josejesuslacasanieto-ag3

February 14, 2024

Actividad Guiada 3 de Algoritmos de Optimización

Nombre: José Jesús La Casa Nieto

https://colab.research.google.com/drive/1BpE_u_hnwlkSdfV7I590Dp4KgvP4TjgB?usp=sharing

https://github.com/JoseJesusLaCasaNieto/03MIAR—Algoritmos-de-Optimizacion—2024

[22]:
```
!pip install requests
!pip install tsplib95
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests) (2024.2.2)
Requirement already satisfied: tsplib95 in /usr/local/lib/python3.10/dist-
packages (0.7.1)
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.10/dist-
packages (from tsplib95) (8.1.7)
Requirement already satisfied: Deprecated~=1.2.9 in
/usr/local/lib/python3.10/dist-packages (from tsplib95) (1.2.14)
Requirement already satisfied: networkx~=2.1 in /usr/local/lib/python3.10/dist-
packages (from tsplib95) (2.8.8)
Requirement already satisfied: tabulate~=0.8.7 in
/usr/local/lib/python3.10/dist-packages (from tsplib95) (0.8.10)
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-
packages (from Deprecated~=1.2.9->tsplib95) (1.14.1)
```

[23]:
```
# Carga de librerías y datos del problema

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
```

```
#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" ;
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

#Coordendas 51-city problem (Christofides/Eilon)
#file = "eil51.tsp" ; urllib.request.urlretrieve("http://comopt.ifi.
 ↪uni-heidelberg.de/software/TSPLIB95/tsp/eil51.tsp.gz", file)

#Coordenadas - 48 capitals of the US (Padberg/Rinaldi)
#file = "att48.tsp" ; urllib.request.urlretrieve("http://comopt.ifi.
 ↪uni-heidelberg.de/software/TSPLIB95/tsp/att48.tsp.gz", file)
```

```
gzip: swiss42.tsp already exists; do you wish to overwrite (y or n)? y
```

[24]:
```
#Carga de datos y generación de objeto problem
###############################################################################
problem = tsplib95.load(file)

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

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

[25]:
```
#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)
```

[25]: 15

[26]:
```
#Funciones básicas
###############################################################################

#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)

sol_temporal = crear_solucion(Nodos)

distancia_total(sol_temporal, problem), sol_temporal
```

[26]:
```
(4854,
 [0,
  4,
  19,
  39,
  12,
  38,
  20,
  34,
  22,
  3,
  24,
  25,
  23,
  9,
  6,
  8,
  5,
  36,
  1,
```

```
  29,
  2,
  7,
  40,
  35,
  17,
  21,
  16,
  11,
  26,
  10,
  31,
  15,
  30,
  33,
  41,
  27,
  13,
  37,
  28,
  18,
  32,
  14])
```

Búsqueda aleatoria

[27]:
```
##############################################################################
# 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, 10000)
```

```
Mejor solución: [0, 30, 11, 23, 5, 12, 18, 19, 26, 31, 36, 7, 14, 35, 17, 15,
16, 13, 4, 9, 24, 40, 21, 41, 6, 32, 29, 37, 3, 1, 2, 22, 8, 25, 28, 34, 33, 27,
39, 10, 38, 20]
Distancia     : 3282
```

Búsqueda local

[28]:
```
##############################################################################
# BUSQUEDA LOCAL
##############################################################################
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: 3282
Distancia Mejor Solucion Local: 3033
```

[29]:
```
#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)

  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 )
```

```
En la iteracion  28 , la mejor solución encontrada es: [0, 26, 18, 10, 40, 24,
21, 39, 32, 31, 35, 36, 7, 1, 8, 9, 23, 41, 25, 11, 12, 13, 5, 6, 3, 27, 28, 2,
14, 16, 19, 4, 29, 30, 22, 38, 34, 33, 20, 17, 37, 15]
Distancia     : 1974
```

Recocido simulado (Simulated Annealing - SA)

[30]:
```
###############################################################################
# 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
```

[31]:
```
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 = []             #x* del seudocodigo
  mejor_distancia = 10e100        #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_referencia - distancia_vecina)
    if distancia_vecina < distancia_referencia or probabilidad(TEMPERATURA,␣
 ↪abs(distancia_referencia - distancia_vecina) ) :
      #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, 1, 7, 37, 16, 17, 31, 34, 32, 9, 8, 27, 6,
36, 35, 20, 33, 38, 39, 21, 40, 24, 22, 30, 29, 28, 4, 5, 15, 14, 19, 13, 26,
```

```
12, 18, 11, 25, 23, 41, 10, 2, 3]
con una distancia total de 1947
```

Mejorar nota Se puede mejorar el algoritmo de búsqueda local de varias maneras. En mi caso, voy a utilizar una inserción de nodo.

[32]:
```
# Búsqueda local

def genera_vecina(solucion):
    # Generador de soluciones vecinas: Inserción de nodo
    mejor_solucion = solucion[:]
    mejor_distancia = distancia_total(solucion, problem)

    for i in range(1, len(solucion)):
        for j in range(len(solucion)):
            if i != j:
                vecina = solucion[:i] + [solucion[j]] + solucion[i:j] +␣
 ↪solucion[j+1:]
                distancia_vecina = distancia_total(vecina, problem)

                if distancia_vecina < mejor_distancia:
                    mejor_solucion = vecina
                    mejor_distancia = distancia_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: 3282
Distancia Mejor Solucion Local: 3104
```

En esta implementación, se recorre cada posición de la solución actual (i) y cada nodo de la solución (j). Para cada par de posición-nodo, creamos una solución vecina insertando el nodo en la posición i. Luego calculamos la distancia total de esta solución vecina y la comparamos con la mejor solución encontrada hasta el momento. Si la distancia de la solución vecina es menor, actualizamos la mejor solución y la distancia.

Se puede mejorar el algoritmo de recocido simulado de varias maneras. En mi caso, voy a implementar una estrategia de generación de vecinos basada en la búsqueda por intercambio 2-opt.

[33]:
```
# Recocido simulado

def genera_vecina_2opt(solucion):

  # Se recorren todos los pares de nodos
```

```
  for i in range(len(solucion)-1):
    for j in range(i+1, len(solucion)):

      # Se calcula la distancia de la solución actual
      distancia_actual = distancia_total(solucion)

      # Se intercambian los nodos i y j
      vecina = solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] +␣
↪solucion[j+1:]

      # Se calcula la distancia de la solución vecina
      distancia_vecina = distancia_total(vecina)

      # Si la solución vecina es mejor, se devuelve
      if distancia_vecina < distancia_actual:
        return vecina

  # Si no se encuentra una solución vecina mejor, se devuelve la solución actual
  return solucion

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 ):
  #problem = datos del problema
  #T = Temperatura

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

  mejor_solucion = []             #x* del seudocodigo
  mejor_distancia = 10e100        #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_referencia - distancia_vecina)
    if distancia_vecina < distancia_referencia or probabilidad(TEMPERATURA,␣
 ↪abs(distancia_referencia - distancia_vecina) ) :
      #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, 3, 1, 6, 5, 26, 4, 28, 21, 38, 22, 39, 8, 9,
24, 40, 23, 41, 25, 12, 11, 10, 18, 13, 19, 31, 33, 34, 20, 27, 2, 30, 29, 32,
35, 36, 17, 15, 16, 14, 37, 7]
con una distancia total de 1906
```