# Actividad Guiada 3


## Emilio Jesús Hernández Salas

- Link repositorio de GitHub: 03MIAR_Algoritmos_de_Optimizacion


## Carga de librerias

```
!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 /usr/local/lib/python3.10/dist-packages (2.27.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests) (2023.5.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests) (3.4)
Requirement already satisfied: tsplib95 in /usr/local/lib/python3.10/dist-packages (0.7.1)
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.10/dist-packages (from tsplib95) (8.1.3)
Requirement already satisfied: Deprecated~=1.2.9 in /usr/local/lib/python3.10/dist-packages (from tsplib95) (1.2.14)
Requirement already satisfied: networkx~=2.1 in /usr/local/lib/python3.10/dist-packages (from tsplib95) (2.8.8)
Requirement already satisfied: tabulate~=0.8.7 in /usr/local/lib/python3.10/dist-packages (from tsplib95) (0.8.10)
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-packages (from Deprecated~=1.2.9->tsplib95) (1.14.1
```

## Carga de los datos del problema

```
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
import copy
import time

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

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

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

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

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



#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)
```

```
15
```

## Funcionas basicas

```
#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(solucion)))]
  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)
```

## BUSQUEDA ALEATORIA

```
################################################################################
# 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, 7, 31, 17, 34, 5, 41, 29, 2, 37, 25, 10, 26, 14, 15, 1, 16, 32, 6, 30, 22, 38, 28, 3, 4, 21, 9, 39, 23, 24, 20,
Distancia     : 3645
```

## BUSQUEDA LOCAL

```
################################################################################
# BUSQUEDA LOCAL
################################################################################
def genera_vecina(solucion):
  #Generador de soluciones vecinas: 2-opt (intercambiar 2 nodos) Si hay N nodos se generan (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
  mejor_distancia = float('inf')
  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:] # swap de i con j

      #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, 16, 11, 38, 49, 10, 39, 33, 45, 15, 24, 43, 26, 3
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: 3645
Distancia Mejor Solucion Local: 3436
```

Alteración del código anterior, uso de la búsqueda aleatoria inicial para darle un mejor candidato de entrada a la búsqueda local

```
# Búsqueda aleatoria
solucion = busqueda_aleatoria(problem, 5000)

# Búsqueda local
print("Distancia Solucion Incial:" , distancia_total(solucion, problem))

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

```
Mejor solución: [0, 21, 41, 11, 3, 29, 22, 18, 25, 24, 40, 38, 26, 27, 30, 39, 9, 23, 34, 16, 6, 37, 35, 4, 32, 2, 33, 20, 1, 17, 1
Distancia     : 3779
Distancia Solucion Incial: 3779
Distancia Mejor Solucion Local: 3515
```

## Búsqueda local, mejora

```
#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)
  print("Solución inicial: ", solucion_referencia)
  print("Distancia inicial: ", mejor_distancia)

  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 operador de vencindad 2-opt)
    if distancia_vecina < mejor_distancia:
      #mejor_solucion = copy.deepcopy(vecina)   #Con copia profunda. Las copias en python 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 )
```

```
Solución inicial:  [0, 6, 23, 16, 24, 35, 5, 32, 13, 19, 12, 33, 8, 25, 21, 28, 7, 26, 27, 36, 11, 1, 40, 10, 17, 34, 15, 4, 37, 31
Distancia inicial:  4991
En la iteracion  39 , la mejor solución encontrada es: [0, 1, 7, 17, 36, 35, 31, 3, 6, 19, 13, 5, 26, 18, 12, 11, 25, 41, 40, 24, 2
Distancia     : 1726
```

## Multi-partida

```
def busqueda_local_mejorada(solucion, problem):
  mejor_solucion = []

  #Generar una solucion inicial de referencia(aleatoria)
  # print("Solucion dentro: ", solucion)
  solucion_referencia = solucion
  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 operador de vencindad 2-opt)
    if distancia_vecina < mejor_distancia:
      # print("Distancia vecina: ", distancia_vecina)
      # print("vecina: ", vecina)
      #mejor_solucion = copy.deepcopy(vecina)   #Con copia profunda. Las copias en python son por referencia
      mejor_solucion = vecina                  #Guarda la mejor solución encontrada
      mejor_distancia = distancia_vecina

    else:
      # print("Return: ", mejor_solucion)
      return mejor_solucion

    solucion_referencia = vecina

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

  mejor_solucion = []
  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

  return mejor_solucion


#Busqueda Local con multi-partida:
def busqueda_multi_partida(problem):
  # solución de referencia
  mejor_solucion = crear_solucion(Nodos)
  solucion = mejor_solucion
  mejor_distancia = distancia_total(mejor_solucion, problem)
  print("Solución inicial: ", mejor_solucion)
  print("Distancia inicial: ", mejor_distancia)

  # multi-arranque
  for k in range(50):
    if k != 0:
      # Nuevo arranque
      # Elección aleatoria de solución inicial
      solucion = crear_solucion(Nodos)
      # O pequeña búsqueda aleatoria
      #solucion = busqueda_aleatoria_modificada(problem, 500)
```

```
    solucion = busqueda_local_mejorada(solucion, problem)
    distancia = distancia_total(solucion, problem)

    if distancia < mejor_distancia:
      # poner distancias nueva y solucion
      print(f"Iteración {k}, distancia: {distancia}")
      mejor_solucion = solucion
      mejor_distancia = distancia

  # devolver solucion
  return mejor_solucion

sol = busqueda_multi_partida(problem)
print("Solución final: ", sol)
print("Con distancia: ", distancia_total(sol, problem))
```

```
Solución inicial:  [0, 21, 27, 41, 7, 23, 34, 3, 19, 25, 14, 2, 22, 26, 4, 29, 31, 35, 36, 32, 5, 38, 39, 37, 15, 40, 30, 6, 1, 20,
Distancia inicial:  4660
Iteración 0, distancia: 1764
Iteración 1, distancia: 1750
Iteración 2, distancia: 1657
Iteración 7, distancia: 1543
Solución final:  [0, 1, 6, 4, 3, 2, 27, 30, 38, 22, 39, 24, 40, 21, 9, 29, 28, 8, 23, 41, 25, 10, 26, 5, 18, 12, 11, 13, 19, 14, 16
Con distancia:  1543
```

## Búsqueda en entornos variables (BNS) multi-partida

Ejemplo 2 operadores:

- SWAP
- Insertion (insertar un valor en todos los lugares)

Si con el SWAP no mejora entonces empezar a usar el Insertion para intentar mejorar la solución.

```
def busqueda_local_insercion(solucion, problem):
  mejor_solucion = solucion.copy()
  mejor_distancia = distancia_total(mejor_solucion, problem)

  for i in range(len(solucion)-1):
    vi = solucion[i]
    for j in range(1,len(solucion)):
      temp = copy.deepcopy(solucion)
      vj = temp[j]
      temp[i] = vj
      temp[j] = vi

      distancia = distancia_total(temp, problem)
      # print("Temp: ", temp)
      # print("Distancia: ", distancia)
      if distancia < mejor_distancia:
        # poner distancias nueva y solucion
        mejor_solucion = temp
        mejor_distancia = distancia

  return mejor_solucion

def busqueda_local_SWAP(solucion, problem):
  mejor_solucion = []

  #Generar una solucion inicial de referencia(aleatoria)
  solucion_referencia = copy.deepcopy(solucion)
  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 operador de vencindad 2-opt)
    if distancia_vecina < mejor_distancia:
      mejor_solucion = vecina                   #Guarda la mejor solución encontrada
      mejor_distancia = distancia_vecina
```

```
    else:
      # print("Return: ", mejor_solucion)
      return mejor_solucion

    solucion_referencia = vecina

def busqueda_entornos_variables_multi_partida(solucion, tiempo, problem):

  mejor_solucion = solucion
  solucion = copy.deepcopy(mejor_solucion)
  mejor_distancia = distancia_total(mejor_solucion, problem)

  terminado = False

  iteracion = 0
  t = time.time()

  while not terminado:
    if iteracion%5 == 0:
      print(f"Iteración {iteracion}, time: {time.time()-t}, mejor distancia: {mejor_distancia}")
    # saltar primera iteracion
    if iteracion != 0:
      # print("Entra")
      solucion = crear_solucion(Nodos)

    # búsqueda local mediante inserción
    solucion = busqueda_local_insercion(solucion, problem)
    # print("Inser: ",solucion)

    # búsqueda local mediante SWAP
    solucion = busqueda_local_SWAP(solucion, problem)
    # print("Vecin: ",solucion)

    # verificar distancia
    distancia = distancia_total(solucion, problem)
    # print("Temp: ", temp)
    # print("Distancia: ", distancia)
    if distancia < mejor_distancia:
      # poner distancias nueva y solucion
      mejor_solucion = copy.deepcopy(solucion)
      mejor_distancia = distancia

    # condición de salida del bucle
    if time.time()-t > tiempo:
      terminado = True

    iteracion += 1

  return mejor_solucion

sol_ini = crear_solucion(Nodos)

print("Solución inicial: ", sol_ini)
# sol = busqueda_local_insercion(sol_ini, problem)
sol = busqueda_entornos_variables_multi_partida(sol_ini, 60*10, problem)
print("Solución final: ", sol)
print("Con distancia: ", distancia_total(sol, problem))
```

```
Solución inicial:  [0, 5, 14, 27, 10, 6, 34, 39, 1, 40, 26, 30, 12, 36, 4, 31, 23, 35, 37, 38, 32, 11, 20, 18, 22, 17, 28, 2, 19, 1
Iteración 0, time: 1.430511474609375e-06, mejor distancia: 5512
Iteración 5, time: 15.922852277755737, mejor distancia: 1676
Iteración 10, time: 34.045132875442505, mejor distancia: 1623
Iteración 15, time: 53.140644550323486, mejor distancia: 1623
Iteración 20, time: 72.54646110534668, mejor distancia: 1579
Iteración 25, time: 90.28682088851929, mejor distancia: 1579
Iteración 30, time: 106.35286021232605, mejor distancia: 1579
Iteración 35, time: 123.59734869003296, mejor distancia: 1502
Iteración 40, time: 142.18990874290466, mejor distancia: 1502
Iteración 45, time: 158.48347640037537, mejor distancia: 1502
Iteración 50, time: 175.84734678268433, mejor distancia: 1502
Iteración 55, time: 191.79933381080627, mejor distancia: 1502
Iteración 60, time: 211.195415019989, mejor distancia: 1502
Iteración 65, time: 226.72948908805847, mejor distancia: 1502
Iteración 70, time: 243.47152733802795, mejor distancia: 1502
Iteración 75, time: 262.5954713821411, mejor distancia: 1476
Iteración 80, time: 281.0244278907776, mejor distancia: 1476
Iteración 85, time: 297.3568696975708, mejor distancia: 1476
Iteración 90, time: 313.48455119132996, mejor distancia: 1476
Iteración 95, time: 331.832665681839, mejor distancia: 1476
Iteración 100, time: 350.5243856906891, mejor distancia: 1476
Iteración 105, time: 367.67851519584656, mejor distancia: 1476
Iteración 110, time: 385.4772665500641, mejor distancia: 1476
Iteración 115, time: 404.0156726837158, mejor distancia: 1476
Iteración 120, time: 421.50008893013, mejor distancia: 1476
```

```
Iteración 125, time: 439.5622444152832, mejor distancia: 1476
Iteración 130, time: 457.85222482681274, mejor distancia: 1476
Iteración 135, time: 473.71797013282776, mejor distancia: 1476
Iteración 140, time: 491.8571615219116, mejor distancia: 1476
Iteración 145, time: 511.06534242630005, mejor distancia: 1476
Iteración 150, time: 527.3330707550049, mejor distancia: 1476
Iteración 155, time: 543.4654603004456, mejor distancia: 1476
Iteración 160, time: 561.7864594459534, mejor distancia: 1476
Iteración 165, time: 578.5837728977203, mejor distancia: 1476
Iteración 170, time: 596.1195013523102, mejor distancia: 1476
Solución final:  [0, 1, 32, 31, 35, 36, 17, 7, 37, 15, 16, 14, 19, 13, 5, 26, 6, 4, 27, 28, 29, 30, 34, 20, 33, 38, 22, 24, 40, 21,
Con distancia:  1476
```