Algoritmos - Actividad Guiada 3

Nombre: Juan Carlos Marin Mejia

https://github.com/jcmm518/03MAIR-Algoritmos-de-optimizacion.git

https://colab.research.google.com/drive/1IX_dFMK36Vd0XEGwe3yySqv95GN_eLmf?usp=sharing

Problema del Viajero (TSP)

```
!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/python3.6/dist-packages (2.23
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packa
Collecting tsplib95
  Downloading https://files.pythonhosted.org/packages/a0/2b/b1932d3674758ec5f49afa72d451
Requirement already satisfied: tabulate~=0.8.7 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: networkx~=2.1 in /usr/local/lib/python3.6/dist-packages (
Collecting Deprecated~=1.2.9
  Downloading https://files.pythonhosted.org/packages/76/a1/05d7f62f956d77b23a640efc650f
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.6/dist-packages
Installing collected packages: Deprecated, tsplib95
Successfully installed Deprecated-1.2.10 tsplib95-0.7.1
```

```
import tsplib95       #Modulo para las instancias del problema del TSP
import random         #Modulo para generar números aleatorios
from math import e    #constante e
import copy           #Para copia profunda de estructuras de datos(en python la asignación es
```

```
import urllib.request #Hacer llamadas http a paginas de la red

#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" ;
```

```
file = "swiss42.tsp" ;
urllib.request.urlretrieve("http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/swiss42.tsp", f

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

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

```
('swiss42.tsp', <http.client.HTTPMessage at 0x7fd3a46ae198>)
```

```
#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 tempfile



#Carga de datos y generación de objeto problem

problem = tsplib95.load(file)

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

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

print(Nodos)
print(Aristas)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 2
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (0, 10)
```

```
#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
```

Metaheurísticas de búsquedas: Busqueda Aleatoria

```
#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)
```

```
def busqueda_aleatoria(problem, N):

  Nodos = list(problem.get_nodes())

  mejor_solucion = []
  mejor_distancia = 10e100                          #Inicializamos con un valor alto

  for i in range(N):                                #Criterio de parada: repetir N veces pero
    solucion = crear_solucion(Nodos)                #Genera una solucion aleatoria
    distancia = distancia_total(solucion, problem)  #Calcula el valor objetivo(distancia tota

    if distancia < mejor_distancia:                 #Compara con la mejor obtenida hasta ahor
      mejor_solucion = solucion
      mejor_distancia = distancia


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

```
solucion = busqueda_aleatoria(problem, 1000 )
```

```
Mejor solución: [0, 27, 2, 8, 28, 6, 10, 30, 9, 24, 25, 41, 29, 21, 4, 23, 12, 33, 20, 1
Distancia     : 3947
```

```
def genera_vecina(solucion):
  #Generador de soluciones vecinas: 2-opt (intercambiar 2 nodos) Si hay N nodos se generan (N
  #print(solucion)
  mejor_solucion = []
  mejor_distancia = 10e100
  for i in range(1,len(solucion)-1):          #Recorremos todos los nodos en bucle doble para
    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+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 = [0, 30, 5, 22, 23, 4, 9, 40, 29, 28, 11, 3, 38, 21, 32, 37, 1, 41, 7, 14, 26, 24,
print("Distancia Solucion Incial:" , distancia_total(solucion, problem))


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

```
Distancia Solucion Incial: 3947
Distancia Solucion Local: 3492
[0, 27, 2, 8, 28, 6, 10, 30, 9, 24, 25, 41, 29, 21, 4, 23, 12, 33, 20, 17, 34, 40, 22, 1
```

Busqueda Local

```
#Busqueda Local:
#  - Sobre el operador de vecindad 2-opt(funcion genera_vecina)
#  - Sin criterio de parada, se para cuando no es posible mejorar
```

```
#    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 terminas. Hemos llegado a un minimo local(según nuestro operador
    if distancia_vecina < mejor_distancia:
      mejor_solucion = copy.deepcopy(vecina)   #Con copia profunda. Las copias en python son
      mejor_distancia = distancia_vecina
    else:
      print("En la iteracion ", iteracion, ", la mejor solución encontrada es:" , mejor_soluc
      print("Distancia     :" , mejor_distancia)
      return mejor_solucion

    solucion_referencia = vecina


sol = busqueda_local(problem )
```

```
En la iteracion  33 , la mejor solución encontrada es: [0, 19, 5, 26, 40, 24, 21, 39, 22
Distancia     : 1938
```

```
#Generador de 1 solucion vecina 2-opt aleatoria (intercambiar 2 nodos)
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,
 27,
```

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

```
#Funcion de probabilidad para aceptar peores soluciones
import math
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
```

Metaheurísticas de búsquedas: Recocido Simulado

```
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_referenci
    if distancia_vecina < distancia_referencia or probabilidad(TEMPERATURA, abs(distancia_ref
      solucion_referencia = copy.deepcopy(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, 32, 34, 31, 17, 7, 3, 27, 2, 8, 10, 11, 12, 25, 9, 2
con una distancia total de 1913
```