Travelling Salesman Problem

Equipo 1:

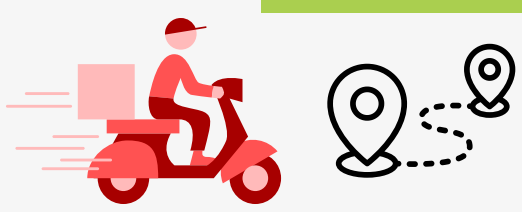
A01721364 Bernardo Jesus Lozano Lozano

A01654132 Ricardo Camacho Castillo

A01025267 Diego Hermosillo Palomo

A01412539 Jose Pablo Sánchez González

A01411597 Karen González Ugalde



RESUMEN

Se compararon distintas soluciones al problema de TSP.
Se prueba la solución exacta, un algoritmo heurístico y un algoritmo genético (metaheurístico).
Los lugares seleccionados como destinos (j) son lugares relevantes en la zona metropolitana de Monterrey

PROBLEMÁTICA

Hoy en día la demanda de entregas de compras en línea ha aumentado exponencialmente, por lo que las empresas han estado implementado nuevos métodos para eficientizar estos procesos.

MARCO TEÓRICO (TSP)

Dado un conjunto de ciudades y la distancia entre cada par, el problema es encontrar la ruta más corta posible en la que visite cada ciudad exactamente una vez y regrese al punto de partida.

Variable de decisión:

$$x_{ij} = \begin{cases} 1 & si\ el\ destino\ j\ es\ visitado\ después\ de\ i \\ 0 & otra\ decisión \end{cases}$$

$$n = num.\,de\ destinos\ a\ visitar = \{1,2,3,\dots,n\}$$

Función Objetivo:

$$min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

sujeto a:

$$\sum_{i=0,i\neq j}^{n} x_{ij} = 1 \qquad j = 0,\dots,n$$

$$\sum_{j=0,j\neq i}^{n} x_{ij} = 1 \qquad i = 0,\dots,n$$

SOLUCIÓN EXACTA

Ruta:
1-14-10-9-12-5-6-7-11-3-2-8-16-4-15-13.

Costo min:
74.55 km

Tiempo de ejecución:
2.217 seg

```
option optcr = 0.0001;

set
i/1*16/ ;
alias (i,j);

Table
c(i,j)
     1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
1  999  6.7  6.4  7.3 11.7 11.4  6.2 10.3 16.4 13.4    8   19  5.7  4.2  4.2  3.4
2  6.7  999  1.7  5.5  9.5  9.2  2.7  2.8 15.3 11.9  2.1 14.8 10.8  5.6  9.3    4
3  6.4  1.7  999  4.2  8.4  8.2  1.9  4.1 13.1 10.9  0.4   14 13.1  4.6 10.6  3.8
4  7.3  5.5  4.2  999 10.5 10.3    4  2.8 15.2 12.2  4.5 17.7 10.1    3  6.2  1.7
5 11.7  9.5  8.4 10.5  999  0.7  7.3 11.3  7.6  7.3  8.6 11.2 16.4  8.6 17.9  9.5
6 11.4  9.2  8.2 10.3  0.7  999  6.7 10.8  8.1  6.7    7 11.6 15.8    8 17.3  8.9
7  6.2  2.7  1.9    4  7.3  6.7  999  4.1 12.4 10.2 0.35 13.7 12.4  3.9 10.1  3.1
8 10.3  2.8  4.1  2.8 11.3 10.8  4.1  999 14.1 11.7  3.9 16.4  9.2  3.1  7.7  1.5
9 16.4 15.3 13.1 15.2  7.6  8.1 12.4 14.1  999  5.2 13.1 14.8 21.9 13.1 23.4   15
10 13.4 11.9 10.9 12.2  7.3  6.7 10.2 11.7  5.2  999  8.3 14.2 16.9    8 18.3   10
11    8  2.1  0.4  4.5  8.6    7 0.35  3.9 13.1  8.3  999 13.6 11.9  4.6 10.4  3.5
12   19 14.8   14 17.7 11.2 11.6 13.7 16.4 14.8 14.2 13.6  999 25.3 16.7 22.9 15.9
13  5.7 10.8 13.1 10.1 16.4 15.8 12.4  9.2 21.9 16.9 11.9 25.3  999  9.8  3.4 10.1
14  4.2  5.6  4.6    3  8.6    8  3.9  3.1 13.1    8  4.6 16.7  9.8  999  8.4  2.2
15  4.2  9.3 10.6  6.2 17.9 17.3 10.1  7.7 23.4 18.3 10.4 22.9  3.4  8.4  999    8
16  3.4    4  3.8  1.7  9.5  8.9  3.1  1.5   15   10  3.5 15.9 10.1  2.2    8  999
;

Variable
z
u(i);
Binary Variable
x(i,j);

Equations
obj
r1
r2
r3;

obj.. z =E= sum((i,j), c(i,j)*x(i,j));
r1(i).. sum(j, x(i,j)) =E= 1;
r2(j).. sum(i, x(i,j)) =E= 1;
r3(i,j) $(ord(i)>1 and ord(j)>1 and ord(i) <> ord(j)).. u(i)-u(j)+card(i)*x(i,j)=L=card(i)-1;

model tsp_reto /all/;

solve tsp_reto using MIP min z;
```

```
---- VAR x

            LOWER          LEVEL          UPPER         MARGINAL

1 .1            .              .         1.0000       999.0000
1 .2            .              .         1.0000         6.7000
1 .3            .              .         1.0000         6.4000
1 .4            .              .         1.0000         7.3000
1 .5            .              .         1.0000        11.7000
1 .6            .              .         1.0000        11.4000
1 .7            .              .         1.0000         6.2000
1 .8            .              .         1.0000        10.3000
1 .9            .              .         1.0000        16.4000
1 .10           .              .         1.0000        13.4000
1 .11           .              .         1.0000         8.0000
1 .12           .              .         1.0000        19.0000
1 .13           .              .         1.0000         5.7000
1 .14           .         1.0000         1.0000         4.2000
1 .15           .              .         1.0000         4.2000
1 .16           .              .         1.0000         3.4000
```

```
                            LOWER          LEVEL

---- VAR z                 -INF           74.5500
```

ALGORITMO HEURÍSTICO

```
import numpy as np
import pandas as pd
from python_tsp.heuristics import solve_tsp_local_search


df = pd.read_csv("Distancias.csv")

dicc = df[["Lugar"]].copy()

df = df.drop(["Lugar"],axis = 1)

lista = df.to_numpy()

lista[lista == 0] = 1000

dicc = dicc.to_numpy()

permutation, distance = solve_tsp_local_search(lista)
print("Distancia recorrida: ",distance, "\n")


for i in range(len(permutation)):
  permutation[i] = dicc[permutation[i]]

dff = pd.DataFrame(permutation, columns = ['Orden de solucion'])
```

- El algoritmo que se utilizo fue busqueda local, este es un heuristico por lo cual no forzosamente encontrara el optimo. El algoritmo de busqueda local tiene como funcion objetivo la minimizacion de distancias inmediatas, esto lo hace mas susceptible a atraparse en optimos locales.
- En la simulacion realizada se llego al optimo local de 76.95, esto varia cada simulacion debido a la naturaleza del algoritmo utilizado.

```
Distancia recorrida:  76.94999999999999
```

	Orden de solucion
0	HEB San Pedro
1	Punto Valle
2	Taquería Orinoco
3	H. San José
4	Parque Fundidora
5	Estadio BBVA
6	Tec de MTY
7	Paseo Tec
8	Super Salads Esfera
9	Fashion Drive
10	Hospital Zambrano Hellion
11	Colegio Irlandés
12	CECVAC
13	McKinsey
14	Prepa Tec Santa Catarina
15	Colegio Himalaya

ALGORITMO GENÉTICO

```
# seleccionamos al ganador
def seleccion(population, evaluation, tournamentSize):
  winner = np.random.randint(0, len(population))
  for i in range(tournamentSize - 1):
    rival = np.random.randint(0, len(population))
    if (evaluation[rival] < evaluation[winner]):
      winner = rival
  return population[winner]

# Juntamos todas las funciones
def geneticAlgorithm(populationSize, cRate, mRate, generations):
  # Creates the initial population (it also evaluates it)
  population = [None] * populationSize
  evaluation = [None] * populationSize
  for i in range(populationSize):
    individual = crearIndividuo()
    population[i] = individual
    evaluation[i] = evaluar(individual)

  index = 0;
  for i in range(1, populationSize):
    if (evaluation[i] < evaluation[index]):
      index = i;
  bestIndividual = population[index]
  bestEvaluation = evaluation[index]
  # Runs the evolutionary process
  for i in range(generations):
    k = 0
    newPopulation = [None] * populationSize
    for j in range(populationSize // 2):
      parentA = seleccion(population, evaluation, 3)
      parentB = seleccion(population, evaluation, 3)
      newPopulation[k], newPopulation[k + 1] = cruzamiento(parentA, parentB, cRate)
      k = k + 2
    population = newPopulation
    for j in range(populationSize):
      population[j] = mutacion(population[j], mRate)
      evaluation[j] = evaluar(population[j])
      # mantiene un registro del mejor individuo hasta el momento
      if (evaluation[j] < bestEvaluation):
        bestEvaluation = evaluation[j]
        bestIndividual = population[j]
  return bestIndividual, bestEvaluation
```

- Generación de la población: se crean cromosomas con el número de la ciudad en orden aleatorio, siempre empezando por el origen siempre y cuando sea factible.
- Selección: mini torneo entre muestras de la población seleccionando al mejor de ellos. Se hace dos veces para obtener dos padres.
- Cruzamiento: Se toma un número aleatorio y se hace un corte, se toma la primera parte de un padre y la segunda del otro y se valida que sea una solución factible.
- Mutación: si un número aleatorio entra dentro de la probabilidad de mutación, se muta y se intercambian dos posiciones aleatorias del individuo.
- Evaluación: Se usa la función fitness, la cual se usa a lo largo de todo el algoritmo y valida que la ruta sea solución factible tomando en cuenta las conexiones de las ciudades y que no se repitan ciudades y siempre se pueda regresar al origen.

```
Tiempo total de ejecución
3.0233561992645264
```

```
Ruta a seguir: [ 1 16  7 11 10  9 12  5  6 14  4 15 13  8  2  3]
['HEB San Pedro', 'Punto Valle', 'Fashion Drive', 'H. Zambrano Hellion', 'Parque Fundidora', 'Estadio BBVA',
'Super Salads Esfera', 'Tec de MTY', 'Paseo Tec', 'H. San José', 'Taquería Orinoco', 'Colegio Himalaya',
'Prepa Tec Santa Catarina', 'McKinskey', 'CECVAC', 'Colegio Irlandés']
Distancia mínima: 87.75000000000003
```

Referencias

https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/
https://optimization.mccormick.northwestern.edu/index.php/Traveling_salesman_problems