**Borrador**

**PROBLEMA DEL VIAJANTE DE COMERCIO**

**Algoritmo usando estrategia de inserción**

En este algoritmo greedy, se sigue el procedimiento que se expone en la guía de la práctica, por lo que nos centraremos ahora en indicar cuáles son los elementos típicos de un algoritmo greedy presentes en nuestro algoritmo:

**- Conjunto de candidatos (C):** ciudades no seleccionadas.

**- Conjunto de candidatos seleccionados (S):** ciudades que han cumplido con el criterio y han sido insertadas en la solución.

**- Función solución (FS):** se trata del bucle externo while del método calcularSolucion(). Controla que se ha llegado a la solución, lo que ocurre cuando hay tantas ciudades en el conjunto solución como en el conjunto de candidatos.

**- Función de factibilidad (FF):** no es necesaria, pues el conjunto de candidatos siemrpre forma solución.

**- Función selección:** son los dos bucles for internos al while del método calcularSolucion(), que prueban todas las posibles combinaciones de ciudades y posiciones en el conjunto solución actual para determinar cual da una menor longitud de circuito resultante.

**- Función objetivo:** es el método mostrarSolucion() que, como su propio nombre indica, se encarga de mostrar el conjunto solución por pantalla.

Pasamos ahora a mostrar el código:

- main.cpp:

#include <cstdlib>

#include "Mapa.h"

**using** **namespace** std;

**int** main(**int** argc, **char**\*\* argv) {

Mapa mapa;

**if** (argc < 2){

cout << "Se necesita el nombre del fichero de ciudades como argumento\n";

**return** 1;

}

**if**(mapa.cargarDeFichero(argv[1]))

mapa.calcularSolucion();

mapa.mostrarSolucion();

**return** 0;

}

- Mapa.h:

#include <cstdlib>

#include "Mapa.h"

**using** **namespace** std;

**int** main(**int** argc, **char**\*\* argv) {

Mapa mapa;

**if** (argc < 2){

cout << "Se necesita el nombre del fichero de ciudades como argumento\n";

**return** 1;

}

**if**(mapa.cargarDeFichero(argv[1]))

mapa.calcularSolucion();

mapa.mostrarSolucion();

**return** 0;

}

- Mapa.cpp:

#include <fstream>

#include <cmath>

#include "Mapa.h"

**using** **namespace** std;

Mapa::Mapa() {

}

Mapa::Mapa(**const** Mapa& orig) {

}

Mapa::~Mapa() {

}

**void** Mapa::calcularSolucion(){

vector<**int**> tmp, vacio;

**int** max\_norte = ciudades[0].second, max\_este = ciudades[0].first, max\_oeste = ciudades[0].first, pos\_este, pos\_oeste, pos\_norte;

**int** dist\_min, dist\_act, min\_pos, lugar\_min, lugar\_aux, c; //min\_pos es la posicion de la solucion en el vector de ciudades y lugar\_min la posicion en el vector de soluciones que ocuapara el nuevo minimo.

**bool** stop;

//Busco e inserto la ciudad mas al este, mas al oeste y mas al norte

**for** (**int** j=0; j<ciudades.size(); j++){

**if**(ciudades[j].first > max\_este){

pos\_este = j;

max\_este = ciudades[j].first;

}**if**(ciudades[j].first < max\_oeste){

pos\_oeste = j;

max\_oeste = ciudades[j].second;

}**if**(ciudades[j].second > max\_norte){

pos\_norte = j;

max\_norte = ciudades[j].second;

}

}

solucion.push\_back(pos\_este);

solucion.push\_back(pos\_oeste);

solucion.push\_back(pos\_norte);

//Recorro el resto de ciudades

**while**(solucion.size() < ciudades.size()){

stop = **false**;

**for**(**int** a=0; a<ciudades.size() && !stop; a++){ //Inicializo dist\_min y min\_pos con la primera ciudad que no este incluida ya en solucion

**if**(!presenteEnSolucion(a)){

dist\_min = calcularLongitud(a, lugar\_min);

min\_pos = a;

stop = **true**;

}

}

**for**(**int** i=0; i<ciudades.size(); i++){ //Recorro el resto de ciudades

**if**(!presenteEnSolucion(i)){

dist\_act = calcularLongitud(i, lugar\_aux); //Para cada una de ellas calculo la longitud de camino que resultaria

**if**(dist\_act < dist\_min){

dist\_min = dist\_act;

min\_pos = i;

lugar\_min = lugar\_aux;

}

}

}

**for**(c=0; c<lugar\_min; c++) //Inserto la ciudad mas conveniente en la posicion mas conveniente

tmp.push\_back(solucion[c]);

tmp.push\_back(min\_pos);

**for**(**int** d=c; d<solucion.size(); d++)

tmp.push\_back(solucion[d]);

solucion = tmp;

tmp = vacio;

}

}

**int** Mapa::calcularLongitud(**int** pos, **int** lugar){

**int** aux = 0, min = 0;

//Inicializo min insertando el elemento en la primera posicion posible, 0:

min += calcularDistancia(pos, solucion[0]);

**for** (**int** t=0; t<solucion.size() - 1; t++){

min += calcularDistancia(solucion[t], solucion[t+1]);

}

min += calcularDistancia(solucion[solucion.size()-1], pos);

//Busco la mejor posicion en la sucesion de ciudades solucion en la que puedo insertar la ciudad pos

**for**(**int** i=0; i<solucion.size() - 1; i++){

aux = 0;

**for**(**int** j=0; j<i; j++)

aux += calcularDistancia(solucion[j], solucion[j+1]);

aux += calcularDistancia(solucion[i], pos) + calcularDistancia(pos, solucion[i + 1]);

**for**(**int** k=i + 1; k<solucion.size() - 1; k++)

aux += calcularDistancia(solucion[k], solucion[k+1]);

aux += calcularDistancia(solucion[solucion.size()-1], solucion[0]);

**if**(aux < min){

min = aux;

lugar = i+1;

}

}

**return** min;

}

**int** Mapa::calcularDistancia(**int** a, **int** b){

pair<**double**,**double**> ciudada = ciudades[a], ciudadb = ciudades[b];

**int** dist = sqrt((ciudadb.first-ciudada.first)\*(ciudadb.first-ciudada.first) + (ciudadb.second-ciudada.second)\*(ciudadb.second-ciudada.second));

**return** dist;

}

**bool** Mapa::presenteEnSolucion(**int** x){

**bool** presente = **false**;

**for** (**int** i=0; i<solucion.size() && !presente; i++)

**if**(solucion[i] == x)

presente = **true**;

**return** presente;

}

**void** Mapa::mostrarSolucion(){

cout << endl;

**for** (**int** i=0; i<solucion.size(); i++)

cout << solucion[i]+1 << "\n";

cout << endl;

}

**bool** Mapa::cargarDeFichero(**const** **char** \*fichero){

**bool** b = **true**;

**int** num, basura;

pair<**double**, **double**> aux;

ifstream fentrada;

fentrada.open(fichero);

**if** (fentrada){

fentrada.ignore(100, ':');

fentrada >> num;

**for**(**int** i=0; i<num; i++){

fentrada >> basura;

fentrada >> aux.first;

fentrada >> aux.second;

ciudades.push\_back(aux);

}

**if** (!fentrada){

cerr << "\nError en la lectura del fichero"<< endl;

b = **false**;

}

fentrada.close();

}**else**{

cerr << "\nError de apertura de fichero" <<endl;

b = **false**;

}

**return** b;

}

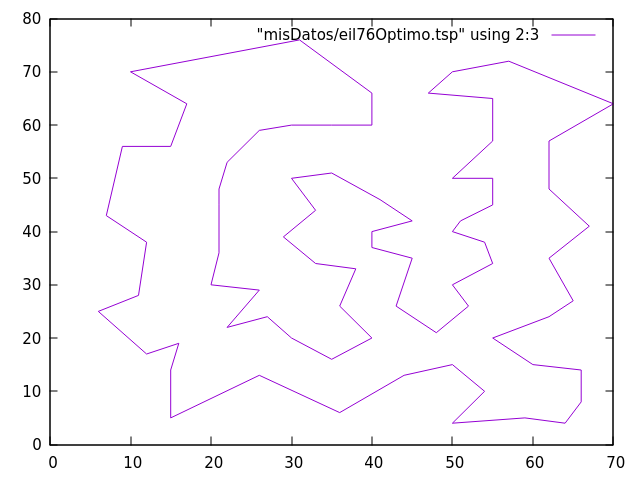
mapa.mostrarSolucion();

**return** 0;

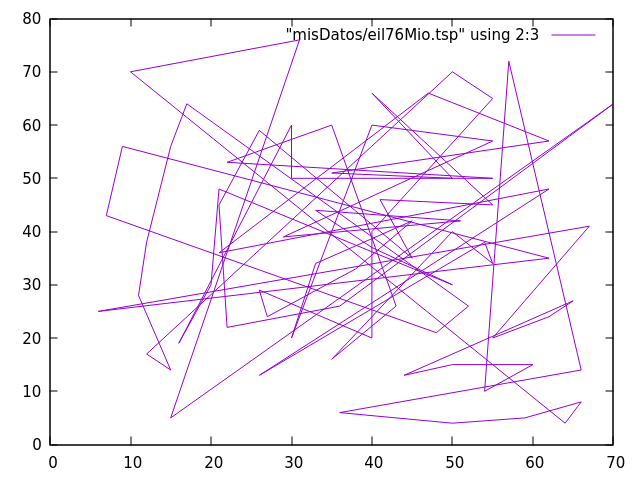
**}**

A continuación, se va a establecer una comparación entre el itinerario solución hallado por el algoritmo y el itinerario solución óptimo:

- Óptimo:



- Greedy:



Como se puede comprobar, este algoritmo greedy queda muy lejos de alcanzar la solución óptima. No obstante, es capaz de resolver estos problemas en relativamente poco tiempo, razón por la cual puede llegar a ser deseable en casos en los que algoritmos óptimos puedan tardar un tiempo inasumible en localizar la solución.

**Algoritmo 3 TSP, Optimización de la Colonia de Hormigas**

El último algoritmo propuesto para resolver el problema del viajante de comercio es el de la Colonia de Hormigas, que si bien no es un algoritmo voraz, funciona de forma muy similar. Este algoritmo, como su propio nombre indica, se basa en el comportamiento de las hormigas.

Las hormigas cuando salen en busca de alimento forman un camino, apreciable a simple vista, que lleva hasta dicho alimento, y que con el paso del tiempo, se va transformando en un camino mínimo entre el hormiguero y su destino. Esto se consigue gracias al uso de feromonas. Cuando una hormiga sale de viaje, va dejando un rastro de feromonas que usará más adelante para volver al origen. Pero aunque es un sistema muy eficaz, hay que tener en cuenta que el rastro no es eterno, ya que las feromonas se evaporan, ni inmutable, ya que otra hormiga podría crear un camino completamente nuevo.

Ahora, supongamos que tenemos dos hormigas llamadas A y B y que el camino que ha encontrado una de ellas, A por ejemplo, es más corto que el que ha encontrado B. En este caso, como el camino de A es más corto, tardará menos tiempo en recorrer el camino que B, haciendo que la cantidad de feromonas que se encuentra en el camino de A en cierto momento concreto sea mayor que en el camino de B. Esto lo que provoca es que el camino de A se convierta en el camino predilecto ya que lo transitarán muchas más hormigas que, a su vez, aumentarán aún más la cantidad de feromonas, haciendo que el camino de B desaparezca. De este fenómeno surge el algoritmo de Optimización de la Colonia de Hormigas.