AG3- Actividad Guiada 3

Nombre: Javier Canales Navarrete

https://github.com/Javicana/03MAIR-Algoritmos-de-Optimizacon-2021

https://colab.research.google.com/drive/1FGhFfDTZHejZDYnWchmdR5NueUWcsJmy?usp=sharing

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/python3.7/dist-packages (2.23.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from reque
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests) (2021.10.8)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests) (3.0.4)
Collecting tsplib95
  Downloading tsplib95-0.7.1-py2.py3-none-any.whl (25 kB)
Collecting Deprecated~=1.2.9
  Downloading Deprecated-1.2.13-py2.py3-none-any.whl (9.6 kB)
Requirement already satisfied: tabulate~=0.8.7 in /usr/local/lib/python3.7/dist-packages (from tsplib95) (0.8.9)
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.7/dist-packages (from tsplib95) (7.1.2)
Requirement already satisfied: networkx~=2.1 in /usr/local/lib/python3.7/dist-packages (from tsplib95) (2.6.3)
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.7/dist-packages (from Deprecated~=1.2.9->tsplib95) (
Installing collected packages: Deprecated, tsplib95
Successfully installed Deprecated-1.2.13 tsplib95-0.7.1
```

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

#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://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/swiss42.tsp", file)

#Coordendas 51-city problem (Christofides/Eilon)
#file = "eil51.tsp" ; urllib.request.urlretrieve("http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/eil51.tsp", file)

#Coordenadas - 48 capitals of the US (Padberg/Rinaldi)
#file = "att48.tsp" ; urllib.request.urlretrieve("http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/att48.tsp", file)
```

```
('swiss42.tsp', <http.client.HTTPMessage at 0x7f48fff87ed0>)
```

```
#Modulos extras, no esenciales
import numpy as np
import matplotlib.pyplot as plt
import imageio                    #Para construir las imagenes con gif
from google.colab import files    #Para descargar ficheros generados con google colab

from tempfile import mkstemp      #Para genera carpetas y ficheros temporales

import random                     #Para generar valores aleatorios


#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, 2)

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

#dir(problem)
```

```
30
```

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)



solucion = crear_solucion(Nodos)
print(solucion)
distancia_total(solucion, problem)
```

```
[0, 21, 26, 36, 4, 10, 38, 2, 37, 18, 40, 11, 35, 12, 15, 33, 5, 28, 13, 39, 24, 9, 29, 7, 22, 20, 31, 8, 27, 23, 6, 14, 1,
5095
```

Busqueda Aleatoria

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

def busqueda_aleatoria(problem, N):
  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, 5000)
```

```
Mejor solución: [0, 33, 13, 41, 10, 16, 15, 7, 36, 35, 29, 40, 3, 2, 38, 11, 23, 9, 25, 1, 6, 19, 18, 32, 17, 31, 37, 4, 5,
Distancia     : 3802
```

Busqueda Local 2-opt

```
################################################################################
# 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
  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, 16, 11, 38, 49, 10, 39, 33, 45, 15, 24, 43
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: 3802
Distancia Mejor Solucion Local: 3560
```

```
#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 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 )
```

```
En la iteracion  47 , la mejor solución encontrada es: [0, 26, 12, 11, 13, 19, 37, 36, 35, 20, 33, 34, 27, 4, 18, 10, 25, 41
Distancia     : 1777
```

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):

  #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:]

print(solucion)
genera_vecina_aleatorio(solucion)
```

```
[0, 33, 13, 41, 10, 16, 15, 7, 36, 35, 29, 40, 3, 2, 38, 11, 23, 9, 25, 1, 6, 19, 18, 32, 17, 31, 37, 4, 5, 30, 20, 24, 39,
[0,
 33,
 13,
 41,
 31,
 16,
 15,
 7,
 36,
 35,
 29,
 40,
 3,
 2,
 38,
 11,
 23,
 9,
 25,
 1,
 6,
 19,
 18,
 32,
 17,
 10,
 37,
 4,
 5,
```

```
        30,
        20,
        24,
        39,
        28,
        14,
        22,
        21,
        8,
        34,
        27,
        12,
        26]
```

```
#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*0.99




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 = []
  mejor_distancia = 10e100


  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_referencia - distancia_vecina)
    if distancia_vecina < distancia_referencia or probabilidad(TEMPERATURA, abs(distancia_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, 6, 2, 27, 25, 41, 10, 34, 32, 28, 8, 23, 40, 24, 21, 39, 9, 29, 22, 38, 30, 33, 20, 31,
```

```
con una distancia total de 1973
```