Algoritmos - Luis Jama - AG3

July 19, 2023

1 Actividad Guiada 3 : El problema del agente viajero – TSP

https://github.com/ljham/03MIAR_ALG_OPTZ.git

2 Luis Jama Tello

[ ]:
```
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
import random         #Para generar valores aleatorios
```

2.0.1 1. Cargar datos del problema

[ ]:
```
#Descargamos el fichero de datos(Matriz de distancias)
file = "swiss42.tsp"
urllib.request.urlretrieve("http://comopt.ifi.uni-heidelberg.de/software/
 ↪TSPLIB95/tsp/swiss42.tsp.gz", file + '.gz')
!gzip -d swiss42.tsp.gz #Descomprimir el fichero de datos
```

[ ]:
```
#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, 1)
```

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

#dir(problem)
```

[ ]: 15

2.0.2 2. Funciones básicas

[ ]:
```
#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)
```

2.0.3 3. Búsqueda Aleatoria

[ ]:
```
################################################################################
# BUSQUEDA ALEATORIA
################################################################################

def busqueda_aleatoria(problem, N):
  #N es el numero de iteraciones
  Nodos = list(problem.get_nodes())

  mejor_solucion = []
```

```
  #mejor_distancia = 10e100                          #Inicializamos con un valor␣
 ↪alto
  mejor_distancia = float('inf')                     #Inicializamos con un valor␣
 ↪alto

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

    if distancia < mejor_distancia:                  #Compara con la mejor␣
 ↪obtenida hasta ahora
      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, 4000)
```

```
Mejor solución: [0, 37, 31, 17, 26, 7, 12, 30, 34, 35, 33, 36, 39, 18, 27, 13,
1, 6, 28, 22, 38, 23, 9, 10, 11, 40, 24, 19, 16, 21, 20, 32, 14, 41, 29, 25, 8,
15, 3, 4, 5, 2]
Distancia     : 3827
```

2.0.4 4. Búsqueda Local

[ ]:
```
##############################################################################
# BUSQUEDA LOCAL
##############################################################################

# solucion = crear_solucion(Nodos)

def genera_vecina(solucion):
  #Generador de soluciones vecinas: 2-opt (intercambiar 2 nodos) Si hay N nodos␣
 ↪se generan (N-1)x(N-2)/2 soluciones
  #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 para evaluar todos los intercambios 2-opt
    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] = [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 = [1, 47, 13, 41, 40, 19, 42, 44, 37, 5, 22, 28, 3, 2, 29, 21, 50,␣
 ↪34, 30, 9, 16, 11, 38, 49, 10, 39, 33, 45, 15, 24, 43, 26, 31, 36, 35, 20,␣
 ↪8, 7, 23, 48, 27, 12, 17, 4, 18, 25, 14, 6, 51, 46, 32]
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: 3827
Distancia Mejor Solucion Local: 3457
```

[ ]:
```
#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)
  print(f'Mejor distancia : {mejor_distancia}')

  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 operador de vencindad 2-opt)
    if distancia_vecina < mejor_distancia:
      #mejor_solucion = copy.deepcopy(vecina)   #Con copia profunda. Las copias en python son por referencia
      mejor_solucion = vecina                   #Guarda la mejor solución encontrada
      mejor_distancia = distancia_vecina

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

    solucion_referencia = vecina


sol = busqueda_local(problem)
```

```
Mejor distancia : 4805
En la iteracion  27 , la mejor solución encontrada es: [0, 26, 18, 12, 11, 2,
27, 20, 33, 34, 38, 22, 30, 29, 9, 23, 8, 19, 14, 16, 15, 37, 7, 17, 31, 36, 35,
32, 28, 10, 25, 41, 40, 24, 21, 39, 3, 4, 6, 5, 13, 1]
Distancia     : 1871
```

5. Búsquedas Basadas en trayectorias : multi-arranque

[ ]:
```
def busqueda_multiarranque(problem, num_arranques):
    mejor_solucion = None
    mejor_distancia = float('inf')

    for _ in range(num_arranques):
        solucion_referencia = crear_solucion(Nodos)  # Generar una solución inicial aleatoria
        distancia_referencia = distancia_total(solucion_referencia, problem)

        while True:
```

```
            vecina = genera_vecina(solucion_referencia)  # Generar una vecina

            distancia_vecina = distancia_total(vecina, problem)  # Calcular la␣
 ↪distancia de la vecina

            if distancia_vecina < distancia_referencia:
                solucion_referencia = vecina
                distancia_referencia = distancia_vecina
            else:
                break

        if distancia_referencia < mejor_distancia:
            mejor_solucion = solucion_referencia
            mejor_distancia = distancia_referencia

    return mejor_solucion

sol = busqueda_multiarranque(problem, 5)

print("La mejor solución encontrada es:" , sol)
print("Distancia Mejor Solucion en Multiarranque:", distancia_total(sol,␣
 ↪problem))
```

```
La mejor solución encontrada es: [0, 3, 32, 34, 33, 20, 35, 36, 31, 17, 1, 6, 4,
8, 9, 23, 21, 39, 29, 30, 28, 2, 27, 38, 22, 24, 40, 41, 10, 25, 11, 12, 18, 26,
5, 13, 19, 14, 16, 15, 37, 7]
Distancia Mejor Solucion en Multiarranque: 1574
```