AG3 - Actividad Guiada 3

Nombre: Johnny Andres Illescas Fernandez
Link:
https://colab.research.google.com/drive/1bOkmoZnBnjsFei508qZTiRHfZ_Z5WpxJ?usp=sharing
Github:
https://github.com/AndresGin/VIU_Algoritmos_de_optimizacion/tree/main/AG3

In [5]:
```
import tsplib95
import random
from math import e
import urllib.request
import networkx as nx
```

In [6]:
```
#DATOS DEL PROBLEMA
file = "swiss42.tsp" ; urllib.request.urlretrieve("http://comopt.ifi.uni-
!gzip -d swiss42.tsp.gz     #Descomprimir el fichero de datos
problem = tsplib95.load(file)

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

#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(lista de nodos)
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[
```

Algoritmo de colonia de hormigas

La función Add_Nodo selecciona al azar un nodo con probabilidad uniforme. Para ser mas eficiente debería seleccionar el próximo nodo siguiendo la probabilidad correspondiente a la ecuación:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha [\nu_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}(t)]^\alpha [\nu_{il}]^\beta}, \text{ si } j \in J_i^k$$

$$p_{ij}^k(t) = 0, \text{ si } j \notin J_i^k$$

In [7]:
```
def Add_Nodo(problem, H ,T ) :
  #Mejora:Establecer una funcion de probabilidad para
  # añadir un nuevo nodo dependiendo de los nodos mas cercanos y de las f
  Nodos = list(problem.get_nodes())
  return random.choice(   list(set(range(1,len(Nodos))) - set(H) )  )
```

```
def Incrementa_Feromona(problem, T, H ) :
  #Incrementa segun la calidad de la solución. Añadir una cantidad invers
  for i in range(len(H)-1):
    T[H[i]][H[i+1]] += 1000/distancia_total(H, problem)
  return T

def Evaporar_Feromonas(T ):
  #Evapora 0.3 el valor de la feromona, sin que baje de 1
  #Mejora:Podemos elegir diferentes funciones de evaporación dependiendo
  T = [[ max(T[i][j] - 0.3 , 1) for i in range(len(Nodos)) ] for j in ran
  return T
```

In [8]:

```
def hormigas(problem, N) :
  #problem = datos del problema
  #N = Número de agentes(hormigas)

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

  #Inicializa las aristas con una cantidad inicial de feromonas:1
  #Mejora: inicializar con valores diferentes dependiendo diferentes crit
  T = [[ 1 for _ in range(len(Nodos)) ] for _ in range(len(Nodos))]

  #Se generan los agentes(hormigas) que serán estructuras de caminos desd
  Hormiga = [[0] for _ in range(N)]

  #Recorre cada agente construyendo la solución
  for h in range(N) :
    #Para cada agente se construye un camino
    for i in range(len(Nodos)-1) :

      #Elige el siguiente nodo
      Nuevo_Nodo = Add_Nodo(problem, Hormiga[h] ,T )
      Hormiga[h].append(Nuevo_Nodo)

    #Incrementa feromonas en esa arista
    T = Incrementa_Feromona(problem, T, Hormiga[h] )
    #print("Feromonas(1)", T)

    #Evapora Feromonas
    T = Evaporar_Feromonas(T)
    #print("Feromonas(2)", T)

    #Seleccionamos el mejor agente
  mejor_solucion = []
  mejor_distancia = 10e100
  for h in range(N) :
    distancia_actual = distancia_total(Hormiga[h], problem)
    if distancia_actual < mejor_distancia:
      mejor_solucion = Hormiga[h]
      mejor_distancia =distancia_actual


  print(mejor_solucion)
  print(mejor_distancia)
```

```
hormigas(problem, 1000)
```

```
[0, 17, 36, 35, 31, 19, 4, 10, 29, 28, 23, 11, 6, 7, 34, 38, 39, 22, 5, 2
7, 20, 9, 25, 13, 15, 26, 12, 18, 3, 8, 21, 24, 33, 16, 2, 30, 32, 41, 37,
40, 14, 1]
3748
```