Augusto Ibanez - TSP

March 1, 2024

1 Algoritmos de optimización - Reto 3

Nombre: Augusto Javier Ibañez Garcia
Github: https://github.com/cibergus/VIU-AlgOptimizacion

Elección de algoritmo: BÚSQUEDA TABÚ

Índice:

2 FUNCIONES Y ALGORITMOS BASE

2.1 Carga de librerias

[1]:
```
# INSTALLS

# !pip install requests    #Hacer llamadas http a paginas de la red
# !pip install tsplib95    #Modulo para las instancias del problema del TSP
```

[2]:
```
# IMPORTS

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
```

2.2 Carga de los datos del problema

[3]:
```
#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)
file1 = "datasets/swiss42.tsp"
ruta1 = "http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/swiss42.tsp.
 ↪gz"
urllib.request.urlretrieve(ruta1, file1 + '.gz')
!gzip -df datasets/swiss42.tsp.gz     #Descomprimir el fichero de datos

#Coordenadas 51-city problem (Christofides/Eilon)
#file2 = "datasets/eil51.tsp"
#ruta3 = "http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/eil51.tsp.
 ↪gz"
#urllib.request.urlretrieve(ruta3, file)
#!gzip -df eil51.tsp.gz

#Coordenadas - 48 capitals of the US (Padberg/Rinaldi)
#file3 = "datasets/att48.tsp"
#ruta3 = "http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/att48.tsp.
 ↪gz"
#urllib.request.urlretrieve(ruta3, file)
#!gzip -df att48.tsp.gz
```

[4]:
```
#Carga de datos y generación de objeto problem
################################################################################
problem = tsplib95.load(file1)

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

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

```
NOMBRE: swiss42
TIPO: TSP
COMENTARIO: 42 Staedte Schweiz (Fricker)
DIMENSION: 42
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_SECTION
0 15 30 23 32 55 33 37 92 114 92 110 96 90 74 76 82 72 78 82 159 122 131 206 112 57 28 43 70 (
  15 0 34 23 27 40 19 32 93 117 88 100 87 75 63 67 71 69 62 63 96 164 132 131 212 106 44 33 5:
  30 34 0 11 18 57 36 65 62 84 64 89 76 93 95 100 104 98 57 88 99 130 100 101 179 86 51 4 18 4
  23 23 11 0 11 48 26 54 70 94 69 75 75 84 84 89 92 89 54 78 99 141 111 109 89 89 11 11 11 54
  32 27 18 11 0 40 20 58 67 92 61 78 65 76 83 89 91 95 43 72 110 141 116 105 190 81 34 19 35 !
  55 40 57 48 40 0 23 55 96 123 78 75 36 36 66 66 63 95 34 34 137 174 156 129 224 90 15 59 75
  33 19 36 26 20 23 0 45 85 111 75 82 69 60 63 70 71 85 44 52 115 161 136 122 210 91 25 37 54
  37 32 65 54 58 55 45 0 124 149 118 126 113 80 42 42 40 40 87 87 94 158 158 163 242 135 65 6:
  92 93 62 70 67 96 85 124 0 28 29 68 63 122 148 155 156 159 67 129 148 78 80 39 129 46 82 65
 114 117 84 94 92 123 111 149 28 0 54 91 88 150 174 181 182 181 95 157 159 50 65 27 102 65 11(
  92 88 64 69 61 78 75 118 29 54 0 39 34 99 134 142 141 157 44 110 161 103 109 52 154 22 63 6(
 110 100 89 89 78 75 82 126 68 91 39 0 14 80 129 139 135 167 39 98 187 136 148 81 186 28 61 9:
  96 87 76 75 65 62 69 113 63 88 34 14 0 72 117 128 124 153 26 88 174 136 142 82 187 32 48 79
  90 75 93 84 76 36 60 80 122 150 99 80 72 0 59 71 63 116 56 25 170 201 189 151 252 104 44 95
  74 63 95 84 83 56 63 42 148 174 134 129 117 59 0 11 8 63 93 35 135 223 195 184 273 146 71 9!
```

[5]:
```python
#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)
```

[5]: 15

2.3 Funciones básicas

[6]:
```python
#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)
```

2.4 BÚSQUEDA ALEATORIA

[7]:
```
##############################################################################
# BUSQUEDA ALEATORIA
##############################################################################

def busqueda_aleatoria(problem, numero_iteraciones):
  Nodos = list(problem.get_nodes())

  mejor_solucion = []
  mejor_distancia = float('inf')                    #Inicializamos con un valor␣
 ↪alto

  for i in range(numero_iteraciones):                                #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_busq_aleatoria = solucion
      mejor_distancia_busq_aleatoria = distancia

  #print("Mejor solución:" , mejor_solucion_busq_aleatoria)
  #print("Distancia     :" , mejor_distancia_busq_aleatoria)
  return mejor_solucion_busq_aleatoria, mejor_distancia_busq_aleatoria
```

[8]:
```
%%time
# Ejemplo de uso: Búsqueda aleatoria con 10.000 iteraciones
solucion, distancia_busq_aleatoria = busqueda_aleatoria(problem, 10000)
solucion_aleatoria = solucion
print("Búsqueda aleatoria")
print("Mejor solución:" , solucion)
print("Distancia     :" , distancia_busq_aleatoria)
print()
```

```
Búsqueda aleatoria
Mejor solución: [0, 17, 28, 37, 27, 19, 8, 22, 5, 29, 21, 40, 3, 31, 25, 12, 11,
```

```
10, 16, 2, 30, 41, 33, 18, 20, 14, 15, 38, 7, 32, 23, 35, 9, 1, 4, 26, 24, 36,
34, 39, 13, 6]
Distancia     : 4941

CPU times: user 2.28 s, sys: 15 ms, total: 2.3 s
Wall time: 2.31 s
```

2.5 BÚSQUEDA LOCAL

[9]:
```
##############################################################################
# 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
```

[10]:
```
#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 Inicial    :", distancia_total(solucion, problem))

nueva_solucion = genera_vecina(solucion)
distancia_genera_vecina = distancia_total(nueva_solucion, problem)
print("Distancia Mejor Solucion Local:", distancia_genera_vecina)
print("Solucion Local:", nueva_solucion)
```

```
Distancia Solucion Inicial    : 4941
Distancia Mejor Solucion Local: 4347
```

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

[11]:
```
#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
    #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 = vecina                   #Guarda la mejor solución␣
 ↪encontrada
      mejor_distancia = distancia_vecina
    else:
      print("En la iteracion ", iteracion, ", la mejor solución encontrada es:
 ↪\n" , mejor_solucion)
      print("Distancia     :" , mejor_distancia)
      return mejor_solucion, mejor_distancia
    solucion_referencia = vecina
```

[12]:
```
#sol = busqueda_local(problem)
solucion_busq_local_ruta, solucion_busq_local_distancia =␣
 ↪busqueda_local(problem)
```

```
En la iteracion  31 , la mejor solución encontrada es:
 [0, 35, 36, 26, 12, 11, 18, 5, 16, 14, 7, 38, 22, 24, 40, 41, 23, 21, 39, 9, 8,
28, 3, 1, 20, 33, 34, 32, 30, 29, 10, 25, 13, 19, 15, 37, 17, 31, 6, 4, 2, 27]
Distancia     : 2040
```

2.6 SIMULATED ANNEALING

[13]:
```
##############################################################################
# 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
```

[14]:
```
def recocido_simulado(problem, 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, mejor_distancia
```

[15]:
```
#sol = recocido_simulado(problem, 1000000)
solucion_SA_ruta, solucion_SA_distancia = recocido_simulado(problem, 1000000)
```

```
La mejor solución encontrada es [0, 12, 11, 23, 24, 40, 21, 39, 29, 30, 9, 8,
10, 25, 41, 22, 38, 34, 33, 7, 19, 13, 5, 6, 1, 35, 36, 37, 15, 16, 14, 17, 31,
20, 32, 28, 26, 18, 4, 3, 2, 27]
con una distancia total de 1928
```

3 SOLUCIÓN PROPUESTA: BÚSQUEDA TABÚ

3.1 Búsqueda Tabú - versión inicial

[16]:
```
def busqueda_tabu(problem, iteraciones, tamaño_tabu):
    # Inicialización
    solucion_referencia = crear_solucion(list(problem.get_nodes()))
    mejor_solucion = list(solucion_referencia)
    mejor_distancia = distancia_total(mejor_solucion, problem)
    lista_tabu = []
    for iteracion in range(iteraciones):
        solucion_vecina = genera_vecina(solucion_referencia)
        distancia_vecina = distancia_total(solucion_vecina, problem)
        # Si la solución vecina es mejor que la mejor solución encontrada y no␣
 ↪está en la lista tabú
        if distancia_vecina < mejor_distancia and solucion_vecina not in␣
 ↪lista_tabu:
            mejor_solucion = solucion_vecina
            mejor_distancia = distancia_vecina
            solucion_referencia = solucion_vecina
            # Actualizar la lista tabú
            lista_tabu.append(solucion_vecina)
            if len(lista_tabu) > tamaño_tabu:
                lista_tabu.pop(0)
    return mejor_solucion, mejor_distancia
```

[17]:
```
# Ejemplo de llamada a la función
solucion_busq_tabu_ruta, solucion_busq_tabu_distancia = busqueda_tabu(problem,␣
 ↪100, 10)
print("Distancia Búsqueda Tabú :", solucion_busq_tabu_distancia)
```

```
Distancia Búsqueda Tabú : 1841
```

3.2 Búsqueda Tabú - versión mejorada

[18]:
```
def busqueda_tabu_v2(problem, iteraciones, tamaño_tabu, temperatura):
    solucion_referencia = crear_solucion(list(problem.get_nodes()))
    mejor_solucion = list(solucion_referencia)
    mejor_distancia = distancia_total(mejor_solucion, problem)
    lista_tabu = []
    for iteracion in range(iteraciones):
        solucion_vecina = genera_vecina(solucion_referencia)
        distancia_vecina = distancia_total(solucion_vecina, problem)
        if distancia_vecina < mejor_distancia and solucion_vecina not in␣
 ↪lista_tabu:
            mejor_solucion = solucion_vecina
            mejor_distancia = distancia_vecina
            solucion_referencia = solucion_vecina
            lista_tabu.append(solucion_vecina)
            if len(lista_tabu) > tamaño_tabu:
                lista_tabu.pop(0)
        elif solucion_vecina not in lista_tabu:
            delta = distancia_vecina - distancia_total(solucion_referencia,␣
 ↪problem)
            if delta < 0 or probabilidad(temperatura, abs(delta)):
                solucion_referencia = solucion_vecina
                distancia_referencia = distancia_vecina
                lista_tabu.append(solucion_vecina)
                if len(lista_tabu) > tamaño_tabu:
                    lista_tabu.pop(0)
                temperatura = bajar_temperatura(temperatura)
    return mejor_solucion, mejor_distancia
```

[19]:
```
# Ejemplo de llamada a la función
temperatura_inicial= 1000000
solucion_busq_tabu_ruta_v2, solucion_busq_tabu_distancia_v2 =␣
 ↪busqueda_tabu_v2(problem, 100, 10, temperatura_inicial)
print("Distancia Búsqueda Tabú v.2:", solucion_busq_tabu_distancia_v2)
```

```
Distancia Búsqueda Tabú v.2: 1767
```

4 RESUMEN Y CONCLUSIONES

[20]:
```
# Distancias
print("DISTANCIAS:")
print("Solucion Aleatoria  :", distancia_busq_aleatoria)
print("Búsqueda Local      :", solucion_busq_local_distancia)
print("Simulated Annealing :", solucion_SA_distancia)
print("Búsqueda Tabú       :", solucion_busq_tabu_distancia)
print("Búsqueda Tabú v.2   :", solucion_busq_tabu_distancia_v2)
```

```
DISTANCIAS:
Solucion Aleatoria  : 4941
Búsqueda Local      : 2040
Simulated Annealing : 1928
Búsqueda Tabú       : 1841
Búsqueda Tabú v.2   : 1767
```

[21]:
```python
# Rutas
print("RUTAS:")
print("Solución Aleatoria  :\n", solucion_aleatoria)
print("Búsqueda Local      :\n", solucion_busq_local_ruta)
print("Simulated Annealing :\n", solucion_SA_ruta)
print("Búsqueda Tabú       :\n", solucion_busq_tabu_ruta)
print("Búsqueda Tabú v.2   :\n", solucion_busq_tabu_ruta_v2)
```

```
RUTAS:
Solución Aleatoria  :
 [0, 17, 28, 37, 27, 19, 8, 22, 5, 29, 21, 40, 3, 31, 25, 12, 11, 10, 16, 2, 30,
41, 33, 18, 20, 14, 15, 38, 7, 32, 23, 35, 9, 1, 4, 26, 24, 36, 34, 39, 13, 6]
Búsqueda Local      :
 [0, 35, 36, 26, 12, 11, 18, 5, 16, 14, 7, 38, 22, 24, 40, 41, 23, 21, 39, 9, 8,
28, 3, 1, 20, 33, 34, 32, 30, 29, 10, 25, 13, 19, 15, 37, 17, 31, 6, 4, 2, 27]
Simulated Annealing :
 [0, 12, 11, 23, 24, 40, 21, 39, 29, 30, 9, 8, 10, 25, 41, 22, 38, 34, 33, 7,
19, 13, 5, 6, 1, 35, 36, 37, 15, 16, 14, 17, 31, 20, 32, 28, 26, 18, 4, 3, 2,
27]
Búsqueda Tabú       :
 [0, 5, 19, 13, 26, 18, 23, 41, 25, 11, 12, 10, 8, 9, 21, 39, 30, 32, 35, 36,
17, 31, 3, 2, 27, 34, 20, 33, 38, 22, 24, 40, 29, 28, 4, 14, 16, 15, 37, 7, 6,
1]
Búsqueda Tabú v.2   :
 [0, 27, 2, 4, 18, 12, 11, 25, 41, 23, 21, 40, 24, 39, 31, 17, 36, 35, 33, 20,
3, 6, 1, 7, 37, 15, 16, 14, 19, 13, 5, 26, 10, 8, 9, 29, 30, 28, 32, 34, 38, 22]
```

- La implementación de la búsqueda tabú para el TSP ha demostrado ser una estrategia efectiva para mejorar las soluciones iniciales del problema.

- La versión inicial proporcionó una mejora sustancial sobre enfoques más simples como la búsqueda aleatoria y local, mientras que la versión mejorada, al incorporar un mecanismo de aceptación de soluciones peores basado en la probabilidad, ha permitido explorar más ampliamente el espacio de soluciones y escapar de óptimos locales, resultando en una mejora adicional.

- Estos resultados subrayan la importancia de la exploración y la explotación equilibradas en los algoritmos de optimización, así como el valor de las estrategias adaptativas para navegar eficientemente por paisajes complejos de soluciones.

[ ]: