


AG3- Actividad Guiada 3

Nombre: Mikel Alberdi Blasco

<https://github.com/mikelalberdi1/Algoritmos-optimizacion>https://colab.research.google.com/drive/1IN_JZKgeIW0Sd7j2W-gUTqIJl_OveKwT?usp=sharing

▼ Carga de librerías

```
!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 (2.25.1)
 Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (3.0.2)
 Requirement already satisfied: urllib3!=1.25.0,!<1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (1.25.1)
 Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (2.10)
 Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (2021.10.8)
 Collecting tsplib95
 Downloading tsplib95-0.7.1-py2.py3-none-any.whl (25 kB)
 Requirement already satisfied: tabulate<0.8.7 in /usr/local/lib/python3.7/dist-packages (0.8.7)
 Collecting Deprecated<1.2.9
 Downloading Deprecated-1.2.13-py2.py3-none-any.whl (9.6 kB)
 Requirement already satisfied: networkx<2.1 in /usr/local/lib/python3.7/dist-packages (2.0)
 Requirement already satisfied: Click<6.0 in /usr/local/lib/python3.7/dist-packages (5.1)
 Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.7/dist-packages (1.12.1)
 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.ifl.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")

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

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

```
('swiss42.tsp', <http.client.HTTPMessage at 0x7f57b16b4f10>)
```

```
#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, 4)
```

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

```
#dir(problem)
```

32

▼ 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, 11, 28, 9, 17, 21, 30, 38, 23, 14, 20, 3, 10, 26, 36, 18, 39, 34, 5, 33, 13, 6, 2, 4980]

```



▼ Búsqueda Aleatoria

```

#####
# BUSQUEDA ALEATORIA
#####

def busqueda_aleatoria(problem, N):
    Nodos = list(problem.get_nodes())

    mejor_solucion = []
    #mejor_distancia = 10e100
    mejor_distancia = float('inf')

    #Inicializamos con un valor alto
    #Inicializamos con un valor alto

    for i in range(N):
        solucion = crear_solucion(Nodos)
        distancia = distancia_total(solucion, problem)

        #Criterio de parada: repetir N veces p
        #Genera una solucion aleatoria
        #Calcula el valor objetivo(distancia t

        if distancia < mejor_distancia:
            mejor_solucion = solucion
            mejor_distancia = distancia

        #Compara con la mejor obtenida hasta a

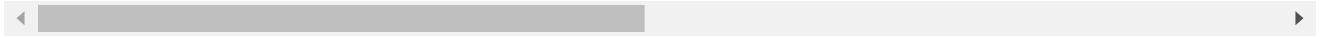
    print("Mejor solución:" , mejor_solucion)

```

```
print("Distancia      :", mejor_distancia)
return mejor_solucion
```

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

```
Mejor solución: [0, 22, 32, 34, 40, 3, 27, 1, 31, 11, 29, 4, 25, 30, 28, 9, 8, 10, 15]
Distancia      : 3738
```



▼ 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
    #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 p
        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]
            vecina = solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + solucion[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]
print("Distancia Solucion Inicial:" , distancia_total(solucion, problem))
```

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

```
Distancia Solucion Inicial: 3738
Distancia Mejor Solucion Local: 3434
```

```

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

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

    solucion_referencia = vecina

sol = busqueda_local(problem )

```

```

En la iteracion  34 , la mejor solución encontrada es: [0, 36, 35, 20, 33, 29, 8, 10,
Distancia      : 1926

```



▼ 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:]

genera_vecina_aleatorio(solucion)
```

```
[0,
 22,
 32,
 34,
 40,
 3,
 27,
 1,
 31,
 11,
 29,
 4,
 25,
 30,
 28,
 9,
 8,
 10,
 19,
 7,
 38,
 36,
 17,
 15,
 16,
 20,
 21,
 35,
 12,
 23,
 24,
 18,
 2,
 5,
 6,
 37,
 14,
 13,
 33,
 39,
 41,
 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_refere
        if distancia_vecina < distancia_referencia or probabilidad(TEMPERATURA, abs(distancia_
            #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, 34, 20, 22, 38, 33, 35, 36, 31, 17, 37, 14, 19, 1] con una distancia total de 1987

