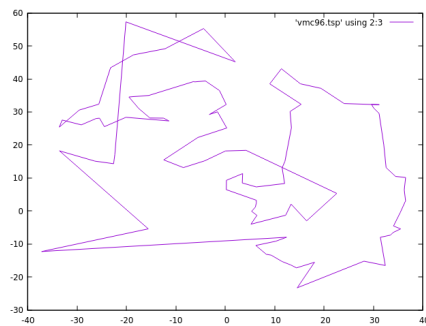
Figura 2: *ulysses16.tsp*



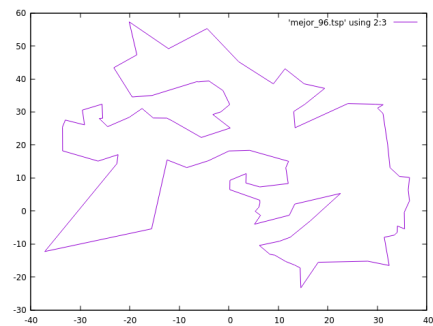
(a) Vecino más cercano



(b) Inserción más económica

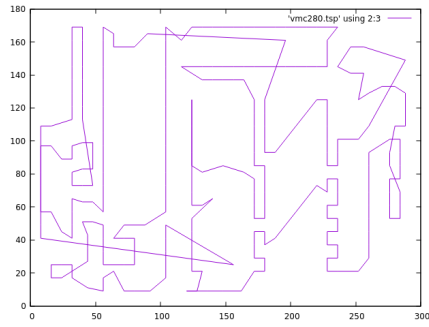


(c) Derivado de Kruskal

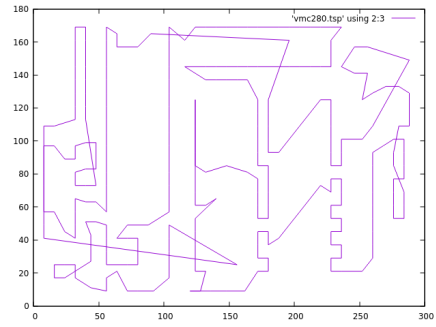


(d) Versión óptima

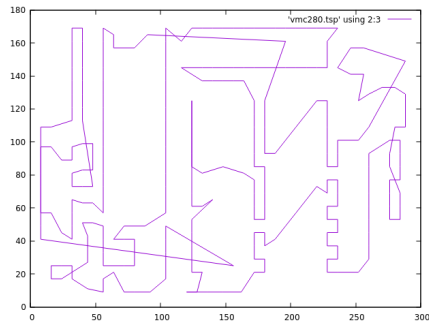
Figura 3: *gr96.tsp*



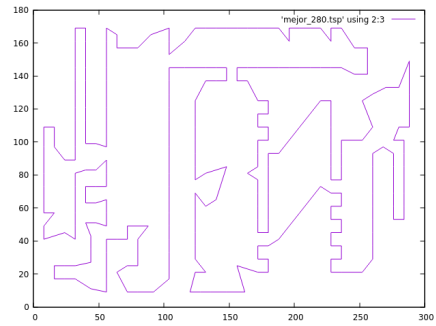
(a) Vecino más cercano



(b) Inserción más económica

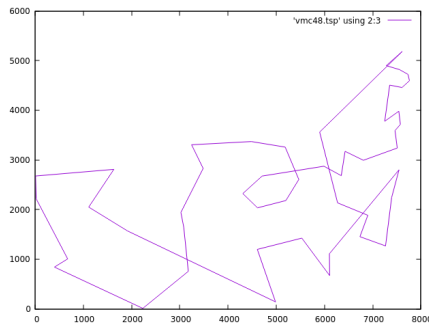


(c) Derivado de Kruskal

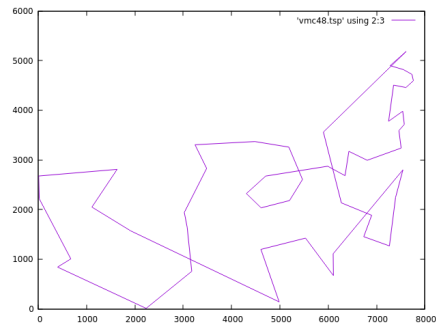


(d) Versión óptima

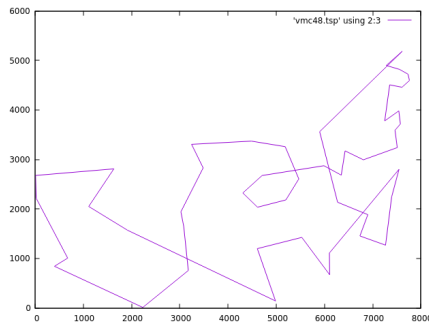
Figura 4: *a280.tsp*



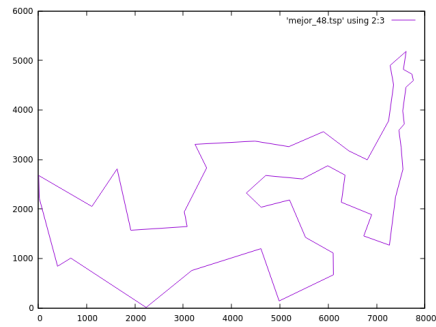
(a) Vecino más cercano



(b) Inserción más económica

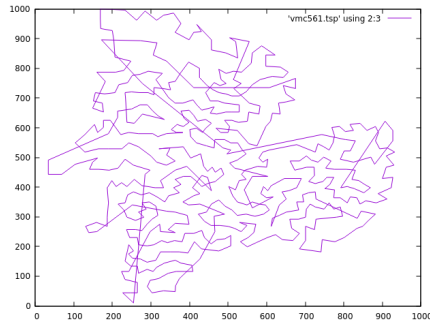


(c) Derivado de Kruskal

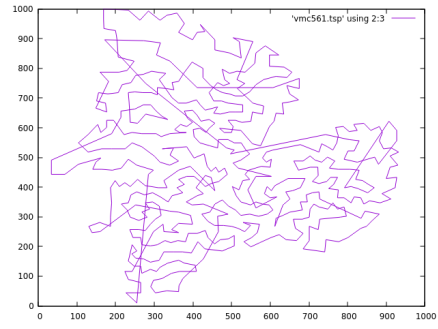


(d) Versión óptima

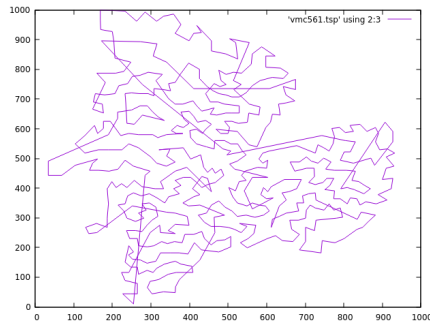
Figura 5: *att48.tsp*



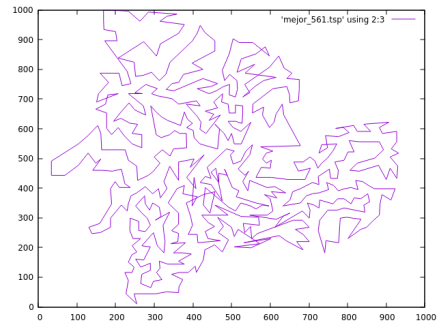
(a) Vecino más cercano



(b) Inserción más económica

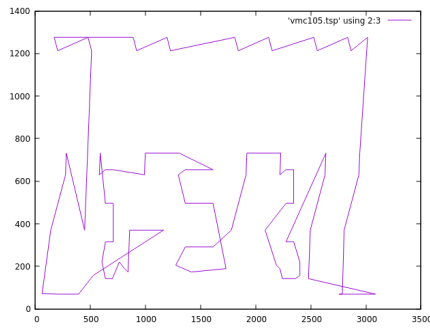


(c) Derivado de Kruskal

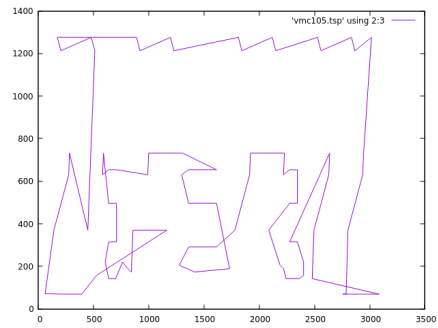


(d) Versión óptima

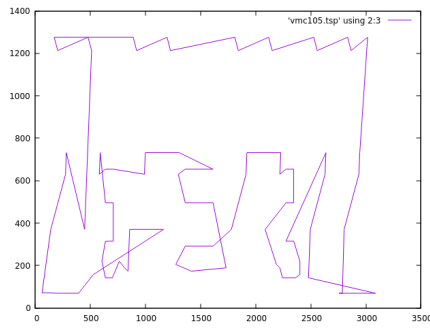
Figura 6: *pa561.tsp*



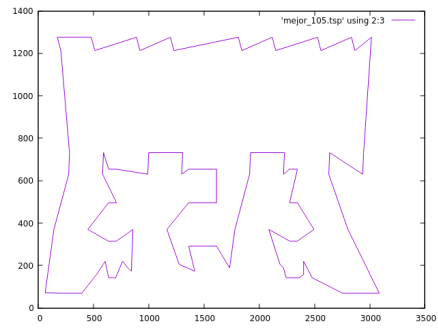
(a) Vecino más cercano



(b) Inserción más económica

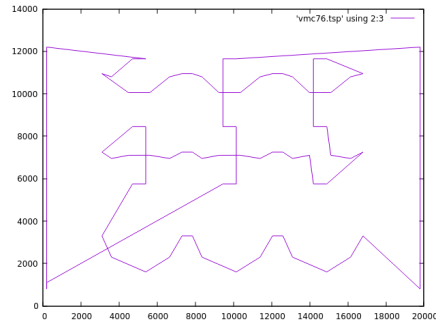


(c) Derivado de Kruskal

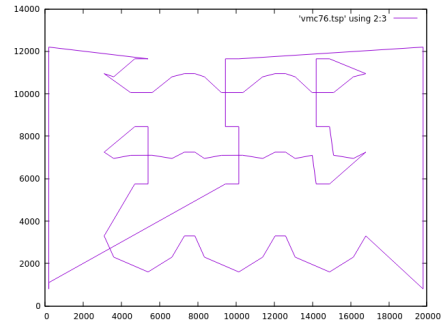


(d) Versión óptima

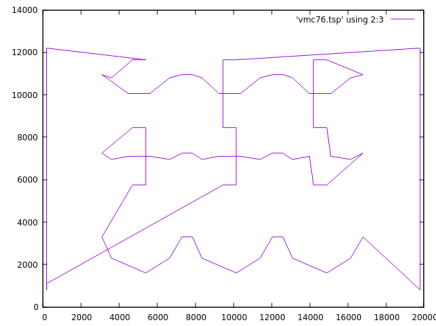
Figura 7: *lin105.tsp*



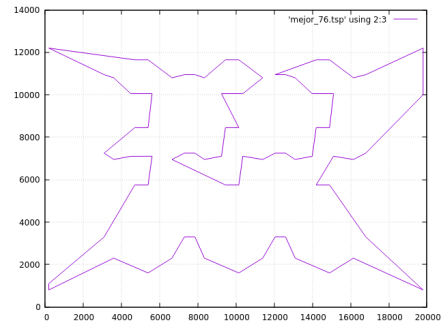
(a) Vecino más cercano



(b) Inserción más económica

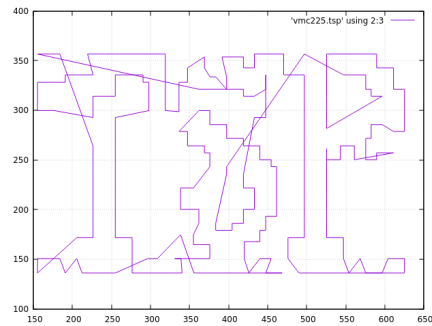


(c) Derivado de Kruskal

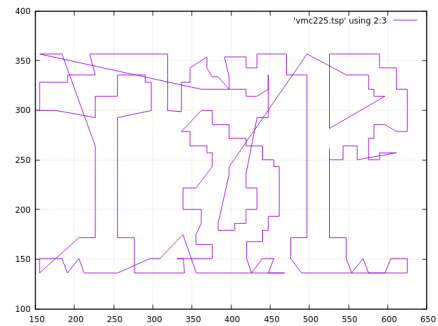


(d) Versión óptima

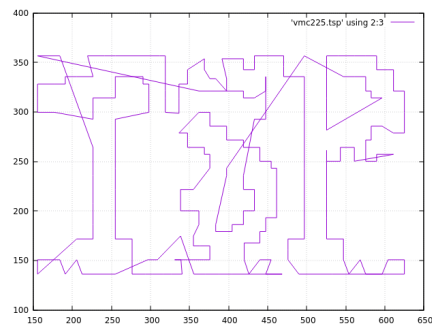
Figura 8: *pr76.tsp*



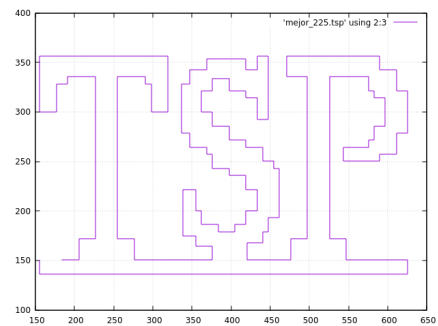
(a) Vecino más cercano



(b) Inserción más económica

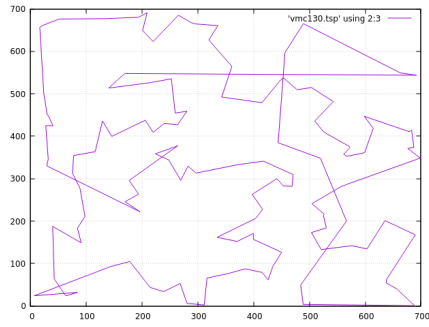


(c) Derivado de Kruskal

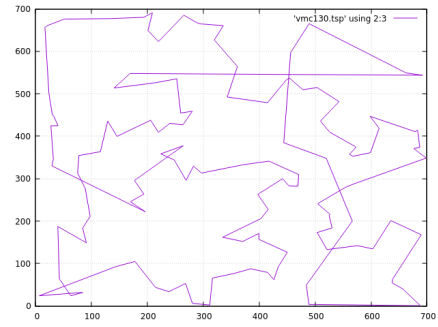


(d) Versión óptima

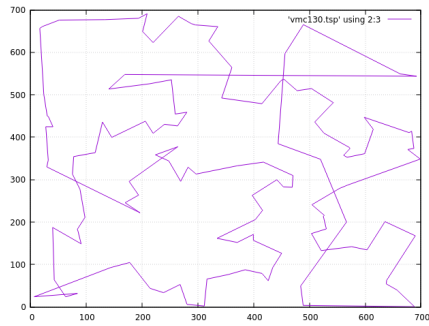
Figura 9: *tsp225.tsp*



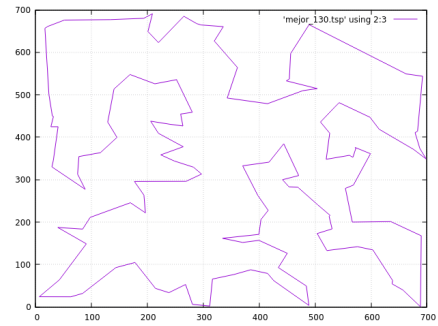
(a) Vecino más cercano



(b) Inserción más económica

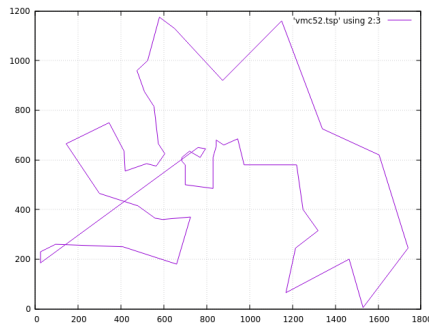


(c) Derivado de Kruskal

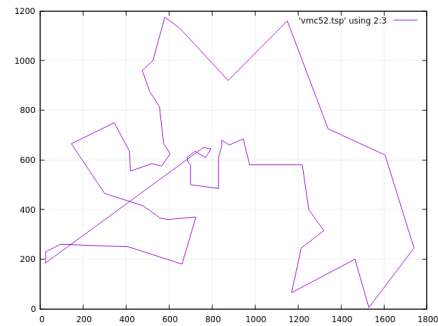


(d) Versión óptima

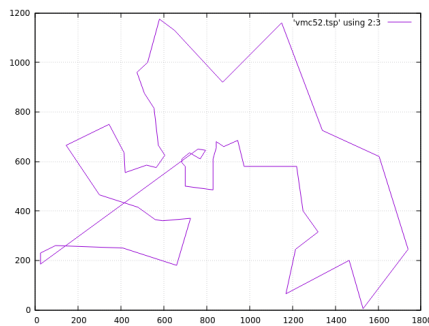
Figura 10: *ch130.tsp*



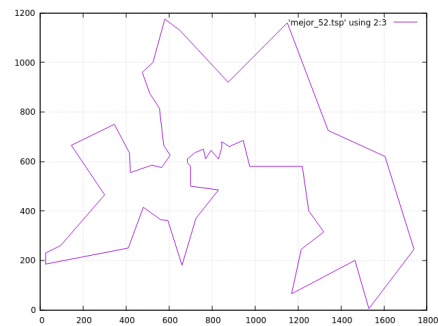
(a) Vecino más cercano



(b) Inserción más económica

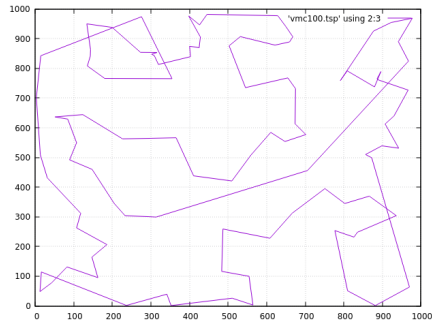


(c) Derivado de Kruskal

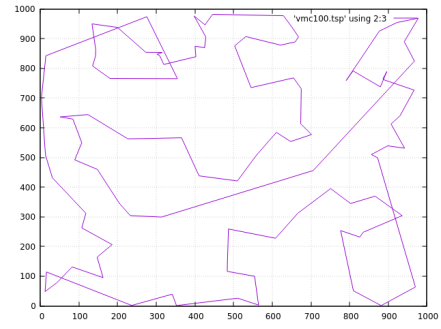


(d) Versión óptima

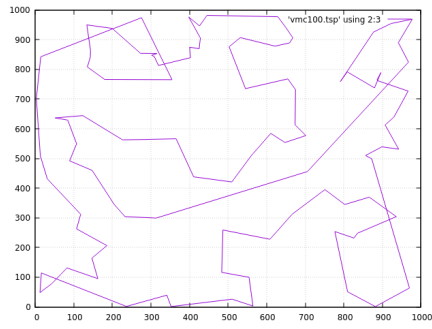
Figura 11: *berlin52.tsp*



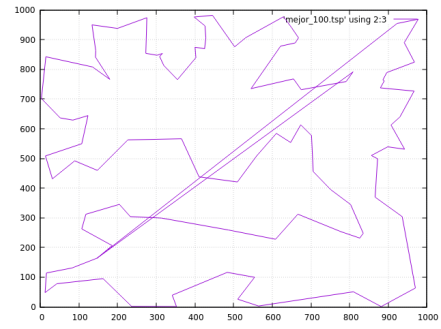
(a) Vecino más cercano



(b) Inserción más económica

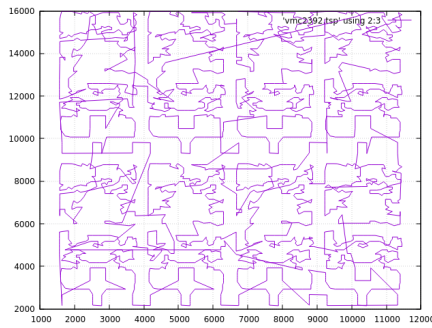


(c) Derivado de Kruskal

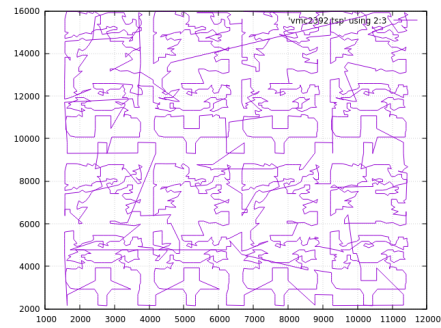


(d) Versión óptima

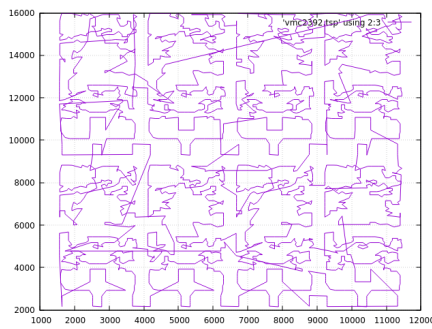
Figura 12: *rd100.tsp*



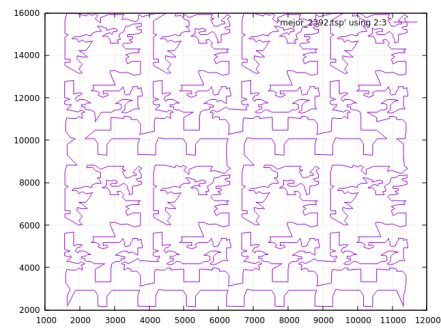
(a) Vecino más cercano



(b) Inserción más económica



(c) Derivado de Kruskal



(d) Versión óptima

Figura 13: *pr2392.tsp*

6. Anexo: código fuente

```
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 #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
35 class TSP {
36
37 private:
38     vector<ciudad> ciudades;
39     double distancia_total;
40     vector<ciudad> camino;
41     vector<vector<double>> matriz_distancias;
42     vector<bool> visitados;
43
44     double calcularDistanciaCamino(const vector<ciudad> &path) {
45         double distancia = 0;
46         for (int i = 0, j = 1; j < path.size(); i++ , j++)
```

```

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

```

```

92     par.first = resultado;
93     par.second = distancia;
94     return par;
95 }
96
97 void InicializarMatrizDistancias() {
98     for (int i = 0; i < ciudades.size(); i++)
99         for (int j = 0; j < ciudades.size(); j++)
100             matriz_distancias[i][j] = distanciaEuclidea(ciudades[i],
101                 ↪ ciudades[j]);
102 }
103
104 void Reservar(int n) {
105     visitados.resize(n);
106     matriz_distancias.resize(n);
107     for (int i = 0; i < n; i++)
108         matriz_distancias[i].resize(n);
109 }
110
111 void ResetVisitados() {
112     for (auto it = visitados.begin(); it != visitados.end(); ++it)
113         *it = false;
114 }
115
116 public:
117     TSP() {
118         distancia_total = 0;
119         ResetVisitados();
120     }
121     TSP(char *archivo) {
122         CargarDatos(archivo);
123         distancia_total = 0;
124         ResetVisitados();
125     }
126     ~TSP() {
127     }
128
129     void VecinoMasCercano() {
130         pair<vector<ciudad>, double> minimo, temp;
131         minimo = VecinoMasCercanoParcial(0);    //Calculo el vecino más
132         ↪ cercano comenzando por el primero
133         for (int i = 0; i < ciudades.size(); i++) {
134             ResetVisitados();
135             temp = VecinoMasCercanoParcial(i);
136         }
137         #if TEST
138             cout << temp.first << endl;
139             cout << "Distancia " << temp.second << endl;

```

```

138  #endif
139      if (temp.second < minimo.second)
140          minimo = temp;
141      }
142      camino = minimo.first;
143      distancia_total = minimo.second;
144  }
145
146  void InsercionMasEconomica() {
147
148  }
149
150  int GetTamano() {
151      return ciudades.size();
152  }
153
154  void DerivadoKruskal() {
155
156  }
157
158  void CargarDatos(char *archivo) {
159      ifstream datos;
160      string s;
161      int n;
162      ciudad aux;
163      datos.open(archivo);
164      if (datos.is_open()) {
165          datos >> s;  //Leo DIMENSIÓN (cabecera)
166          datos >> n;  //Leo NÚMERO de ciudades y reservo espacio en
                     ↪ matrices y vector.
167          Reservar(n);
168
169          for (int i = 0; i < n; i++) {
170              datos >> aux.n;  // Leo número de ciudad
171              aux.n--;        //Decremento el número: los índices del archivo
                             ↪ comienzan en 1. Los del vector en 0.
172              datos >> aux.x >> aux.y; //Leo coordenadas
173              ciudades.push_back(aux);
174          }
175          datos.close();
176      }
177      else
178          cout << "Error al leer " << archivo << endl;
179
180      InicializarMatrizDistancias();
181  }
182
183  void imprimirResultado() {

```

```

184     cout << endl << "Mejor solución:" << endl;
185     cout << camino << endl;
186     cout << "Distancia: " << distancia_total << endl;
187 }
188
189 void Exportar(const char *name) {
190     ofstream salida;
191     salida.open(name);
192     if (salida.is_open()) {
193         salida << "DIMENSION: ";
194         salida << ciudades.size() << endl;
195         salida << "DISTANCIA: " << distancia_total << endl;
196         for (auto it = camino.begin(); it != camino.end(); ++it) {
197             salida << it->n + 1 << " " << it->x << " " << it->y << endl;
198         }
199         salida.close();
200     }
201     else
202         cout << "Error al exportar." << endl;
203 }
204
205 };
206
207
208 int main(int argc, char **argv) {
209
210     if (argc != 2) {
211         cerr << "Error de formato: " << argv[0] << " <fichero>." << endl;
212         exit(-1);
213     }
214
215     string nombre="";
216
217     /****** Vecino más cercano******/
218     TSP vecino_mas_cercano(argv[1]);
219     cout << "Heurística del Vecino más cercano." << endl;
220     vecino_mas_cercano.VecinoMasCercano();
221     vecino_mas_cercano.imprimirResultado();
222
223     nombre = "vmc";
224     nombre += to_string(vecino_mas_cercano.GetTamano());
225     nombre += ".tsp";
226     vecino_mas_cercano.Exportar(nombre.c_str());
227     cout << "Exportado archivo " << nombre << endl;
228
229     /****** Inserción más económica******/
230
231     TSP insercion_mas_economica(argv[1]);

```



```

232
233     cout << "Heurística de la inserción más económica." << endl;
234     insercion_mas_economica.InsercionMasEconomica();
235     insercion_mas_economica.imprimirResultado();
236
237     /*nombre = "ime_";
238     nombre += argv[1];
239     insercion_mas_economica.Exportar(nombre.c_str());
240     cout << "Exportado archivo " << nombre << endl;*/
241
242
243     /****** Derivado de Kruskal******/
244
245     TSP derivado_kruskal(argv[1]);
246
247     cout << "Heurística derivada de Kruskal." << endl;
248     derivado_kruskal.DerivadoKruskal();
249     derivado_kruskal.imprimirResultado();
250
251     /*nombre = "kruskal_";
252     nombre += argv[1];
253
254     derivado_kruskal.Exportar(nombre.c_str());
255     cout << "Exportado archivo " << nombre << endl;*/
256
257 }

```

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 GetTamano() {
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.GetTamanio());
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
6  ↪  ../datosTSP/berlin52.opt.tour
7  ./calcular_distancia ../datosTSP/ch130.tsp ../datosTSP/ch130.opt.tour
8  ./calcular_distancia ../datosTSP/gr96.tsp ../datosTSP/gr96.opt.tour
9  ./calcular_distancia ../datosTSP/lin105.tsp ../datosTSP/lin105.opt.tour
10 ./calcular_distancia ../datosTSP/pa561.tsp ../datosTSP/pa561.opt.tour
11 ./calcular_distancia ../datosTSP/pr2392.tsp ../datosTSP/pr2392.tour
12 ./calcular_distancia ../datosTSP/pr76.tsp ../datosTSP/pr76.tour
13 ./calcular_distancia ../datosTSP/rd100.tsp ../datosTSP/rd100.tour
14 ./calcular_distancia ../datosTSP/tsp225.tsp ../datosTSP/tsp225.tour
15 ./calcular_distancia ../datosTSP/ulysses16.tsp ../datosTSP/ulysses16.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/pr2392.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'
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'
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'
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'
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'
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'
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'
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 'pr2392_mejor'
110 plot 'mejor_4392.tsp' using 2:3 with lines
111
112 set output 'pr2392_vmc.png'
113 plot 'vmc4392.tsp' using 2:3 with lines
114
115 set output 'pr2392_ime.png'
116 plot 'ime4392.tsp' using 2:3 with lines
117
118 set output 'pr2392_kruskal.png'
119 plot 'kruskal4392.tsp' using 2:3 with lines
120
121
122
123 set output 'pr76_mejor'
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'

```



```

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'
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'
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.