Algoritmos Genéticos

UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL ROSARIO

Ornella Corsetti: orcorsetti@gmail.com

Santiago Ciriaci: santiagociriaci@gmail.com

Andrés Botello: andresjbotello@gmail.com

GRUPO 2 Ciclo Lectivo 2018

TRABAJO PRACTICO 3 – PROBLEMA DEL VIAJANTE.

INDICE

ENUNCIADO

1. Hallar la ruta de distancia mínima que logre unir todas las capitales de provincias de la República Argentina, utilizando un método exhaustivo. ¿Puede resolver el problema? Justificar de manera teórica.

2. Realizar un programa que cuente con un menú con las siguientes opciones:

a) Permitir ingresar una provincia y hallar la ruta de distancia mínima que logre unir todas las capitales de provincias de la República Argentina partiendo de dicha capital utilizando la siguiente heurística: "Desde cada ciudad ir a la ciudad más cercana no visitada." Recordar regresar siempre a la ciudad de partida. Presentar un mapa de la República con el recorrido indicado. Además, indicar la ciudad de partida, el recorrido completo y la longitud del trayecto. El programa deberá permitir seleccionar la capital que el usuario desee ingresar como inicio del recorrido.

b) Encontrar el recorrido mínimo para visitar todas las capitales de las provincias de la República Argentina siguiendo la heurística mencionada en el punto a. Deberá mostrar como salida el recorrido y la longitud del trayecto.

c) Hallar la ruta de distancia mínima que logre unir todas las capitales de provincias de la República Argentina, utilizando un algoritmo genético. Recomendaciones para el algoritmo: N = 50 Número de cromosomas de las poblaciones. M = 200 Cantidad de ciclos. Cromosomas: permutaciones de 23 números naturales del 1 al 23 donde cada gen es una ciudad. Las frecuencias de crossover y de mutación quedan a criterio del grupo. Se deberá usar crossover cíclico.

SOLUCIONES

EJERCICIO 1:

Para este problema no es válido utilizar el método exhaustivo porque deberíamos generar una cantidad extremadamente grande (24! = 620448401733239439360000) de posibles recorridos que llevaría una cantidad de tiempo inadecuada.

CÓDIGO EJERCICIO 2

```
from random import randint
from random import random
import xlwt
opc=9
while (opc!=4):
   opc=9
   while opc<1 or opc>4:      #Valida Opción
      opc=int(input('1)Heurística  2)Recorrido mínimo  3)Algoritmo Genético  4)Salir : '))
      #Acá agregaremos la tabla

tab=[[0,646,792,933,53,986,985,989,375,834,1127,794,2082,979,1080,1334,1282,1005,749,393,579,939,2373,799],

[646,0,677,824,698,340,466,907,348,919,1321,699,2281,362,517,809,745,412,293,330,577,401,2618,1047],

[792,677,0,157,830,814,1131,1534,500,291,1845,13,2819,691,633,742,719,1039,969,498,1136,553,3131,1527],

[933,824,157,0,968,927,1269,1690,656,263,1999,161,2974,793,703,750,741,1169,1117,654,1293,629,3284,1681],

[53,698,830,968,0,1038,1029,1005,427,857,1116,833,2064,1030,1132,1385,1333,1053,795,444,602,991,2350,789],

[986,340,814,927,1038,0,427,1063,659,1098,1548,802,2473,149,330,600,533,283,435,640,834,311,2821,1311],
```

[985,466,1131,1269,1029,427,0,676,790,1384,1201,1121,2081,569,756,1323,957,152,245,775,586,713,2435,1019],

[989,907,1534,1690,1005,1063,676,0,1053,1709,543,1529,1410,1182,1370,1658,1591,824,643,1049,422,1286,1762,479],

[375,348,500,656,427,659,790,1053,0,658,1345,498,2320,622,707,959,906,757,574,19,642,566,2635,1030],

[834,919,291,263,857,1098,1384,1709,658,0,1951,305,2914,980,924,1007,992,1306,1200,664,1293,827,3207,1624],

[1127,1321,1845,1999,1116,1548,1201,543,1345,1951,0,1843,975,1647,1827,2120,2054,1340,1113,1349,745,1721,1300,327],

[794,669,13,161,833,802,1121,1529,498,305,1843,0,2818,678,620,729,706,1029,961,495,1132,523,3130,1526],

[2082,2281,2819,2974,2064,2473,2081,1410,2320,2914,975,2818,0,2587,2773,3063,2997,2231,2046,2325,1712,2677,359,1294],

[979,362,691,793,1030,149,569,1182,622,980,1647,678,2587,0,189,477,410,430,540,602,915,166,2931,1391],

[1080,517,633,703,1132,330,756,1370,707,924,1827,620,2773,189,0,293,228,612,727,689,1088,141,3116,1562],

[1334,809,742,750,1385,600,1023,1658,959,1007,2120,729,3073,477,293,0,67,874,1017,942,1382,414,3408,1855],

[1282,745,719,741,1333,533,957,1591,906,992,2054,706,2997,410,228,67,0,808,950,889,316,353,3341,1790],

[1005,412,1039,1169,1053,283,152,824,757,1306,1340,1029,2231,430,612,874,808,0,248,740,686,583,2585,1141],

[749,293,969,1117,795,435,235,643,574,1200,1113,961,2046,540,727,1017,950,284,0,560,412,643,2392,882],

[393,330,498,654,444,640,775,1049,19,664,1349,495,2325,602,689,942,889,740,560,0,641,547,2641,1035],

[579,577,1136,1293,602,834,586,422,642,1293,745,1132,1712,915,1088,1382,1316,686,412,641,0,977,2044,477],

[939,401,535,629,991,311,713,1286,566,827,1721,523,2677,166,141,414,353,583,643,547,977,0,3016,1446],

[2373,2618,3131,3284,2350,2821,2435,1762,2635,3207,1300,3130,359,2931,3116,3408,3341,2585,2392,2641,2044,3016,0,1605],

[799,1047,1527,1681,789,1311,1019,479,1030,1624,327,1526,1294,1391,1562,1855,1790,1141,882,1035,477,1446,1605,0]]

```
        nombres=['Cdad. de Bs. As.','Cordoba','Corrientes','Formosa','La Plata','La Rioja','Mendoza','Neuquen','Parana','Posadas','Rawson','Resistencia','Rio Gallegos','S.F.d.V.d. Catamarca','S.M.d. Tucuman','S.S.d. Jujuy','Salta','San Juan','San Luis','Santa Fe','Santa Rosa','Sgo. del Estero','Ushuaia','Viedma']
        if (opc==1): #Heurística
            cap=30
            while (cap<1)or(cap>24):    #Selecciona Ciudad Inicial
                print('Seleccione la Capital donde desea comenzar:')
                cap=int(input('1-Cdad. de Bs. As. \n2-Cordoba \n3-Corrientes \n4-Formosa \n5-La Plata \n6-La Rioja \n7-Mendoza \n8-Neuquen \n9-Parana \n10-Posadas \n11-Rawson \n12-Resistencia \n13-Rio Gallegos \n14-S.F.d.V.d. Catamarca \n15-S.M.d. Tucuman \n16-S.S.d. Jujuy \n17-Salta \n18-San Juan \n19-San Luis \n20-Santa Fe \n21-Santa Rosa \n22-Sgo. del Estero \n23-Ushuaia \n24-Viedma \nOpcion Deseada:  '))
            partida=cap-1
            cap=cap-1
            rec=[]
            for r1 in range (2): #Inicializa lista de Recorridos
                rec.append([])
            enc=[]
```

```
            rec[0].append(cap)
            rec[1].append(0)
            for i in range (24): #Inicializa lista para validar ciudades recorridas
               enc.append(False)
            enc[cap]=True
            for i in range (23): #Recorre las 23 Ciudades desde la inicial
               mini=max(tab[cap])+1 #Guarda un valor por encima de todas las distancias
               for ca in range (24):
                  capi=tab[cap][ca]
                  if (tab[cap][ca]<mini) and (enc[ca]==False) and (tab[cap][ca]!=0): #Valida que
sea la distancia menor que no este recorrida y sea distinto de cero
                     mini=tab[cap][ca]
               cap=tab[cap].index(mini) #Busca numero de ciudad de la menor distancia
               enc[cap]=True #Marcamos la ciudad como recorrida
               rec[0].append(cap) #Agregamos Ciudad
               rec[1].append(mini) #Agregamos su recorrido
            rec[0].append(partida) #Volvemos a provincia inicial
            rec[1].append(tab[cap][partida])
            print('Punto de Partida: ',nombres[partida],'.') #Muestra punto de partida
            print('Recorrido: ') #Muestra recorrido
            for r in range (25):
               print(r+1,' - ', nombres[rec[0][r]])
            print('Distancia Total: ',sum(rec[1])) #Muestra distancia total

      if (opc==2): #Mejor Recorrido
            minimo=[] #Creamos la variable mínimo para almacenar la de menor distancia
            for i in range(2):
               minimo.append([])
            for capi in range (24): #Lo corremos 24 veces por las 24 provincias
               cap=capi
               partida=cap
               rec=[]
               for r1 in range (2):
                  rec.append([])
               enc=[]
               rec[0].append(cap)
               rec[1].append(0)
```

```
            for i in range (24):
                enc.append(False)
            enc[cap]=True
            for i in range (23):
                mini=max(tab[cap])+1
                for ca in range (24):
                    capi=tab[cap][ca]
                    if (tab[cap][ca]<mini) and (enc[ca]==False) and (tab[cap][ca]!=0):
                        mini=tab[cap][ca]
                cap=tab[cap].index(mini)
                enc[cap]=True
                rec[0].append(cap)
                rec[1].append(mini)
            rec[0].append(partida)
            rec[1].append(tab[cap][partida])
            if partida==0: #Seleccionamos la primera como mínimo por defecto
                minimo=rec
            if sum(minimo[1])>sum(rec[1]): #Comparamos si es menor a la anterior y la guardamos si se cumple
                minimo=rec
        print('Recorrido: ') #Mostramos mejor recorrido
        for r in range (25):
            print(r+1,' - ', nombres[minimo[0][r]])
        print('Distancia Total: ',sum(minimo[1])) #Mostramos distancia total

    if(opc==3): #Algoritmo Genético
        wb=xlwt.Workbook() #Creamos hoja de Excel
        ws=wb.add_sheet('Problema del Viajante')
        pc=0.85 #Probabilidad de Crossover
        pm=0.15 #Probabilidad de Mutación
        lista=[]
        cr=[]
        for i in range(201):
            cr.append([])
            for j in range(50):
                cr[i].append([])
        for j in range(50):  #Crea población inicial
```

```
        for i in range(24):
            v=False
            while (v==False):
                x=randint(0,23)
                if(x not in cr[0][j]):
                    v=True
                    cr[0][j].append(x)
    for x in range(200):
        rec=[]
        suma=0
        reco=[]
        for j in range(50): #Calcula distancia total e individual
            tot=0
            for k in range(23):
                cact=cr[x][j][k]  #Ciudad actual
                cdes=cr[x][j][k+1]  #Ciudad destino
                tot=tot+tab[cact][cdes]
            rec.append(tot)
            reco.append(tot) #Guardamos los valores totales del recorrido
        for r in range (50):
            rec[r]=100000-rec[r]
        suma=sum(rec) #Total de distancia recorrida por todos los cromosomas
        fit=[]
        for j in range(50): #Calculamos funcion fitness
            if(round(rec[j]*10000/suma)==100):
                fit.append(10000-100)
            else:
                fit.append(round(rec[j]*10000/suma))
        ru=[]#Ruleta
        for i in range(50): #"Carga Ruleta"
            for j in range (fit[i]):
                ru.append(i)
        recmin=sorted(rec,reverse=True) #Ordenamos de Menor a Mayor según las distancias
        for elit in range (10): #Pasan los mejores 10
            cr[x+1][elit]=cr[x][rec.index(recmin[elit])]
        for ind in range (20): #Cruzamos 40 padres para obtener 40 hijos
```

```
                op=False
                cr1=ru[randint(0, len(ru)-1)] #Selecciona Candidatos
                cr2=ru[randint(0, len(ru)-1)]
                cr1=cr[x][cr1]
                cr2=cr[x][cr2]
                pc1=random()
                if (pc1<pc): #Controla Probabilidad
                   hij1=[]
                   hij2=[]
                   for h in range (24): #Crea los hijos
                      hij1.append(24)
                      hij2.append(24)
                   enc=[]
                   for i in range (24): #Genera una lista para controlar los hijos recorridos
                      enc.append(False)
                   ini=0 #El Crossover inicia en la posición 0
                   it=1
                   while (op==False): #Itera hasta que se recorre todo el cromosoma
                      valpri=cr1[ini] #Saca el valor a compara de ambos padres
                      valseg=cr2[ini]
                      while (valpri!=valseg) and (False in enc): #Controla que sean distintos y no
esten intercambiados
                         if (it%2==0): #Si la iteracion es par
                            hij1[ini]=cr2[ini] #Asigna gen de cromosomas a sus hijos y lo registra
                            hij2[ini]=cr1[ini]
                            enc[ini]=True
                         else: #Si la iteracion es impar
                            hij1[ini]=cr1[ini]
                            hij2[ini]=cr2[ini]
                            enc[ini]=True
                         if False in enc: #Controla que exista algún valor sin recorrer y mueve
el indice
                            if enc[cr1.index(cr2[ini])]==False:
                               ini=cr1.index(cr2[ini])
                            else:
                               ini=enc.index(False)
                            valpri=cr1[ini] #Saca los valores nuevos para volver a controlar
```

```
                valseg=cr2[ini]
            if (valpri==valseg): #Controla sin son iguales y lo asigna
                if (it%2==0):
                    hij1[ini]=cr2[ini]
                    hij2[ini]=cr1[ini]
                    enc[ini]=True
                else:
                    hij1[ini]=cr1[ini]
                    hij2[ini]=cr2[ini]
                    enc[ini]=True
                if False in enc:
                    ini=enc.index(False)
            it+=1 #Aumenta el número de iteración
            if not(False in enc):
                op=True
        cr[x+1][ind*2+10]=hij1 #Asignamos los hijos a la siguiente generación
        cr[x+1][ind*2+11]=hij2
    else: #Si no se cumple la probabilidad van los padres
        cr[x+1][ind*2+10]=cr1
        cr[x+1][ind*2+11]=cr2
    pm1=random()
    if pm>=pm1: #Swap Mutation #Analiza probabilidad de mutación
        p1=randint(0,23) #Seleccionamos los dos genes a intercambiar
        p2=randint(0,23)
        aux=cr1[p1] #Intercambiamos los genes
        cr1[p1]=cr1[p2]
        cr1[p2]=aux
        aux=cr2[p1]
        cr2[p1]=cr2[p2]
        cr2[p2]=aux
        cr[x+1][ind*2+10]=cr1 #Asignamos los padres mutados
        cr[x+1][ind*2+11]=cr2
  ws.write(x,0,min(reco)) #Guardamos en el Excel el recorrido mínimo
print('Distancia Minima: ',min(reco),' Km.') #Mostramos distancia mínima
indice=reco.index(min(reco))
for i in range (23): #Mostramos recorrido mínimo
  print (i+1,' - ',nombres[cr[199][indice][i]],'.')
```

wb.save(r'C:\Users\Public\Documents\Problema del Viajante.xls') **#Guardamos la hoja de Excel**

CONCLUSIONES

CONCLUSIÓN HEURÍSTICA

Con este método nos aseguramos de encontrar un recorrido considerablemente más óptimo que en el algoritmo genético ya que se evalúa de ciudad a ciudad la menor distancia posible.

Hemos obtenido como resultado que, partiendo desde Neuquén, el recorrido por el resto de las capitales y volviendo a la misma es de 9335km. Siendo esta la mejor combinación.

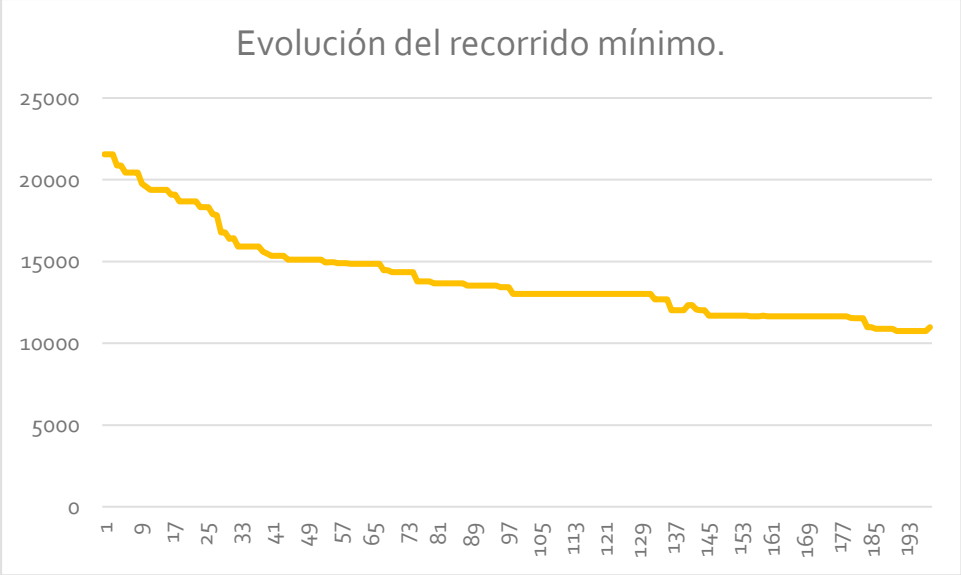
CONCLUSIÓN DE ALG. GENÉTICO (CON ELITISMO)

Se puede notar que en comparación al método anterior no llega a un óptimo tan aceptable. Debido a que analiza según a un cromosoma generado aleatoriamente y no comparando la menor distancia posible. En la mayoría de los casos se observa una diferencia de 1500km como mínimo entre un método y otro.

En las primeras iteraciones se observa una evolución más notable, la cual deducimos que sucede debido a que los padres son variados entre sí, mientras que en las últimas iteraciones el algoritmo se normaliza a los padres que resultaron más óptimos.

Con respecto a la mutación podemos observar que no se presentan cambios drásticos en las distancias. Esto sucede porque optamos por una mutación que no altere tan agresivamente el cromosoma. Dicha mutación es la Swap Mutation la cual consiste en intercambiar dos genes aleatorios del cromosoma.

Optamos por una probabilidad de mutación de 0,15 porque notamos que con un valor menor a este se atasca el algoritmo. Por otro lado seleccionamos una probabilidad de crossover elevada de 0,85 para tener mayor probabilidad de intercambio y por ende mejores generaciones.


Evolución del recorrido mínimo.
25000
20000
15000
10000
5000
0
1
9
17
25
33
41
49
57
65
73
81
89
97
105
113
121
129
137
145
153
161
169
177
185
193

APLICACIONES

Aplicaciones en internet

Supongamos que el viajante de comercio es un bit de datos, y que las ciudades son servidores de Red distribuidos por todo el planeta. Esta variante del problema del viajante de comercio es algo inherente al uso óptimo de una plataforma masiva de distribución como es Internet. No olvidemos que en cada ruta puede haber miles de ciudades en este caso. Es curioso ya que para resolver esta variante algunos investigadores se han inspirado en el comportamiento de las hormigas.

Problema de placa de circuitos impresos PCB

Ésta es sin duda una de las utilidades más ingeniosas que puede plantear el problema del viajante de comercio: la creación de placas de circuitos. Este problema se enfoca en dos subproblemas: el orden óptimo de taladrar las placas y los caminos óptimos que comunican los chips.

En los **problemas de perforado** hemos de tomar las ciudades como las posiciones a perforar y las distancias entre ellas como el tiempo que necesita la máquina en trasladarse de una otra. El punto inicial y final será un punto adicional donde permanece la perforadora mientras descansa. Claramente si estas máquinas no son correctamente programadas el tiempo que tarde en recorrer un orificio u otro puede ser significativo con lo que la producción de placas bajaría en un período de tiempo.

En el **problema de conexión de chips** la idea es minimizar la cantidad de cable necesaria para unir todos los puntos de una placa sin que haya interferencias. Como los chips son de pequeño tamaño no se pueden poner más de dos cables en un único pin. Tomando las ciudades como los pins y la cantidad de cable necesaria para unirlos como la distancia, el problema es equivalente al de viajante de comercio.