Nombre: Salvador Gimeno

Actividad Guiada 3 - AG3

Github: https://github.com/salvagimeno-ai/03MAIR-Algoritmos-de-optimizacion/tree/master/

Google Colab: https://colab.research.google.com/drive/1SFiu14gez7rDy_YKk2ML4hiHbgBWj4

```
!pip install request
```

```
Collecting request
  Downloading https://files.pythonhosted.org/packages/f1/27/7cbde262d854aedf217061a97
Collecting get (from request)
  Downloading https://files.pythonhosted.org/packages/3f/ef/bb46f77f7220ac1b7edba0c76
Collecting post (from request)
  Downloading https://files.pythonhosted.org/packages/0f/05/bd79da5849ea6a92485ed7029
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (
Collecting query_string (from get->request)
  Downloading https://files.pythonhosted.org/packages/12/3c/412a45daf5bea9b1d06d7de41
Collecting public (from query_string->get->request)
  Downloading https://files.pythonhosted.org/packages/54/4d/b40004cc6c07665e48af22cfe
Building wheels for collected packages: request, get, post, query-string, public
  Building wheel for request (setup.py) ... done
  Created wheel for request: filename=request-2019.4.13-cp36-none-any.whl size=1676 s
  Stored in directory: /root/.cache/pip/wheels/30/84/5f/484cfba678967ef58c16fce689092
  Building wheel for get (setup.py) ... done
  Created wheel for get: filename=get-2019.4.13-cp36-none-any.whl size=1692 sha256=4d
  Stored in directory: /root/.cache/pip/wheels/c1/e3/c1/d02c8c58538853e4c9b78cadb74f6
  Building wheel for post (setup.py) ... done
  Created wheel for post: filename=post-2019.4.13-cp36-none-any.whl size=1661 sha256=
  Stored in directory: /root/.cache/pip/wheels/c3/c3/24/b5c132b537ab380c02d69e6bd4dec
  Building wheel for query-string (setup.py) ... done
  Created wheel for query-string: filename=query_string-2019.4.13-cp36-none-any.whl s
  Stored in directory: /root/.cache/pip/wheels/d6/a4/78/01b20a9dc224dcc009fab669f7f27
  Building wheel for public (setup.py) ... done
  Created wheel for public: filename=public-2019.4.13-cp36-none-any.whl size=2536 sha
  Stored in directory: /root/.cache/pip/wheels/23/7c/6e/f5b4e09d6596c8b8802b347e48f14
Successfully built request get post query-string public
Installing collected packages: public, query-string, get, post, request
Successfully installed get-2019.4.13 post-2019.4.13 public-2019.4.13 query-string-201
```

```
!pip install tsplib95
```

```
Collecting tsplib95
  Downloading https://files.pythonhosted.org/packages/90/9f/5fbf6118d00719cc4688b175a
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.6/dist-packages (
Collecting networkx==2.1 (from tsplib95)
  Downloading https://files.pythonhosted.org/packages/11/42/f951cc6838a4dff6ce57211c4
     |████████████████████████████████| 1.6MB 4.1MB/s
Requirement already satisfied: decorator>=4.1.0 in /usr/local/lib/python3.6/dist-pack
Building wheels for collected packages: networkx
  Building wheel for networkx (setup.py) ... done
  Created wheel for networkx: filename=networkx-2.1-py2.py3-none-any.whl size=1447766
  Stored in directory: /root/.cache/pip/wheels/44/c0/34/6f98693a554301bdb405f8d65d95b
Successfully built networkx
ERROR: albumentations 0.1.12 has requirement imgaug<0.2.7,>=0.2.5, but you'll have im
Installing collected packages: networkx, tsplib95
  Found existing installation: networkx 2.3
    Uninstalling networkx-2.3:
      Successfully uninstalled networkx-2.3
Successfully installed networkx-2.1 tsplib95-0.3.3
```

```
import urllib.request
file='swiss42.tsp'
urllib.request.urlretrieve('http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/swiss42.tsp'
```

```
('swiss42.tsp', <http.client.HTTPMessage at 0x7f7cff55a908>)
```

```
import random
from math import e
import tsplib95


#definimos la variable problema
problem = tsplib95.load_problem(file)

Nodos = list(problem.get_nodes())

#Obtener nodos
list(problem.get_nodes())  # es un objeto iterable - ls podremos aplicar la funcion list
#Obtener aristas
list(problem.get_edges())

problem.wfunc(0,2)   # nos devuelve la distancia entre dos puntos
```

```
30
```

```
def distancia(a,b,problem):
  return problem.wfunc(a,b)
distancia (0,1,problem)
```

```
15
```

```
def crear_solucion(Nodos):
  solucion = [0]
```

```
    for i in range(len(Nodos)-1):
      solucion = solucion + [random.choice(list(set(Nodos) - set({0}) - set(solucion)))]
    return solucion

  def distancia_total(solucion, problem):
    distancia_ret = 0
    for i in range(len(solucion)-1):
      distancia_ret += distancia(solucion[i], solucion[i+1], problem )
    return distancia_ret + distancia(solucion[len(solucion)-1], solucion[0], problem )

  solucion_prueba = crear_solucion(Nodos)

  print(distancia_total(solucion_prueba, problem), solucion_prueba )
```

```
  4714 [0, 32, 18, 2, 22, 3, 38, 21, 35, 7, 37, 19, 15, 4, 26, 17, 41, 16, 8, 40, 23, 2
```

```
  #Busqueda Aleatoria
  def busqueda_aleatoria(problem,N):  #N = numero de iteraciones
    Nodos = list(problem.get_nodes())

    mejor_solucion = []
    mejor_distancia = 10e10   #colocamos un valor muy alto

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

      if distancia_solucion < mejor_distancia:
        mejor_solucion = solucion
        mejor_distancia = distancia_solucion

    print(mejor_distancia, mejor_solucion)
    return mejor_solucion

  busqueda_aleatoria(problem, 5000)


  #Funcion vecinos
  def generar_vecina(solucion, problem):
    mejor_solucion = []
    mejor_distancia = 10e10
    for i in range(1,len(solucion)-1):  #recorremos desde el 2o nodo hasta el penultimo
      for j in range(i+1, len(solucion)): #recorremos desde el siguiente nodo hasta el final
        vecina = solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + solucion[j
        distancia_vecina = distancia_total(vecina, problem)
        if distancia_vecina < mejor_distancia:
          mejor_solucion = solucion
          mejor_distancia = distancia_vecina
    return mejor_solucion

  #solucion_prueba = crear_solucion(Nodos)
  #generar_vecina(solucion_prueba, problem)
```

```
def busqueda_local(problem,N):
  mejor_solucion = []
  mejor_distancia = 10e10

  Nodos = list(problem.get_nodes())

  solucion_referencia = crear_solucion(Nodos)

  for i in range(N):
    vecina = generar_vecina(solucion_referencia, problem)
    distancia_vecina = distancia_total(vecina, problem)
    if distancia_vecina < mejor_distancia:
        mejor_solucion = vecina
        mejor_distancia = distancia_vecina

    solucion_referencia = mejor_solucion

  print(distancia_total(mejor_solucion, problem), mejor_solucion)

  return mejor_solucion

busqueda_local(problem,100)


#METAHEURISTICAS: Recocido Simulado
def probabilidad(T,d):   # T = temperatura, d = distancia
  r=random.random()    #generamos un numero aleatorio

  if r >= (e**(-1*d)/((T * .5*10**(-5)))):
    return True
  else:
    return False

def bajar_temperatura(T):
  return T*.9
  #return T-1


def recocido_simulado(problema, TEMPERATURA):

  solucion_referencia = crear_solucion(Nodos)

  distancia_referencia = distancia_total(solucion_referencia, problem)

  mejor_solucion = []
  mejor_distancia = 10e10

  while TEMPERATURA > 1:
    vecina = generar_vecina(solucion_referencia, problem)
    distancia_vecina = distancia_total(vecina, problem)

    if distancia_vecina < mejor_distancia:
      mejor_solucion = vecina
      mejor_distancia = distancia_vecina
```

```
    if distancia_vecina < distancia_referencia or probabilidad(TEMPERATURA, abs(distancia_
      solucion_referencia = vecina
      distancia_referencia = distancia_vecina

    TEMPERATURA = bajar_temperatura(TEMPERATURA)

    print(mejor_distancia, mejor_solucion)

    return mejor_solucion


recocido_simulado(problem,1000)
```

```
4514 [0, 12, 30, 25, 26, 37, 35, 27, 20, 7, 10, 28, 31, 1, 33, 24, 36, 13, 29, 22, 39
[0,
 12,
 30,
 25,
 26,
 37,
 35,
 27,
 20,
 7,
 10,
 28,
 31,
 1,
 33,
 24,
 36,
 13,
 29,
 22,
 39,
 32,
 2,
 14,
 41,
 11,
 9,
 21,
 5,
 17,
 19,
 16,
 3,
 6,
 23,
 4,
 40,
 34,
 38,
 8,
 18,
 15]
```