AG - Actividad guiada 3

Víctor Cebrián

URL Git: https://github.com/vcebrian/03MAIR---Algoritmos-de-optimizacion/tree/master/AG3

```
import urllib.request

file = "swiss42.tsp"

urllib.request.urlretrieve("http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/swi

#42 ciudades de Suiza en formato matríz
```

```
('swiss42.tsp', <http.client.HTTPMessage at 0x7f1e6dc06898>)
```

```
!pip install tsplib95
```

```
Collecting tsplib95
  Downloading https://files.pythonhosted.org/packages/d1/4f/6a1cb104ce9b400
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.6/dist-
Collecting networkx==2.1 (from tsplib95)
  Downloading https://files.pythonhosted.org/packages/11/42/f951cc6838a4dff
    100% |████████████████████████████████| 1.6MB 11.7MB/s
Requirement already satisfied: decorator>=4.1.0 in /usr/local/lib/python3.6
Building wheels for collected packages: networkx
  Building wheel for networkx (setup.py) ... done
  Stored in directory: /root/.cache/pip/wheels/44/c0/34/6f98693a554301bdb40
Successfully built networkx
imgaug 0.2.8 has requirement numpy>=1.15.0, but you'll have numpy 1.14.6 wh
albumentations 0.1.12 has requirement imgaug<0.2.7,>=0.2.5, but you'll have
Installing collected packages: networkx, tsplib95
  Found existing installation: networkx 2.2
    Uninstalling networkx-2.2:
      Successfully uninstalled networkx-2.2
Successfully installed networkx-2.1 tsplib95-0.3.2
```

```
import tsplib95
import random
from math import e

problem = tsplib95.load_problem(file)

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

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

```
#Devuelve el factorial de un numero
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

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

```
#Devuelve la distancia entre dos nodos
def distancia(a,b, problem):
  return problem.wfunc(a,b)


solucion = crear_solucion(Nodos)


#Devuelve la distancia total de una trayectoria
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], prob


def busqueda_aleatoria(problem, N):
  Nodos = list(problem.get_nodes())
  mejor_solucion = []
  mejor_distancia = 10e100

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

    if distancia < mejor_distancia:
      mejor_solucion = solucion
      mejor_distancia = distancia

  print("Mejor solucion:" , mejor_solucion)
  print("Distancia: ", mejor_distancia)

  return mejor_solucion


sol = busqueda_aleatoria(problem, 10000)
```

```
Mejor solucion: [0, 6, 37, 34, 7, 17, 31, 20, 40, 8, 4, 24, 39, 38, 10, 41,
Distancia:  3426
```

```
def genera_vecina(solucion):
  #Generador de soluciones vecinas: 2-opt (intercambiar 2 nodos) Si hay N nodos se
  #print(solucion)
  mejor_solucion = []
  mejor_distancia = 10e100
  for i in range(1,len(solucion)-1):
    for j in range(i+1, len(solucion)):
      vecina = solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + so
      distancia_vecina = distancia_total(vecina, problem)
      if distancia_vecina <= mejor_distancia:
        mejor_distancia = distancia_vecina
        mejor_solucion = vecina
  return mejor_solucion

solucion = crear_solucion(Nodos)
print (solucion)

nueva_solucion = genera_vecina(solucion)
print (nueva_solucion)
```

```
9, 14, 6, 30, 13, 10, 29, 34, 18, 36, 27, 1, 32, 7, 21, 28, 20, 25, 24, 41,
37, 14, 6, 30, 13, 10, 29, 34, 18, 36, 27, 1, 32, 7, 21, 28, 20, 25, 24, 41
```

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

```
  mejor_solucion = []
  mejor_distancia = 10e100

  solucion_referencia = crear_solucion(Nodos)

  for i in range(N):
    vecina = genera_vecina(solucion_referencia)
    distancia_vecina = distancia_total(vecina, problem)

    if distancia_vecina < mejor_distancia:
      mejor_solucion = vecina
      mejor_distancia = distancia_vecina

    solucion_referencia = vecina

  print("Mejor solucion:" , mejor_solucion)
  print("Distancia: ", mejor_distancia)

  return mejor_solucion

sol = busqueda_local(problem, 50)
```

```
Mejor solucion: [0, 27, 2, 12, 11, 25, 41, 23, 40, 24, 21, 39, 22, 38, 30,
Distancia:  1536
```

```
def genera_vecina_aleatorio(solucion):
  #Generador de 1 solucion vecina 2-opt (intercambiar 2 nodos)
  #Se puede mejorar haciendo que la elección no se uniforme sino entre las que es
  i = random.choice(range(1, len(solucion)) )
  j = random.choice(list(set(range(1, len(solucion))) - {i}))
  vecina = solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + soluc
  return vecina


def probabilidad(T,d):
  r=random.random();
  if(r <= (e**(-1*d)/(T*1.0))):
    return True
  else:
    return False

def bajar_temperatura(T):
  return T-1
```

```
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

  while TEMPERATURA > 0:
    #Genera una solución vecina(aleatoria)
    vecina = genera_vecina(solucion_referencia)

    #Calcula su valor(distancia)
    distancia_vecina = distancia_total(vecina, problem)

    #Si es la mejor solución de todas se guarda
    if distancia_vecina < mejor_distancia:
        mejor_solucion = vecina
        mejor_distancia = distancia_vecina

    #Si la nueva vecina es mejor se cambia y si es peor se cambia según una probal
```

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

      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, 100)
```

```
La mejor solución encontrada es [0, 7, 17, 31, 36, 35, 39, 21, 40, 24, 22,
con una distancia total de 1863
```

```
def Add_Nodo(problem, H ,T ) :
  #Establecer una una funcion de probabilidad para
  # añadir un nuevo nodo dependiendo de los nodos mas cercanos y de las feromonas
  Nodos = list(problem.get_nodes())
  return random.choice(   list(set(range(1,len(Nodos))) - set(H) )  )


def Incrementa_Feromona(problem, T, H):
  #Incrementar segun la calidad de la solución. Añadir una cantidad inversamente p
  for i in range(len(H)-1):
    T[H[i]][H[i+1]] += 1000/distancia_total(H, problem)
  return T

def Evaporar_Feromonas(T):
  #Podemos elegir diferentes funciones de evaporación dependiendo de la cantidad a
  #Evapora 0.3 el valor de la feromona, sin que baje de 1
  T = [[ max(T[i][j] - 0.3 , 1) for i in range(len(Nodos)) ] for j in range(len(No
  return T


def hormigas(problem, N) :
  #problem = datos del problema
  #N = Número de agentes(hormigas)

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

  #Inicializa las aristas con una cantidad inicial de feromonas:1
  T = [[ 1 for _ in range(len(Nodos)) ] for _ in range(len(Nodos))]

  #Se generan los agentes(hormigas) que serán estructuras de caminos desde 0
  Hormiga = [[0] for _ in range(N)]

  #Recorre cada agente construyendo la solución
  for h in range(N) :
    #print("\nAgente:", h)
    #Para cada agente se construye un camino
    for i in range(len(Nodos)-1) :

      #Elige el siguiente nodo
      Nuevo_Nodo = Add_Nodo(problem, Hormiga[h] ,T )

      Hormiga[h].append(Nuevo_Nodo)

    #Incrementa feromonas en esa arista
    T = Incrementa_Feromona(problem, T, Hormiga[h] )
    #print("Feromonas(1)", T)

    #Evapora Feromonas
    T = Evaporar_Feromonas(T)
    #print("Feromonas(2)", T)
```

```
    #Seleccionamos el mejor agente
  mejor_solucion = []
  mejor_distancia = 10e100
  for h in range(N) :
    distancia_actual = distancia_total(Hormiga[h], problem)
    if distancia_actual < mejor_distancia:
      mejor_solucion = Hormiga[h]
      mejor_distancia =distancia_actual


  print(mejor_solucion)
  print(mejor_distancia)


hormigas(problem, 1000)
```

```
[0, 12, 25, 38, 27, 20, 17, 6, 9, 29, 40, 22, 41, 8, 4, 18, 36, 31, 7, 15,
3750
```