MEMORIA DE LA PRÁCTICA 3. ALGORÍTMICA

ALGORITMOS GREEDY

Pablo Alaminos Morón
Celia Botella López
José Luis de la Rosa Morillas
David Pérez Núñez
Antonio Rodríguez Alaminos

ÍNDICE

REPARTO DE TRABAJO:
Pablo Alaminos Morón → Realización de la presentación.
Celia Botella López → Realización del algoritmo de cercanía y del de inserción.
José Luis de la Rosa Morillas → Realización de un algoritmo propio base y corrección de errores de los algoritmos.
David Pérez Núñez → Descripciones de los algoritmos de cercanía e inserción y obtención de imágenes.
Antonio Rodríguez Alaminos → Realización del algoritmo propio TSPOptimo y descripción del código propio.

DEFINICIÓN DE ALGORITMO GREEDY
Un algoritmo greedy es una estrategia de búsqueda por la cual se sigue una heurística consistente en elegir la opción óptima en cada paso local con el fin de llegar a una solución general óptima.

Un algoritmo Greedy o voraz se caracteriza por:
-La construcción de la solución por etapas.
-En cada momento elige un movimiento de acuerdo con un criterio de selección.
-No vuelve a considerar los movimientos ya seleccionados ni puede volver a modificarlos en posteriores etapas.
-Utiliza una función objetivo o criterio de optimalidad.

DESCRIPCIÓN DEL PROBLEMA
El problema del viajante de comercio consiste en, dado un conjunto de ciudades y las distancias entre ellas, determinar cuál es el camino más corto que pase por todas ellas y vuelva al punto de inicio, o lo que es lo mismo, un circuito hamiltoniano minimal.

Para la resolución del problema, usaremos tres estrategias: cercanía, inserción y propio.

ALGORITMO DE CERCANÍA
-Explicación
El algoritmo de cercanía se basa en la heurística del vecino más cercano, cuyo funcionamiento es extremadamente simple: dada una ciudad inicial v0, se agrega como ciudad siguiente aquella vi(no incluida en el circuito) que se encuentre más cercana a v0. El procedimiento se repite hasta que todas las ciudades se hayan visitado.

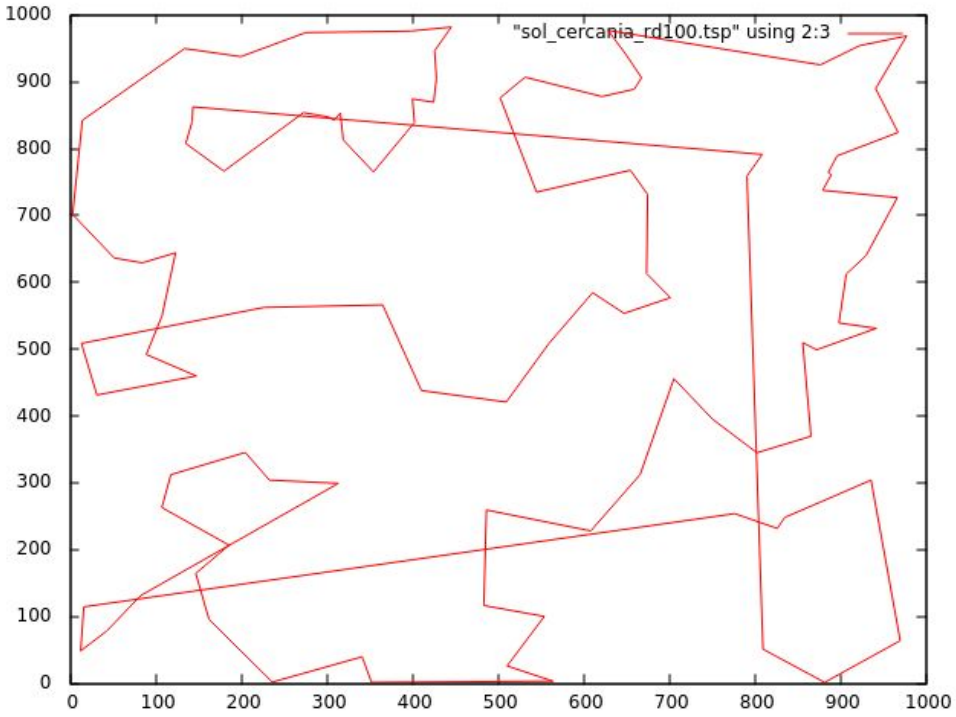
Dado un vector de nodos, tomamos el primer elemento de este como inicio y lo introduciremos en el vector solución. Recurrentemente, tomaremos el nodo más cercano al actual que no esté en la solución, lo introducimos en la solución y lo convertimos en el nodo actual. Repetimos el proceso n-1 veces y finalmente se vuelve a introducir el primer nodo. Sin embargo, esta resolución del problema no garantiza que el camino tenga longitud mínima.

-Código

```
//Busca la ciudad más cercana a la actual y la inserta en la solucion.
void insertar_ciudad(vector<Ciudad> &ciudades, vector<Ciudad> &solucion){
   Ciudad actual;
   double distancia_minima = -1;
   int posicion_insertar = 0;

   for(int i = 0; i < ciudades.size(); i++){
         actual = ciudades[i];
         solucion.push_back(actual);
         double distancia = distancia_total(solucion);
         if(distancia < distancia_minima || distancia_minima==-1){
                  distancia_minima = distancia;
                  posicion_insertar = i;
         }
         solucion.pop_back();
   }

   solucion.push_back(ciudades[posicion_insertar]);
   ciudades.erase(ciudades.begin() + posicion_insertar);
}


//Calcula una ruta mediante la estrategia de cercania.
void calcular_ruta(vector<Ciudad> &ciudades, vector<Ciudad> &solucion){
   Ciudad primera=ciudades[0];
   solucion.push_back(primera);
   ciudades.erase(ciudades.begin());

   while(!ciudades.empty())
         insertar_ciudad(ciudades, solucion);

   solucion.push_back(primera);
}
```

-Gráficas cercanía

Fichero rd100.tsp



Fichero a280.tsp

ALGORITMO DE INSERCIÓN

-Explicación

En las estrategias de 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 poder implementar este tipo de estrategia deben definirse 3 elementos:

→ Cómo se construye el recorrido parcial inicial

→ Cual es el nodo siguiente a insertar en el recorrido parcial

→ Donde se inserta el nodo seleccionado

Dado un vector de nodos, partimos de un ciclo de tres nodos creado con las siguientes ciudades: la más al este, la más al oeste, y la más al norte. Partiendo de esto, vamos añadiendo más ciudades, las más cercanas a los lados, destruyendo los mismos lados en el proceso y creando otros enlaces entre los nodos afectados. La característica más atractiva de este algoritmo es que el camino que te aparece es el óptimo.

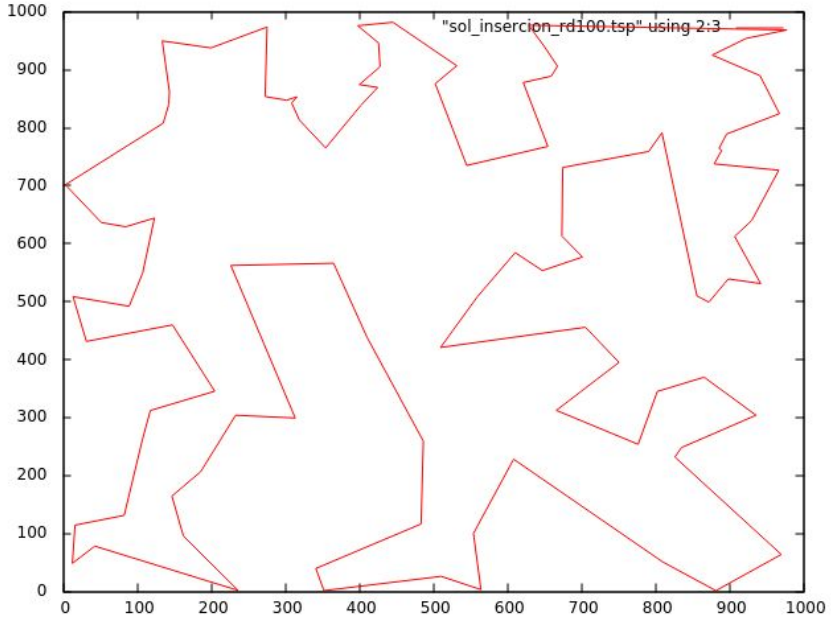
-Código

```
//Busca la ciudad más cercana a la actual y la inserta en la solución.
void insertar_ciudad(vector<Ciudad> &ciudades, vector<Ciudad> &solucion){
   Ciudad actual;
   double distancia_minima = -1;
   int posicion_insertar = 0;

   actual = ciudades[0];

   for(int i = 0; i < solucion.size(); i++){
         solucion.insert(solucion.begin() + i, actual);
         double distancia = distancia_total(solucion);
         if(distancia < distancia_minima || distancia_minima==-1){
               distancia_minima = distancia;
               posicion_insertar = i;
         }
         solucion.erase(solucion.begin() + i);
   }

   solucion.insert(solucion.begin() + posicion_insertar, actual);
   ciudades.erase(ciudades.begin());
}
```

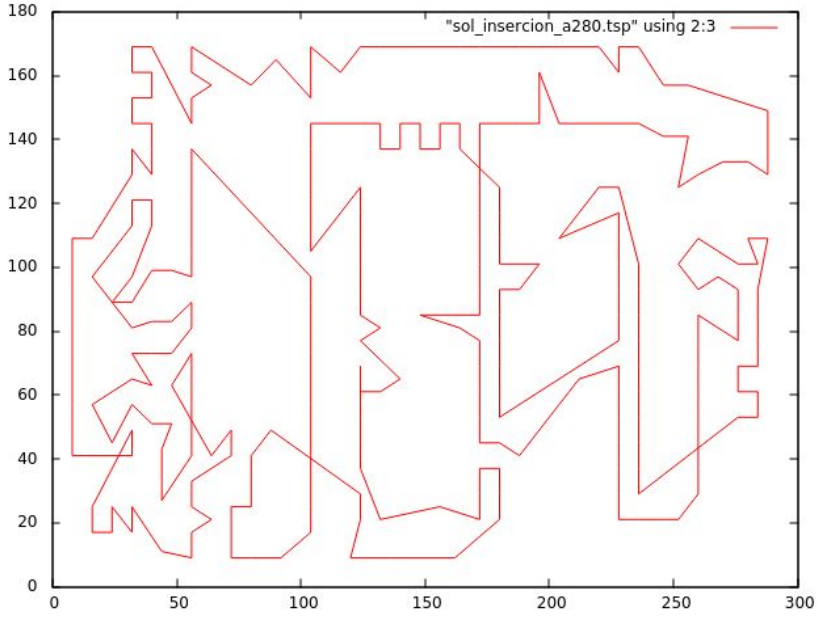
```
//Calcula una ruta mediante la estrategia de insercción.
void calcular_ruta(vector<Ciudad> &ciudades, vector<Ciudad> &solucion){
    triangulo(ciudades, solucion);

    while(!ciudades.empty())
        insertar_ciudad(ciudades, solucion);
}
```

-Gráficas

Fichero rd100.tsp



Fichero a280.tsp



ALGORITMO PROPIO

-Explicación

Dado nuestro vector de nodos, los introduciremos en una matriz nxn donde cada casilla i,j es la distancia entre el nodo i y el nodo j. A partir de aquí iremos introduciendo las distancias más pequeñas, teniendo en cuenta que cada nodo no puede aparecer más de dos veces. Repetimos el proceso n veces.

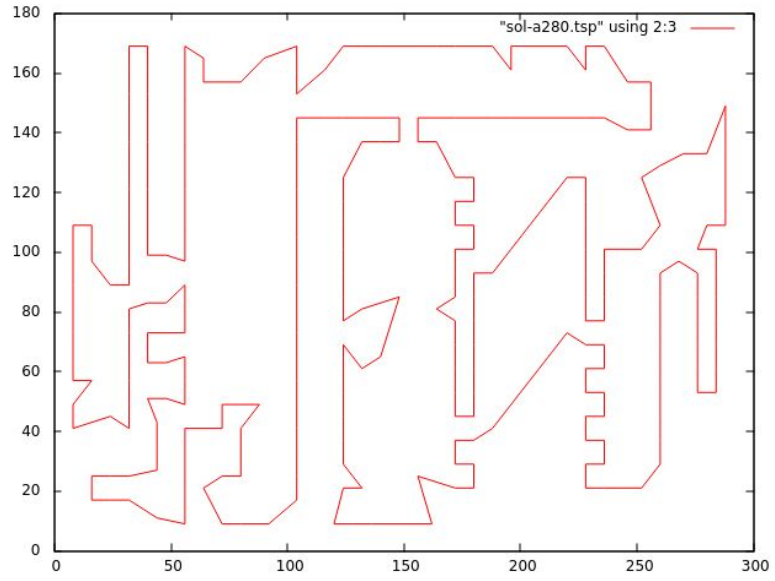
-Código

```
void calcularRuta(vector<Ciudad> &ciudades, vector<Ciudad> &ciudades_solucion){
   int mCiudades [1024][1024];
   int ciudad1, ciudad2;
   int usados [ciudades.size()]; //Registraremos los nodos utilizados
   par aux;
   vector <par> resultados;

   for (int i=0;i<ciudades.size();i++)
         usados[i]=0;

   generaMatriz(ciudades, mCiudades);
   obtenerPasoMenor(ciudades, mCiudades, ciudad1, ciudad2);

   aux.i=ciudad1;
   usados[ciudad1++];

   aux.j=ciudad2;
   usados[ciudad2++];

   resultados.push_back(aux); //Insertamos el par de nodos cercanos

   //ciudades.erase(ciudades.begin());
   while(encontrado(usados, ciudades.size())){

         //Vamos seleccionando las ciudades, dentro de la funciÃ³n tambiÃ©n se va eliminando
         obtenerPasoMenor(ciudades, mCiudades, ciudad1, ciudad2);
         while (usados[ciudad1]!=2 && usados[ciudad2]!=2) //Si no encontramos una soluciÃ³n vÃ¡lida volvemos a buscar
                  obtenerPasoMenor(ciudades, mCiudades, ciudad1, ciudad2);

         aux.i=ciudad1; //Insertamos el primer valor minimo
         usados[ciudad1++];
```

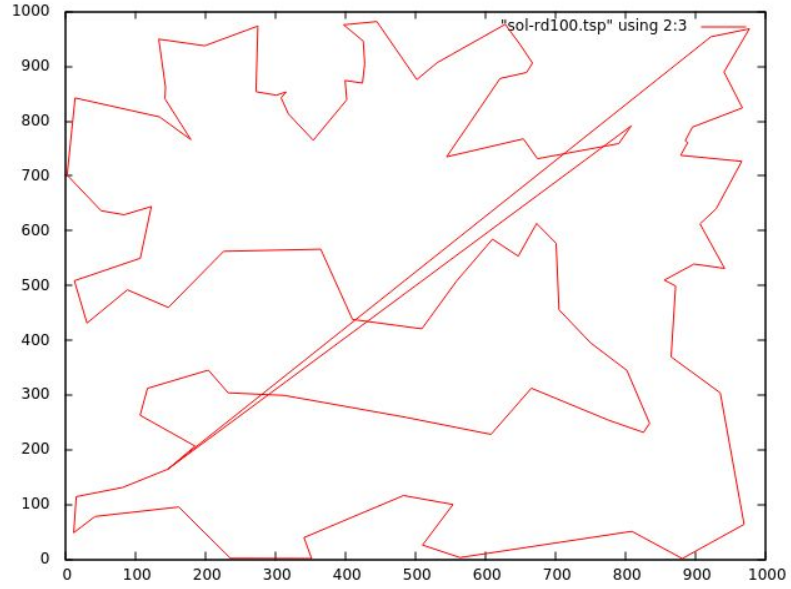
```
            aux.j=ciudad2; //Insertamos su enlace
            usados[ciudad2++];

            resultados.push_back(aux);
        }

        unirPuntos(resultados,ciudades_solucion, ciudades);

}
```

-Gráficas

Fichero a280.tsp

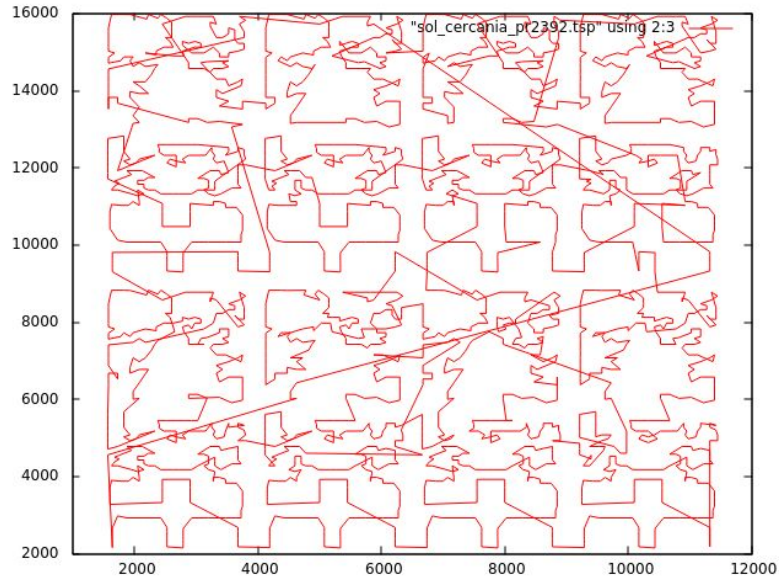


Fichero rd100.tsp

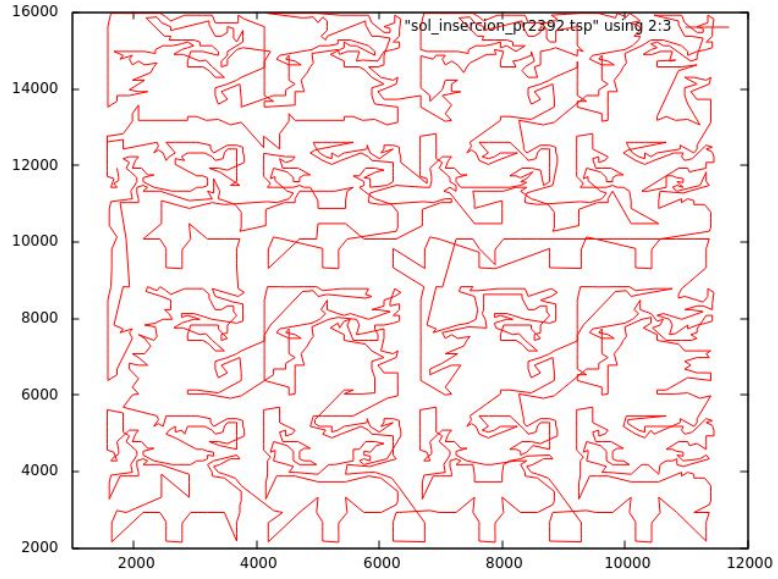


-COMPARATIVA DE GRAFOS

Algoritmo de cercanía



Algoritmo de inserción



Algoritmo propio