AG3 - Actividad Guiada 3

Nombre: Alberto Rodriguez Arizaga

Actividad guiada de Algoritmos Optimización (AG3)

https://colab.research.google.com/drive/14jKTxli5jPJ2lgfniLAXtmCsGcu3eOmn?usp=sharing

https://github.com/brtIn05/03MIAR---Algoritmos-de-Optimizacion

Inicialización del problema

```
#Modulo de llamadas http para descargar ficheros
!pip install requests

#Libreria del problema TSP: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html
!pip install tsplib95
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (2.
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting tsplib95
  Downloading tsplib95-0.7.1-py2.py3-none-any.whl (25 kB)
Collecting networkx~=2.1
  Downloading networkx-2.8.8-py3-none-any.whl (2.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 10.7 MB/s eta 0:00:00
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.8/dist-packages (
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/python3.8/dist-packa
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.8/dist-packag
Installing collected packages: networkx, Deprecated, tsplib95
  Attempting uninstall: networkx
    Found existing installation: networkx 3.0
    Uninstalling networkx-3.0:
      Successfully uninstalled networkx-3.0
Successfully installed Deprecated-1.2.13 networkx-2.8.8 tsplib95-0.7.1
```

```
import tsplib95
import random
import math
import urllib.request
```

✓ 0 s completado a las 20:18

```
file = "swiss42.tsp" ; urllib.request.urlretrieve("http://comopt.ifi.uni-heidelberg.de/soft
!gzip -d swiss42.tsp.gz     #Descomprimir el fichero de datos
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)

#dir(problem)
```

```
30
```

Creación de Funciones Básicas

```
# Generamos las funciones básicas que serán utilizadas posteriromente

# Generamos una solución aleatoria comenzando por 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

# Distancia entre dos nodos
def distancia(a,b, problem):
  return problem.get_weight(a,b)

# Distancia total de una solución completa
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

```
# Busca aleatoriamente una solución

def busqueda_aleatoria(problem, N):
  #Número de iteraciones N
  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 pe
    solucion = crear_solucion(Nodos)                #Genera una solucion aleatoria
    distancia = distancia_total(solucion, problem)  #Calcula el valor objetivo(distancia to

    if distancia < mejor_distancia:                 #Compara con la mejor obtenida hasta ah
      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, 35, 16, 20, 36, 17, 3, 10, 34, 30, 29, 41, 22, 12, 13, 7, 37, 11,
Distancia     : 3620
```

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 = float('inf')
  for i in range(1,len(solucion)-1):          #Recorremos todos los nodos en bucle doble pa
    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)
```

```
      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

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: 3620
     Distancia Mejor Solucion Local: 3391
```

```
#Busqueda Local:
#  - Sobre el operador de vecindad 2-opt(funcion genera_vecina)
#  - Sin criterio de parada, se para cuando no es posible mejorar.
#  - Implementación completa usando la función anterior
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 operac
    if distancia_vecina < mejor_distancia:
      #mejor_solucion = copy.deepcopy(vecina)   #Con copia profunda. Las copias en python s
      mejor_solucion = vecina                   #Guarda la mejor solución encontrada
      mejor_distancia = distancia_vecina

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

    solucion_referencia = vecina
```

```
    solucion_referencia = vecina


sol = busqueda_local(problem )
```

```
     En la iteracion  34 , la mejor solución encontrada es: [0, 17, 36, 35, 31, 32, 27, 2,
     Distancia     : 1709
```

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:]


#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 ):

  solucion_referencia = crear_solucion(Nodos)
  distancia_referencia = distancia_total(solucion_referencia, problem)

  mejor_solucion = []             #x* del seudocodigo
  mejor_distancia = float('inf')        #F* del seudocodigo


  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_referer
    if distancia_vecina < distancia_referencia or probabilidad(TEMPERATURA, abs(distancia_r
      #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, 27, 28, 2, 29, 32, 38, 30, 39, 22, 21, 24, 40, 23
con una distancia total de 2048
```

Productos de pago de Colab - Cancelar contratos