AG3 - Actividad Guiada 3

Nombre: Luis Enrique Sanchez Zamora
URL: https://colab.research.google.com/drive/1zrEVXhjAXc7A0XrTYJmbd_9QF4mUYXK3?usp=sharing

Repositorio: https://github.com/sanieni6/03MIAR---Algoritmos-de-Optimizacion

Carga de librerias

In [1]:

```
!pip install requests    #Hacer llamadas http a paginas de la red
!pip install tsplib95    #Modulo para las instancias del problema del TSP
```

```
Requirement already satisfied: requests in /opt/anaconda3/envs/viuenv/lib/python3.11/site
-packages (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anaconda3/envs/viuenv/lib
/python3.11/site-packages (from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/envs/viuenv/lib/python3.11/
site-packages (from requests) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/envs/viuenv/lib/pytho
n3.11/site-packages (from requests) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/envs/viuenv/lib/pytho
n3.11/site-packages (from requests) (2025.6.15)
Requirement already satisfied: tsplib95 in /opt/anaconda3/envs/viuenv/lib/python3.11/site
-packages (0.7.1)
Requirement already satisfied: Click>=6.0 in /opt/anaconda3/envs/viuenv/lib/python3.11/si
te-packages (from tsplib95) (8.2.1)
Requirement already satisfied: Deprecated~=1.2.9 in /opt/anaconda3/envs/viuenv/lib/python
3.11/site-packages (from tsplib95) (1.2.18)
Requirement already satisfied: networkx~=2.1 in /opt/anaconda3/envs/viuenv/lib/python3.11
/site-packages (from tsplib95) (2.8.8)
Requirement already satisfied: tabulate~=0.8.7 in /opt/anaconda3/envs/viuenv/lib/python3.
11/site-packages (from tsplib95) (0.8.10)
Requirement already satisfied: wrapt<2,>=1.10 in /opt/anaconda3/envs/viuenv/lib/python3.1
1/site-packages (from Deprecated~=1.2.9->tsplib95) (1.17.2)
```

Carga de los datos del problema

In [2]:

```
import urllib.request #Hacer llamadas http a paginas de la red
import tsplib95       #Modulo para las instancias del problema del TSP
import math           #Modulo de funciones matematicas. Se usa para exp
import random         #Para generar valores aleatorios


#http://elib.zib.de/pub/mp-testdata/tsp/tsplib/
#Documentacion :
  # http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf
  # https://tsplib95.readthedocs.io/en/stable/pages/usage.html
  # https://tsplib95.readthedocs.io/en/v0.6.1/modules.html
  # https://pypi.org/project/tsplib95/

#Descargamos el fichero de datos(Matriz de distancias)
file = "swiss42.tsp" ;
urllib.request.urlretrieve("http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/swi
ss42.tsp.gz", file + '.gz')
!gzip -d swiss42.tsp.gz     #Descomprimir el fichero de datos

#Coordendas 51-city problem (Christofides/Eilon)
#file = "eil51.tsp" ; urllib.request.urlretrieve("http://comopt.ifi.uni-heidelberg.de/sof
tware/TSPLIB95/tsp/eil51.tsp.gz", file)
```

```
#Coordenadas - 48 capitals of the US (Padberg/Rinaldi)
#file = "att48.tsp" ; urllib.request.urlretrieve("http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/att48.tsp.gz", file)
```

```
swiss42.tsp already exists -- do you wish to overwrite (y or n)? ^C
```

In [3]:

```
#Carga de datos y generación de objeto problem
###############################################################################
problem = tsplib95.load(file)

#Nodos
Nodos = list(problem.get_nodes())

#Aristas
Aristas = list(problem.get_edges())
```

In [4]:

```
Aristas
```

Out[4]:

```
[(0, 0),
 (0, 1),
 (0, 2),
 (0, 3),
 (0, 4),
 (0, 5),
 (0, 6),
 (0, 7),
 (0, 8),
 (0, 9),
 (0, 10),
 (0, 11),
 (0, 12),
 (0, 13),
 (0, 14),
 (0, 15),
 (0, 16),
 (0, 17),
 (0, 18),
 (0, 19),
 (0, 20),
 (0, 21),
 (0, 22),
 (0, 23),
 (0, 24),
 (0, 25),
 (0, 26),
 (0, 27),
 (0, 28),
 (0, 29),
 (0, 30),
 (0, 31),
 (0, 32),
 (0, 33),
 (0, 34),
 (0, 35),
 (0, 36),
 (0, 37),
 (0, 38),
 (0, 39),
 (0, 40),
 (0, 41),
 (1, 0),
 (1, 1),
 (1, 2),
 (1, 3),
 (1, 4),
```

```
(1, 5),
(1, 6),
(1, 7),
(1, 8),
(1, 9),
(1, 10),
(1, 11),
(1, 12),
(1, 13),
(1, 14),
(1, 15),
(1, 16),
(1, 17),
(1, 18),
(1, 19),
(1, 20),
(1, 21),
(1, 22),
(1, 23),
(1, 24),
(1, 25),
(1, 26),
(1, 27),
(1, 28),
(1, 29),
(1, 30),
(1, 31),
(1, 32),
(1, 33),
(1, 34),
(1, 35),
(1, 36),
(1, 37),
(1, 38),
(1, 39),
(1, 40),
(1, 41),
(2, 0),
(2, 1),
(2, 2),
(2, 3),
(2, 4),
(2, 5),
(2, 6),
(2, 7),
(2, 8),
(2, 9),
(2, 10),
(2, 11),
(2, 12),
(2, 13),
(2, 14),
(2, 15),
(2, 16),
(2, 17),
(2, 18),
(2, 19),
(2, 20),
(2, 21),
(2, 22),
(2, 23),
(2, 24),
(2, 25),
(2, 26),
(2, 27),
(2, 28),
(2, 29),
(2, 30),
(2, 31),
(2, 32),
(2, 33),
(2, 34),
```

```
(2, 35),
(2, 36),
(2, 37),
(2, 38),
(2, 39),
(2, 40),
(2, 41),
(3, 0),
(3, 1),
(3, 2),
(3, 3),
(3, 4),
(3, 5),
(3, 6),
(3, 7),
(3, 8),
(3, 9),
(3, 10),
(3, 11),
(3, 12),
(3, 13),
(3, 14),
(3, 15),
(3, 16),
(3, 17),
(3, 18),
(3, 19),
(3, 20),
(3, 21),
(3, 22),
(3, 23),
(3, 24),
(3, 25),
(3, 26),
(3, 27),
(3, 28),
(3, 29),
(3, 30),
(3, 31),
(3, 32),
(3, 33),
(3, 34),
(3, 35),
(3, 36),
(3, 37),
(3, 38),
(3, 39),
(3, 40),
(3, 41),
(4, 0),
(4, 1),
(4, 2),
(4, 3),
(4, 4),
(4, 5),
(4, 6),
(4, 7),
(4, 8),
(4, 9),
(4, 10),
(4, 11),
(4, 12),
(4, 13),
(4, 14),
(4, 15),
(4, 16),
(4, 17),
(4, 18),
(4, 19),
(4, 20),
(4, 21),
(4, 22),
```

(4, 23),
(4, 24),
(4, 25),
(4, 26),
(4, 27),
(4, 28),
(4, 29),
(4, 30),
(4, 31),
(4, 32),
(4, 33),
(4, 34),
(4, 35),
(4, 36),
(4, 37),
(4, 38),
(4, 39),
(4, 40),
(4, 41),
(5, 0),
(5, 1),
(5, 2),
(5, 3),
(5, 4),
(5, 5),
(5, 6),
(5, 7),
(5, 8),
(5, 9),
(5, 10),
(5, 11),
(5, 12),
(5, 13),
(5, 14),
(5, 15),
(5, 16),
(5, 17),
(5, 18),
(5, 19),
(5, 20),
(5, 21),
(5, 22),
(5, 23),
(5, 24),
(5, 25),
(5, 26),
(5, 27),
(5, 28),
(5, 29),
(5, 30),
(5, 31),
(5, 32),
(5, 33),
(5, 34),
(5, 35),
(5, 36),
(5, 37),
(5, 38),
(5, 39),
(5, 40),
(5, 41),
(6, 0),
(6, 1),
(6, 2),
(6, 3),
(6, 4),
(6, 5),
(6, 6),
(6, 7),
(6, 8),
(6, 9),
(6, 10),
(6, 11)

(6, 11),
(6, 12),
(6, 13),
(6, 14),
(6, 15),
(6, 16),
(6, 17),
(6, 18),
(6, 19),
(6, 20),
(6, 21),
(6, 22),
(6, 23),
(6, 24),
(6, 25),
(6, 26),
(6, 27),
(6, 28),
(6, 29),
(6, 30),
(6, 31),
(6, 32),
(6, 33),
(6, 34),
(6, 35),
(6, 36),
(6, 37),
(6, 38),
(6, 39),
(6, 40),
(6, 41),
(7, 0),
(7, 1),
(7, 2),
(7, 3),
(7, 4),
(7, 5),
(7, 6),
(7, 7),
(7, 8),
(7, 9),
(7, 10),
(7, 11),
(7, 12),
(7, 13),
(7, 14),
(7, 15),
(7, 16),
(7, 17),
(7, 18),
(7, 19),
(7, 20),
(7, 21),
(7, 22),
(7, 23),
(7, 24),
(7, 25),
(7, 26),
(7, 27),
(7, 28),
(7, 29),
(7, 30),
(7, 31),
(7, 32),
(7, 33),
(7, 34),
(7, 35),
(7, 36),
(7, 37),
(7, 38),
(7, 39),
(7, 40),
(7, 41)

(7, 41),
(8, 0),
(8, 1),
(8, 2),
(8, 3),
(8, 4),
(8, 5),
(8, 6),
(8, 7),
(8, 8),
(8, 9),
(8, 10),
(8, 11),
(8, 12),
(8, 13),
(8, 14),
(8, 15),
(8, 16),
(8, 17),
(8, 18),
(8, 19),
(8, 20),
(8, 21),
(8, 22),
(8, 23),
(8, 24),
(8, 25),
(8, 26),
(8, 27),
(8, 28),
(8, 29),
(8, 30),
(8, 31),
(8, 32),
(8, 33),
(8, 34),
(8, 35),
(8, 36),
(8, 37),
(8, 38),
(8, 39),
(8, 40),
(8, 41),
(9, 0),
(9, 1),
(9, 2),
(9, 3),
(9, 4),
(9, 5),
(9, 6),
(9, 7),
(9, 8),
(9, 9),
(9, 10),
(9, 11),
(9, 12),
(9, 13),
(9, 14),
(9, 15),
(9, 16),
(9, 17),
(9, 18),
(9, 19),
(9, 20),
(9, 21),
(9, 22),
(9, 23),
(9, 24),
(9, 25),
(9, 26),
(9, 27),
(9, 28),

(9, 29),
(9, 30),
(9, 31),
(9, 32),
(9, 33),
(9, 34),
(9, 35),
(9, 36),
(9, 37),
(9, 38),
(9, 39),
(9, 40),
(9, 41),
(10, 0),
(10, 1),
(10, 2),
(10, 3),
(10, 4),
(10, 5),
(10, 6),
(10, 7),
(10, 8),
(10, 9),
(10, 10),
(10, 11),
(10, 12),
(10, 13),
(10, 14),
(10, 15),
(10, 16),
(10, 17),
(10, 18),
(10, 19),
(10, 20),
(10, 21),
(10, 22),
(10, 23),
(10, 24),
(10, 25),
(10, 26),
(10, 27),
(10, 28),
(10, 29),
(10, 30),
(10, 31),
(10, 32),
(10, 33),
(10, 34),
(10, 35),
(10, 36),
(10, 37),
(10, 38),
(10, 39),
(10, 40),
(10, 41),
(11, 0),
(11, 1),
(11, 2),
(11, 3),
(11, 4),
(11, 5),
(11, 6),
(11, 7),
(11, 8),
(11, 9),
(11, 10),
(11, 11),
(11, 12),
(11, 13),
(11, 14),
(11, 15),
(11, 16),
(11, 17),

(11, 17),
(11, 18),
(11, 19),
(11, 20),
(11, 21),
(11, 22),
(11, 23),
(11, 24),
(11, 25),
(11, 26),
(11, 27),
(11, 28),
(11, 29),
(11, 30),
(11, 31),
(11, 32),
(11, 33),
(11, 34),
(11, 35),
(11, 36),
(11, 37),
(11, 38),
(11, 39),
(11, 40),
(11, 41),
(12, 0),
(12, 1),
(12, 2),
(12, 3),
(12, 4),
(12, 5),
(12, 6),
(12, 7),
(12, 8),
(12, 9),
(12, 10),
(12, 11),
(12, 12),
(12, 13),
(12, 14),
(12, 15),
(12, 16),
(12, 17),
(12, 18),
(12, 19),
(12, 20),
(12, 21),
(12, 22),
(12, 23),
(12, 24),
(12, 25),
(12, 26),
(12, 27),
(12, 28),
(12, 29),
(12, 30),
(12, 31),
(12, 32),
(12, 33),
(12, 34),
(12, 35),
(12, 36),
(12, 37),
(12, 38),
(12, 39),
(12, 40),
(12, 41),
(13, 0),
(13, 1),
(13, 2),
(13, 3),
(13, 4),
(13, 5)

(13, 5),
(13, 6),
(13, 7),
(13, 8),
(13, 9),
(13, 10),
(13, 11),
(13, 12),
(13, 13),
(13, 14),
(13, 15),
(13, 16),
(13, 17),
(13, 18),
(13, 19),
(13, 20),
(13, 21),
(13, 22),
(13, 23),
(13, 24),
(13, 25),
(13, 26),
(13, 27),
(13, 28),
(13, 29),
(13, 30),
(13, 31),
(13, 32),
(13, 33),
(13, 34),
(13, 35),
(13, 36),
(13, 37),
(13, 38),
(13, 39),
(13, 40),
(13, 41),
(14, 0),
(14, 1),
(14, 2),
(14, 3),
(14, 4),
(14, 5),
(14, 6),
(14, 7),
(14, 8),
(14, 9),
(14, 10),
(14, 11),
(14, 12),
(14, 13),
(14, 14),
(14, 15),
(14, 16),
(14, 17),
(14, 18),
(14, 19),
(14, 20),
(14, 21),
(14, 22),
(14, 23),
(14, 24),
(14, 25),
(14, 26),
(14, 27),
(14, 28),
(14, 29),
(14, 30),
(14, 31),
(14, 32),
(14, 33),
(14, 34),

(14, 35),
(14, 36),
(14, 37),
(14, 38),
(14, 39),
(14, 40),
(14, 41),
(15, 0),
(15, 1),
(15, 2),
(15, 3),
(15, 4),
(15, 5),
(15, 6),
(15, 7),
(15, 8),
(15, 9),
(15, 10),
(15, 11),
(15, 12),
(15, 13),
(15, 14),
(15, 15),
(15, 16),
(15, 17),
(15, 18),
(15, 19),
(15, 20),
(15, 21),
(15, 22),
(15, 23),
(15, 24),
(15, 25),
(15, 26),
(15, 27),
(15, 28),
(15, 29),
(15, 30),
(15, 31),
(15, 32),
(15, 33),
(15, 34),
(15, 35),
(15, 36),
(15, 37),
(15, 38),
(15, 39),
(15, 40),
(15, 41),
(16, 0),
(16, 1),
(16, 2),
(16, 3),
(16, 4),
(16, 5),
(16, 6),
(16, 7),
(16, 8),
(16, 9),
(16, 10),
(16, 11),
(16, 12),
(16, 13),
(16, 14),
(16, 15),
(16, 16),
(16, 17),
(16, 18),
(16, 19),
(16, 20),
(16, 21),
(16, 22),
(16, 23),

(16, 23),
(16, 24),
(16, 25),
(16, 26),
(16, 27),
(16, 28),
(16, 29),
(16, 30),
(16, 31),
(16, 32),
(16, 33),
(16, 34),
(16, 35),
(16, 36),
(16, 37),
(16, 38),
(16, 39),
(16, 40),
(16, 41),
(17, 0),
(17, 1),
(17, 2),
(17, 3),
(17, 4),
(17, 5),
(17, 6),
(17, 7),
(17, 8),
(17, 9),
(17, 10),
(17, 11),
(17, 12),
(17, 13),
(17, 14),
(17, 15),
(17, 16),
(17, 17),
(17, 18),
(17, 19),
(17, 20),
(17, 21),
(17, 22),
(17, 23),
(17, 24),
(17, 25),
(17, 26),
(17, 27),
(17, 28),
(17, 29),
(17, 30),
(17, 31),
(17, 32),
(17, 33),
(17, 34),
(17, 35),
(17, 36),
(17, 37),
(17, 38),
(17, 39),
(17, 40),
(17, 41),
(18, 0),
(18, 1),
(18, 2),
(18, 3),
(18, 4),
(18, 5),
(18, 6),
(18, 7),
(18, 8),
(18, 9),
(18, 10),
(18, 11)

(18, 11),
(18, 12),
(18, 13),
(18, 14),
(18, 15),
(18, 16),
(18, 17),
(18, 18),
(18, 19),
(18, 20),
(18, 21),
(18, 22),
(18, 23),
(18, 24),
(18, 25),
(18, 26),
(18, 27),
(18, 28),
(18, 29),
(18, 30),
(18, 31),
(18, 32),
(18, 33),
(18, 34),
(18, 35),
(18, 36),
(18, 37),
(18, 38),
(18, 39),
(18, 40),
(18, 41),
(19, 0),
(19, 1),
(19, 2),
(19, 3),
(19, 4),
(19, 5),
(19, 6),
(19, 7),
(19, 8),
(19, 9),
(19, 10),
(19, 11),
(19, 12),
(19, 13),
(19, 14),
(19, 15),
(19, 16),
(19, 17),
(19, 18),
(19, 19),
(19, 20),
(19, 21),
(19, 22),
(19, 23),
(19, 24),
(19, 25),
(19, 26),
(19, 27),
(19, 28),
(19, 29),
(19, 30),
(19, 31),
(19, 32),
(19, 33),
(19, 34),
(19, 35),
(19, 36),
(19, 37),
(19, 38),
(19, 39),
(19, 40),
(19, 41)

(19, 41),
(20, 0),
(20, 1),
(20, 2),
(20, 3),
(20, 4),
(20, 5),
(20, 6),
(20, 7),
(20, 8),
(20, 9),
(20, 10),
(20, 11),
(20, 12),
(20, 13),
(20, 14),
(20, 15),
(20, 16),
(20, 17),
(20, 18),
(20, 19),
(20, 20),
(20, 21),
(20, 22),
(20, 23),
(20, 24),
(20, 25),
(20, 26),
(20, 27),
(20, 28),
(20, 29),
(20, 30),
(20, 31),
(20, 32),
(20, 33),
(20, 34),
(20, 35),
(20, 36),
(20, 37),
(20, 38),
(20, 39),
(20, 40),
(20, 41),
(21, 0),
(21, 1),
(21, 2),
(21, 3),
(21, 4),
(21, 5),
(21, 6),
(21, 7),
(21, 8),
(21, 9),
(21, 10),
(21, 11),
(21, 12),
(21, 13),
(21, 14),
(21, 15),
(21, 16),
(21, 17),
(21, 18),
(21, 19),
(21, 20),
(21, 21),
(21, 22),
(21, 23),
(21, 24),
(21, 25),
(21, 26),
(21, 27),
(21, 28),

(21, 29),
(21, 30),
(21, 31),
(21, 32),
(21, 33),
(21, 34),
(21, 35),
(21, 36),
(21, 37),
(21, 38),
(21, 39),
(21, 40),
(21, 41),
(22, 0),
(22, 1),
(22, 2),
(22, 3),
(22, 4),
(22, 5),
(22, 6),
(22, 7),
(22, 8),
(22, 9),
(22, 10),
(22, 11),
(22, 12),
(22, 13),
(22, 14),
(22, 15),
(22, 16),
(22, 17),
(22, 18),
(22, 19),
(22, 20),
(22, 21),
(22, 22),
(22, 23),
(22, 24),
(22, 25),
(22, 26),
(22, 27),
(22, 28),
(22, 29),
(22, 30),
(22, 31),
(22, 32),
(22, 33),
(22, 34),
(22, 35),
(22, 36),
(22, 37),
(22, 38),
(22, 39),
(22, 40),
(22, 41),
(23, 0),
(23, 1),
(23, 2),
(23, 3),
(23, 4),
(23, 5),
(23, 6),
(23, 7),
(23, 8),
(23, 9),
(23, 10),
(23, 11),
(23, 12),
(23, 13),
(23, 14),
(23, 15),
(23, 16),

```
 (23, 17),
 (23, 18),
 (23, 19),
 (23, 20),
 (23, 21),
 (23, 22),
 (23, 23),
 (23, 24),
 (23, 25),
 (23, 26),
 (23, 27),
 (23, 28),
 (23, 29),
 (23, 30),
 (23, 31),
 (23, 32),
 (23, 33),
 ...]
```

```
NOMBRE: swiss42
TIPO: TSP
COMENTARIO: 42 Staedte Schweiz (Fricker)
DIMENSION: 42
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_SECTION
0 15 30 23 32 55 33 37 92 114 92 110 96 90 74 76 82 72 78 82 159 122 131 206 112 57 28 43 70 (
  15 0 34 23 27 40 19 32 93 117 88 100 87 75 63 67 71 69 62 63 96 164 132 131 212 106 44 33 5
  30 34 0 11 18 57 36 65 62 84 64 89 76 93 95 100 104 98 57 88 99 130 100 101 179 86 51 4 18 4
  23 23 11 0 11 48 26 54 70 94 69 75 75 84 84 89 92 89 54 78 99 141 111 109 89 89 11 11 11 54
  32 27 18 11 0 40 20 58 67 92 61 78 65 76 83 89 91 95 43 72 110 141 116 105 190 81 34 19 35 5
  55 40 57 48 40 0 23 55 96 123 78 75 36 36 66 66 63 95 34 34 137 174 156 129 224 90 15 59 75
  33 19 36 26 20 23 0 45 85 111 75 82 69 60 63 70 71 85 44 52 115 161 136 122 210 91 25 37 54
  37 32 65 54 58 55 45 0 124 149 118 126 113 80 42 42 40 40 87 87 94 158 158 163 242 135 65 6
  92 93 62 70 67 96 85 124 0 28 29 68 63 122 148 155 156 159 67 129 148 78 80 39 129 46 82 65
 114 117 84 94 92 123 111 149 28 0 54 91 88 150 174 181 182 181 95 157 159 50 65 27 102 65 11
  92 88 64 69 61 78 75 118 29 54 0 39 34 99 134 142 141 157 44 110 161 103 109 52 154 22 63 6
 110 100 89 89 78 75 82 126 68 91 39 0 14 80 129 139 135 167 39 98 187 136 148 81 186 28 61 9
  96 87 76 75 65 62 69 113 63 88 34 14 0 72 117 128 124 153 26 88 174 136 142 82 187 32 48 79
  90 75 93 84 76 36 60 80 122 150 99 80 72 0 59 71 63 116 56 25 170 201 189 151 252 104 44 95
  74 63 95 84 83 56 63 42 148 174 134 129 117 59 0 11 8 63 93 35 135 223 195 184 273 146 71 9
```

In [5]:

```
#Probamos algunas funciones del objeto problem

#Distancia entre nodos
problem.get_weight(0, 1)

#Todas las funciones
#Documentación: https://tsplib95.readthedocs.io/en/v0.6.1/modules.html

#dir(problem)
```

Out[5]:

```
15
```

Funcionas basicas

In [6]:

```
#Funcionas basicas
################################################################################

#Se genera una solucion aleatoria con comienzo en en el nodo 0
def crear_solucion(Nodos):
  solucion = [Nodos[0]]
  for n in Nodos[1:]:
    solucion = solucion + [random.choice(list(set(Nodos) - set({Nodos[0]}) - set(solucio
n)))]
  return solucion
```

```
#Devuelve la distancia entre dos nodos
def distancia(a,b, problem):
  return problem.get_weight(a,b)

#Devuelve la distancia total de una trayectoria/solucion
def distancia_total(solucion, problem):
  distancia_total = 0
  for i in range(len(solucion)-1):
    distancia_total += distancia(solucion[i] ,solucion[i+1] ,  problem)
  return distancia_total + distancia(solucion[len(solucion)-1] ,solucion[0], problem)

sol_temporal = crear_solucion(Nodos)

distancia_total(sol_temporal, problem), sol_temporal
```

Out[6]:

```
(4668,
 [0,
  26,
  19,
  27,
  6,
  24,
  10,
  28,
  16,
  37,
  14,
  40,
  23,
  8,
  29,
  17,
  1,
  31,
  38,
  35,
  4,
  2,
  30,
  13,
  3,
  20,
  25,
  39,
  7,
  34,
  12,
  9,
  22,
  33,
  21,
  36,
  5,
  15,
  11,
  32,
  41,
  18])
```

BUSQUEDA ALEATORIA

In [11]:

```
##############################################################################
# BUSQUEDA ALEATORIA
##############################################################################
```

```
def busqueda_aleatoria(problem, N):
  #N es el numero de iteraciones
  Nodos = list(problem.get_nodes())

  mejor_solucion = []
  #mejor_distancia = 10e100                          #Inicializamos con un valor alto
  mejor_distancia = float('inf')                     #Inicializamos con un valor alto

  for i in range(N):                                 #Criterio de parada: repetir N veces
pero podemos incluir otros
    solucion = crear_solucion(Nodos)                 #Genera una solucion aleatoria
    distancia = distancia_total(solucion, problem)   #Calcula el valor objetivo(distancia
total)

    if distancia < mejor_distancia:                  #Compara con la mejor obtenida hasta
ahora
      mejor_solucion = solucion
      mejor_distancia = distancia


  print("Mejor solución:" , mejor_solucion)
  print("Distancia     :" , mejor_distancia)
  return mejor_solucion


#Busqueda aleatoria con 5000 iteraciones
solucion = busqueda_aleatoria(problem, 10000)
```

```
Mejor solución: [0, 38, 14, 31, 15, 26, 6, 29, 22, 30, 10, 11, 24, 9, 41, 19, 37, 35, 36,
13, 12, 4, 28, 34, 40, 21, 39, 2, 8, 33, 7, 20, 18, 27, 32, 1, 5, 3, 17, 16, 23, 25]
Distancia     : 3714
```

BUSQUEDA LOCAL

In [26]:

```
#############################################################################
# BUSQUEDA LOCAL
#############################################################################
def genera_vecina(solucion):
  #Generador de soluciones vecinas: 2-opt (intercambiar 2 nodos) Si hay N nodos se genera
n (N-1)x(N-2)/2 soluciones
  #Se puede modificar para aplicar otros generadores distintos que 2-opt
  #print(solucion)
  mejor_solucion = []
  mejor_distancia = 10e100
  for i in range(1,len(solucion)-1):           #Recorremos todos los nodos en bucle doble
para evaluar todos los intercambios 2-opt
    for j in range(i+1, len(solucion)):

      #Se genera una nueva solución intercambiando los dos nodos i,j:
      #  (usamos el operador + que para listas en python las concatena) : ej.: [1,2] + [3
] = [1,2,3]
      vecina = solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + solucion
[j+1:]

      #Se evalua la nueva solución ...
      distancia_vecina = distancia_total(vecina, problem)

      #... para guardarla si mejora las anteriores
      if distancia_vecina <= mejor_distancia:
        mejor_distancia = distancia_vecina
        mejor_solucion = vecina
  return mejor_solucion


#solucion = [1, 47, 13, 41, 40, 19, 42, 44, 37, 5, 22, 28, 3, 2, 29, 21, 50, 34, 30, 9, 1
6, 11, 38, 49, 10, 39, 33, 45, 15, 24, 43, 26, 31, 36, 35, 20, 8, 7, 23, 48, 27, 12, 17,
4, 18, 25, 14, 6, 51, 46, 32]
print("Distancia Solucion Incial:" , distancia_total(solucion, problem))
```

```
nueva_solucion = genera_vecina(solucion)
print("Distancia Mejor Solucion Local:", distancia_total(nueva_solucion, problem))
```

```
Distancia Solucion Incial: 3714
Distancia Mejor Solucion Local: 3463
```

In [27]:

```
#Busqueda Local:
#  - Sobre el operador de vecindad 2-opt(funcion genera_vecina)
#  - Sin criterio de parada, se para cuando no es posible mejorar.
def busqueda_local(problem):
  mejor_solucion = []

  #Generar una solucion inicial de referencia(aleatoria)
  solucion_referencia = crear_solucion(Nodos)
  mejor_distancia = distancia_total(solucion_referencia, problem)

  iteracion=0             #Un contador para saber las iteraciones que hacemos
  while(1):
    iteracion +=1         #Incrementamos el contador
    #print('#',iteracion)

    #Obtenemos la mejor vecina ...
    vecina = genera_vecina(solucion_referencia)

    #... y la evaluamos para ver si mejoramos respecto a lo encontrado hasta el momento
    distancia_vecina = distancia_total(vecina, problem)

    #Si no mejoramos hay que terminar. Hemos llegado a un minimo local(según nuestro oper
ador de vencindad 2-opt)
    if distancia_vecina < mejor_distancia:
      #mejor_solucion = copy.deepcopy(vecina)   #Con copia profunda. Las copias en pytho
n son por referencia
      mejor_solucion = vecina                  #Guarda la mejor solución encontrada
      mejor_distancia = distancia_vecina

    else:
      print("En la iteracion ", iteracion, ", la mejor solución encontrada es:" , mejor_
solucion)
      print("Distancia     :" , mejor_distancia)
      return mejor_solucion

    solucion_referencia = vecina


sol = busqueda_local(problem )
```

```
En la iteracion  33 , la mejor solución encontrada es: [0, 6, 5, 14, 15, 7, 17, 31, 20, 3
3, 34, 38, 22, 29, 4, 18, 10, 8, 9, 21, 23, 41, 25, 11, 12, 26, 13, 19, 16, 37, 36, 35, 32
, 30, 39, 24, 40, 28, 27, 2, 3, 1]
Distancia     : 1798
```

SIMULATED ANNEALING

In [7]:

```
###############################################################################
# SIMULATED ANNEALING
###############################################################################

#Generador de 1 solucion vecina 2-opt 100% aleatoria (intercambiar 2 nodos)
#Mejorable eligiendo otra forma de elegir una vecina.
def genera_vecina_aleatorio(solucion):

  #Se eligen dos nodos aleatoriamente
  i,j = sorted(random.sample( range(1,len(solucion)) , 2))
```

```
  #Devuelve una nueva solución pero intercambiando los dos nodos elegidos al azar
  return solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + solucion[j+1:]


#Funcion de probabilidad para aceptar peores soluciones
def probabilidad(T,d):
  if random.random() <  math.exp( -1*d / T)  :
    return True
  else:
    return False

#Funcion de descenso de temperatura
def bajar_temperatura(T):
  return T*0.9999
```

multiarranque para mejora

In [8]:

```
def recocido_simulado(problem, TEMPERATURA ):
  #problem = datos del problema
  #T = Temperatura

  solucion_referencia = crear_solucion(Nodos)
  distancia_referencia = distancia_total(solucion_referencia, problem)

  mejor_solucion = []              #x* del seudocodigo
  mejor_distancia = 10e100         #F* del seudocodigo


  N=0
  while TEMPERATURA > .0001:
    N+=1
    #Genera una solución vecina
    vecina =genera_vecina_aleatorio(solucion_referencia)

    #Calcula su valor(distancia)
    distancia_vecina = distancia_total(vecina, problem)

    #Si es la mejor solución de todas se guarda(siempre!!!)
    if distancia_vecina < mejor_distancia:
        mejor_solucion = vecina
        mejor_distancia = distancia_vecina

    #Si la nueva vecina es mejor se cambia
    #Si es peor se cambia según una probabilidad que depende de T y delta(distancia_refer
encia - distancia_vecina)
    if distancia_vecina < distancia_referencia or probabilidad(TEMPERATURA, abs(distanci
a_referencia - distancia_vecina) ) :
      #solucion_referencia = copy.deepcopy(vecina)
      solucion_referencia = vecina
      distancia_referencia = distancia_vecina

    #Bajamos la temperatura
    TEMPERATURA = bajar_temperatura(TEMPERATURA)

  print("La mejor solución encontrada es " , end="")
  print(mejor_solucion)
  print("con una distancia total de " , end="")
  print(mejor_distancia)
  return mejor_solucion

sol  = recocido_simulado(problem, 10000000)
```

```
La mejor solución encontrada es [0, 27, 28, 8, 10, 25, 11, 12, 18, 26, 6, 1, 7, 17, 31, 32
, 30, 29, 9, 41, 23, 40, 24, 21, 39, 22, 38, 34, 33, 20, 35, 36, 37, 15, 16, 14, 19, 13, 5
, 4, 2, 3]
con una distancia total de 1451
```