Problema del vendedor viajero

Por Montserrat Sacie Alcázar, Guillermo Cortina Fernández, Alberto Pastor Moreno e Iván Fernández Mena

Índice

¿De qué va el problema?

- Un comerciante desea visitar “n” ciudades, comenzando y terminando el viaje en la misma ciudad.
- Conocemos la distancia entre cada ciudad además de sus conexiones.
- Se engloba dentro de los “Problemas de Rutas”.

Un poco de historia

Fue definido en los años 1800s por el matemático irlandés William Rowan Hamilton y por el matemático británico Thomas Kirkman.

Hasta 1930 cuando varios matemáticos en Viena y Harvard plantean una formulación matemática acorde a la problemática correspondiente.

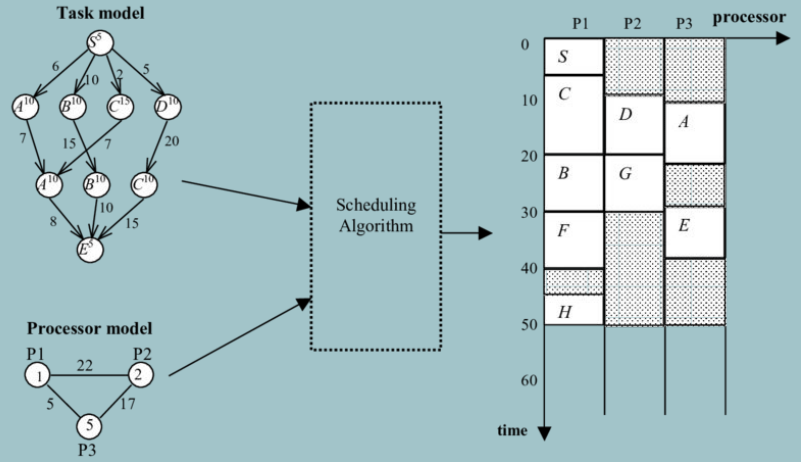
Un poco de historia

Naciendo así la cuestión que aquí tratamos: ¿Cómo encontrar el camino óptimo? Hassler Whitney de la Universidad de Princeton introdujo el nombre “travelling salesman problem”, TSP, poco tiempo después.

Año	Autores	Nodos
1954	G. Dantzig, R. Fulkerson, S. Johnson	49
1971	M. Held, R.M. Karp	64
1975	P.M. Camerini, L. Fratta, F. Maffioli	67
1977	M. Grötschel	120
1980	H. Crowder and M. W. Padberg	318
1987	M. Padberg and G. Rinaldi	532
1987	M. Grötschel and O. Holland	666
1987	M. Padberg and G. Rinaldi	2392
1994	D. Applegate, R. Bixby, V. Chvátal, W. Cook	7397
1998	D. Applegate, R. Bixby, V. Chvátal, W. Cook	13509
2001	D. Applegate, R. Bixby, V. Chvátal, W. Cook	15112
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook	18512
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, K. Helsgaun	24978
2004	W. Cook, Espinoza and Goycoolea	33810
2006	D. Applegate, R. Bixby, V. Chvátal, W. Cook	85900

Aplicaciones

- Problema de scheduling
- Problema de la placa de circuitos impresos.
- Estructura de la red de Internet
- Problema de red de basuras
- Otras...



Modelización

Sea **G = (V, A)** un grafo completo. Los vértices **i = 2, . . . , n** se corresponden con las ciudades a visitar y el vértice 1, con la ciudad de origen y destino. A cada **arco (i, j)**, se le asocia un valor no negativo **cij** , que representa el coste de viajar del nodo i al j.

Función objetivo:

$$min \sum_{i \neq 1} c_{ij} x_{ij}$$

Modelización

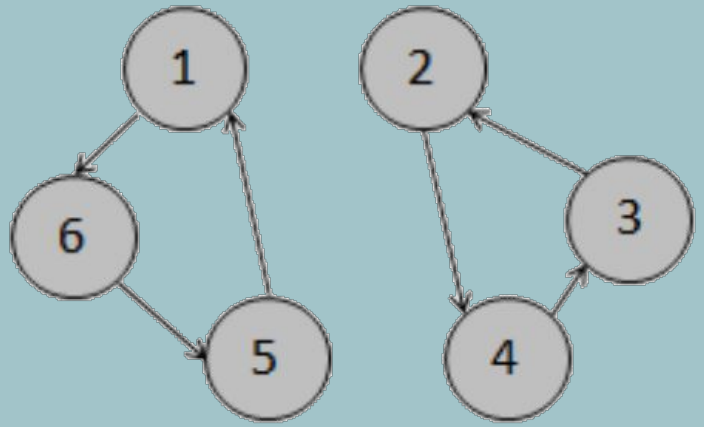
Sujeto a las condiciones:

$$\sum_{j \in \delta^-(i)} x_{ij} = 1 \forall i \in V, \qquad \sum_{j \in \delta^+(i)} x_{ij} = 1 \forall i \in V,$$

donde:

$$\delta^-(i) = \{a = (j, i) \in A\} \qquad \delta^+(i) = \{a = (i, j) \in A\}$$

Modelización



No es un camino válido para el TSP. Necesitamos por lo tanto añadir alguna restricción más al problema.

Limitamos el número de arcos a dos en todos estos subconjuntos, entonces evitaríamos subcircuitos dentro del grafo.

$$\forall W \subsetneq V, \quad A(W) = \{a = (i,j) \in A | i,j \in W\}$$

Modelización

Así, las restricciones de ruptura de subcircuito son las siguientes:

$$\sum_{i,j \in A(W)} X_i j \leq |W| - 1 \quad \forall W \subseteq V, 2 \leq |W| \leq n/2$$

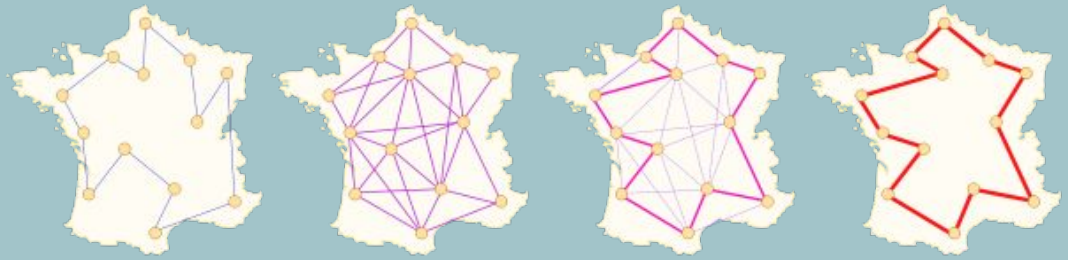
equivalente a

$$\sum_{i \in W, j \notin W} X_i j \geq 1 \quad \forall W \subseteq V, 2 \leq |W| \leq n/2$$

Algoritmos resolución TSP

La complejidad del cálculo del problema del agente viajero ha despertado múltiples iniciativas por mejorar la eficiencia en el cálculo de rutas.

- Método de fuerza bruta



Algoritmos resolución TSP

1. **Algoritmos heurísticos de construcción**

   Método del vecino más cercano

2. **Algoritmos heurísticos de mejora**

   K-opt

3. **Algoritmos exactos**

   Método de branch and bound - WINQSB

Algoritmos resolución TSP

4. **Método genérico de Tucker, Miller y Zemlin**
5. **Otros algoritmos**

    Búsqueda Tabú

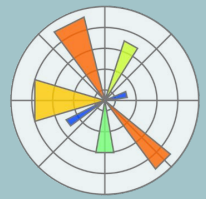
    Algoritmos genéticos

    GRAPS

Implementación

En la implementación del algoritmo del vecino más cercano hemos utilizado el lenguaje de programación **Python** junto a las librerías **matplotlib, numpy y networkx**.

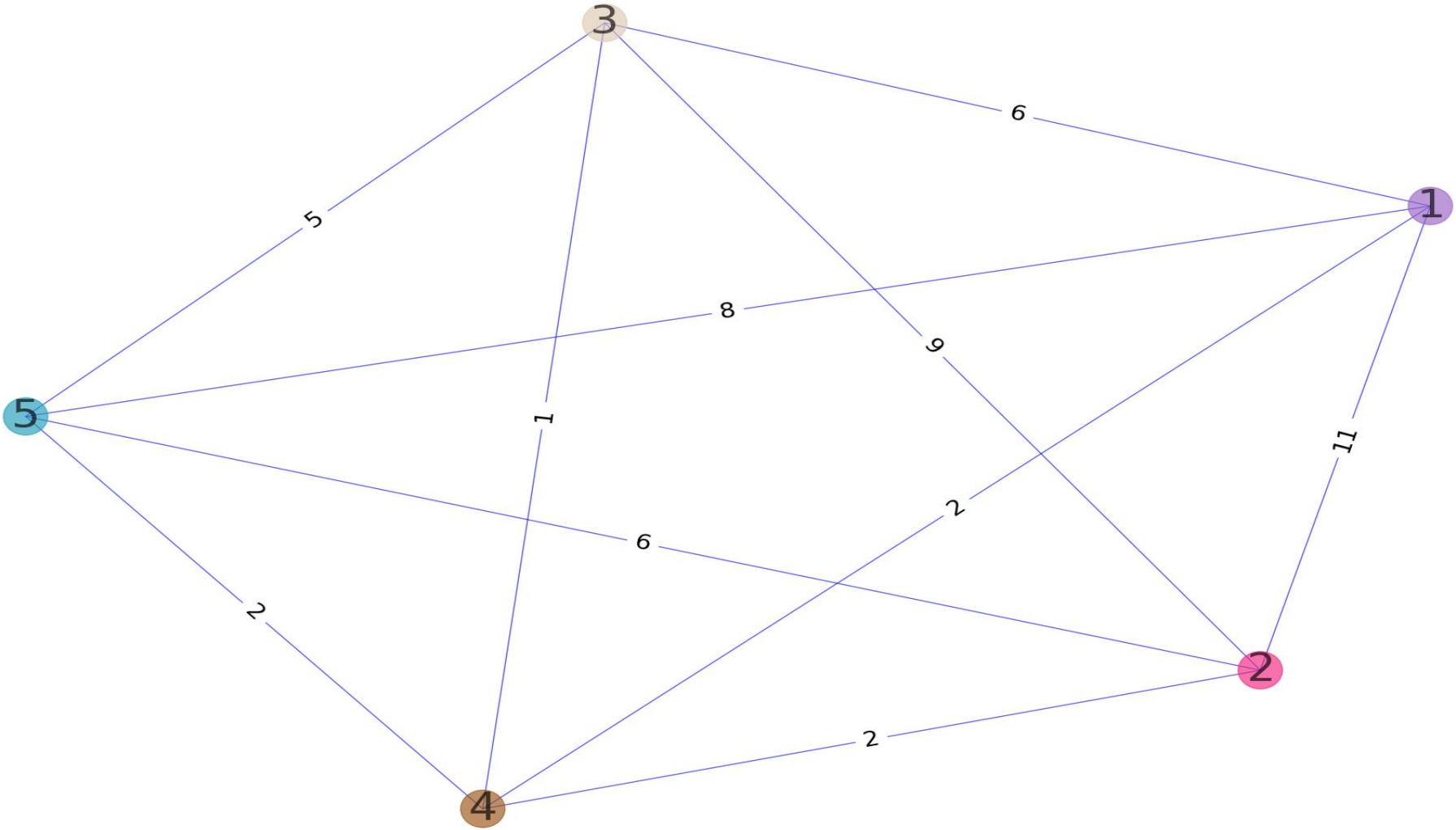
La implementación propuesta para este problema puede ser encontrada junto a todos los archivos necesarios para su ejecución en el siguiente enlace: https://github.com/albertopastormr/mog. En esta url se encuentra además una implementación del algoritmo **branch and bound** utilizando el lenguaje de programación **C++**.

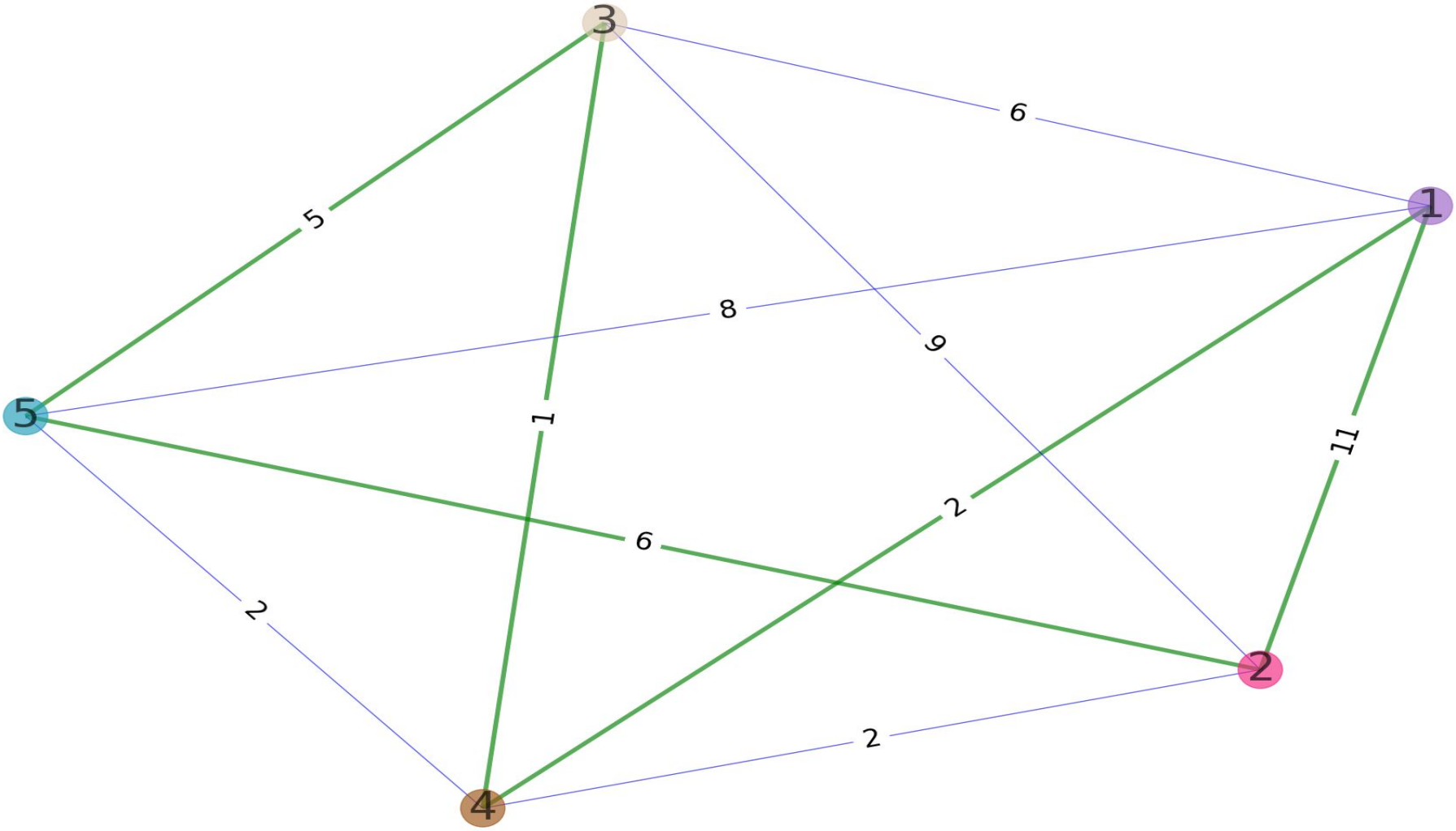


matplotlib

NetworkX

```python
def generate_tsp_path(G, origin):
    tour = []
    nodes_visited = [origin]
    actual_node = origin
    while(len(nodes_visited) < G.number_of_nodes()):
        min_distance = float('inf')
        for n in G.neighbors(actual_node):
            if G[actual_node][n]['weight'] < min_distance and n not in nodes_visited
             or n == origin and len(nodes_visited) == G.number_of_nodes()-1:
                min_distance = G[actual_node][n]['weight']
                next_node = n
                next_edge = (actual_node,next_node)
        nodes_visited.append(actual_node)
        tour.append(next_edge)
        actual_node = next_node
    tour.append((actual_node,origin))
    return tour
```


3
6
1
5
8
9
1
11
6
2
2
2
2
4


3
6
1
5
8
9
1
11
6
2
2
2
2
4

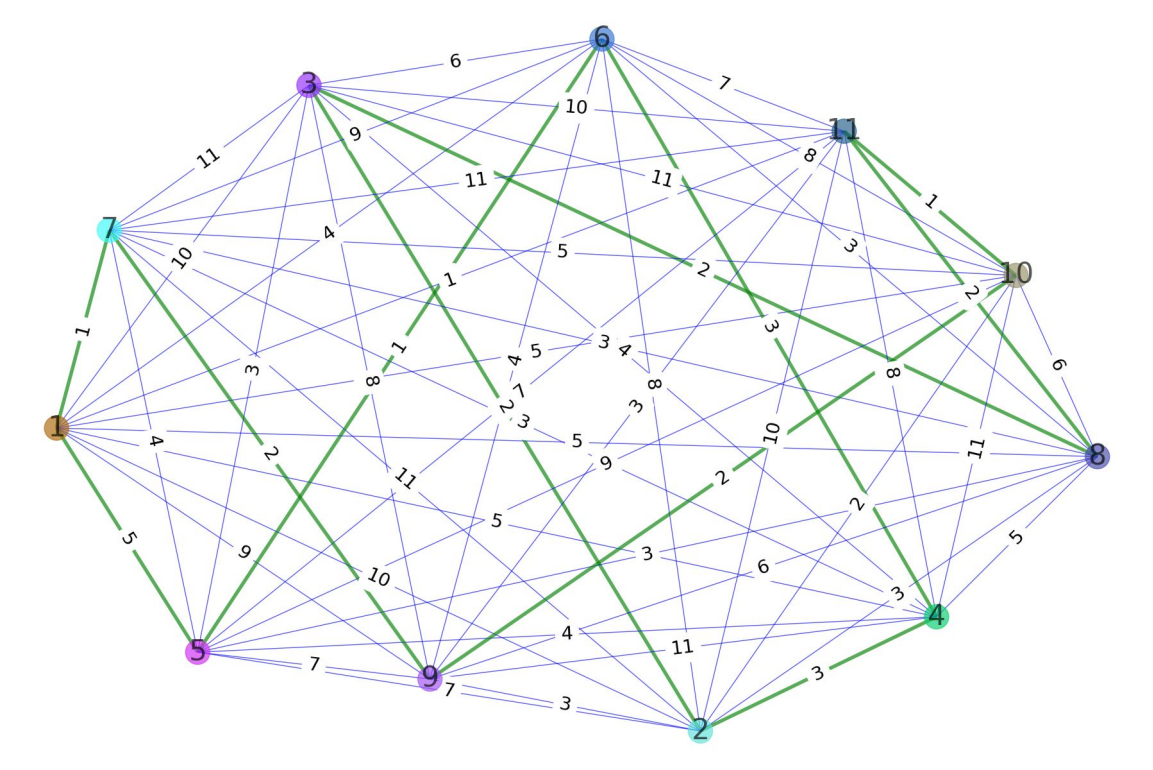
Implementación

```
1  void caminoMinimo(const int &start,vector<int> &sol,vector<int> &solFinal, const int &V, const Matriz<int> &distancias,
2          vector<bool> &marcas, int &contVert, int &k, int &suma, int &sumaMin) {
3
4          for (int i = 0; i < V; i++) {
5                  if (distancias[k][i] != 0 && distancias[k][i] != -1) {
6                          if (i == start && contVert == V) {
7                                  sol[contVert] = i + 1;
8                                  suma += distancias[k][i];
9                                  contVert++;
10
11                                 if (suma <= sumaMin) {
12                                         sumaMin = suma;
13                                         suma = 0;
14
15                                         for (int i = 0; i < sol.size(); i++) {
16                                                 solFinal[i] = sol[i];
17                                         }
18                                 }
19                         }
```

Implementación

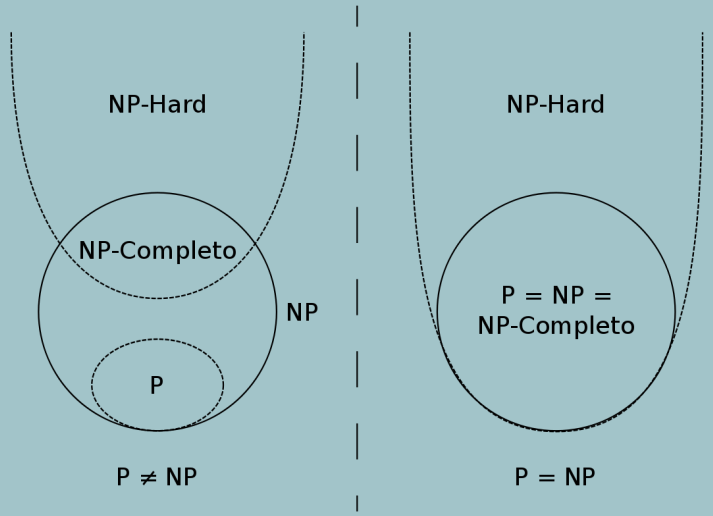
```
20                      else if (!marcas[i]) {
21                              marcas[i] = true;
22                              sol[contVert] = i + 1;
23                              contVert++;
24                              suma += distancias[k][i];
25
26                              caminoMinimo(start, sol,solFinal, V, distancias, marcas, contVert, i, suma, sumaMin);
27
28                              marcas[i] = false;
29                              contVert--;
30                              suma -= distancias[k][i];
31
32                      }
33              }
34      }
35  }
```

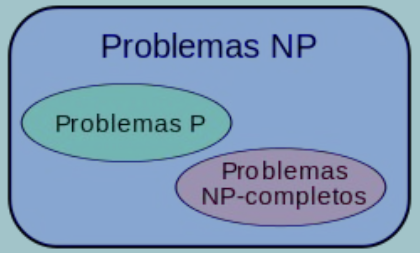
Implementación



Complejidad computacional

- Problemas NP-Completos
- Categorización TSP

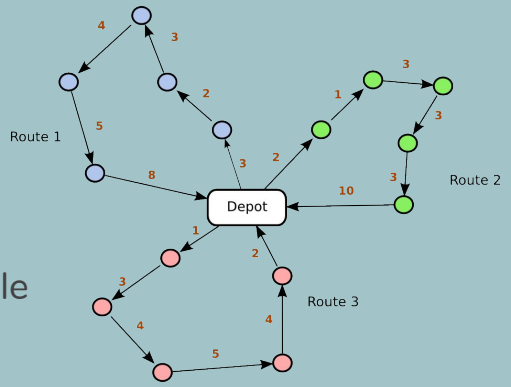




Variantes

Podemos plantear una nueva situación en la que no sea su ciente con un vehículo. Las nuevas restricciones dan lugar a nuevos problemas de ruta. Distinguimos algunas de las principales:

- Problemas de rutas de vehículos con capacidades idénticas (CVRP, Capacitated Vehicle Routing Problem).
- Problema con recogidas (VRPB, Vehicle Routing Problem with Backhauls).
- Problemas de vehiculos con ventanas horarias (VRPTW, Vehicle Routing Problem with Time Windows).
- Problemas con múltiples almacenes(VRPMD, Vehicle Routing Problem with Multiple Depots)



Todas estas variantes del problema del viajante pueden formularse añadiendo restricciones y variables nuevas al modelo original del TSP.

Conclusión

Podemos concluir que este problema ha sido muy relevante históricamente pero aún sigue siendo crucial para resolver algunas de las problemáticas que más preocupación causan mundialmente.(TSPLIB)

Consideramos que este problema podría satisfacer la curiosidad científica de muchos futuros investigadores.

FIN

GRACIAS POR ESCUCHAR