# Actividad Guiada 3

## Algoritmos de Optimización, Máster en Inteligencia Artificial, VIU

**José Diogo Rivero Freitas**

Notebook en Google Colab: https://colab.research.google.com/drive/1PEAcZWWKH0q50U6Q-TwRARDix1Ylbu7p?usp=sharing

GitHub personal: https://github.com/JDiogoRiveroFreitas

Carpeta de la asignatura en GitHub: https://github.com/JDiogoRiveroFreitas/AlgoritmosOptmizacion-03MIAR.git

## Carga de librerias

```
!pip install fastapi
!pip install kaleido
!pip install python-multipart
!pip install uvicorn
!pip install tabulate
```

```
Requirement already satisfied: fastapi in /usr/local/lib/python3.10/dist-packages (0
Requirement already satisfied: pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1,!=2.1.0,<3.0.0,>
Requirement already satisfied: starlette<0.36.0,>=0.35.0 in /usr/local/lib/python3.10
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/
Requirement already satisfied: anyio<5,>=3.4.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: kaleido in /usr/local/lib/python3.10/dist-packages (0
Requirement already satisfied: python-multipart in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: uvicorn in /usr/local/lib/python3.10/dist-packages (0
Requirement already satisfied: click>=7.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: h11>=0.8 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: typing-extensions>=4.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-packages (0
```

```
!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
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: tsplib95 in /usr/local/lib/python3.10/dist-packages (0
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: Deprecated~=1.2.9 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: networkx~=2.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: tabulate~=0.8.7 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-packa
```

# 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


#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/sw
!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/so

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

```
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())
```

```
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
```

```
#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(solucio
  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
```

```
    (4356,
     [0,
      34,
      14,
      35,
      2,
      24,
      30,
      8,
      19,
      29,
      38,
      27,
      10,
      31,
      37,
      36,
      18,
      23,
      5,
      32,
      3,
      26,
      28,
      41,
      40,
      11,
      4,
      12,
      39,
      22,
      25,
      33,
      15,
      6,
      21,
      9,
```

```
        17,
        16,
        13,
        7,
        20,
        1])
```

## 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
    solucion = crear_solucion(Nodos)                 #Genera una solucion aleatoria
    distancia = distancia_total(solucion, problem)  #Calcula el valor objetivo(distancia

    if distancia < mejor_distancia:                  #Compara con la mejor obtenida hasta
      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, 22, 7, 38, 30, 3, 6, 41, 29, 21, 40, 37, 16, 15, 14, 1, 17, 19, 1
Distancia     : 3634
```

## BUSQUEDA LOCAL

```
##############################################################################
# BUSQUEDA LOCAL
##############################################################################
def genera_vecina(solucion):
  #Generador de soluciones vecinas: 2-opt (intercambiar 2 nodos) Si hay N nodos se gener
  #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
    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] + [
      vecina = solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + solucion

      #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,
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: 3634
Distancia Mejor Solucion Local: 3260
```

```
#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 ope
    if distancia_vecina < mejor_distancia:
      #mejor_solucion = copy.deepcopy(vecina)   #Con copia profunda. Las copias en pytho
      mejor_solucion = vecina                   #Guarda la mejor solución encontrada
      mejor_distancia = distancia_vecina

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

    solucion_referencia = vecina


sol = busqueda_local(problem )
```

```
En la iteracion  32 , la mejor solución encontrada es: [0, 17, 35, 36, 37, 15, 19, 1
Distancia     : 1776
```

## SIMULATED ANNEALING

```
##############################################################################
# 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):
```