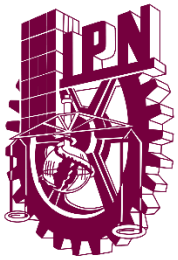

Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

Ingeniería Biónica

Inteligencia Artificial

Práctica 3. “Implementación del algoritmo de optimización ACO (Ants Colony Optimization) para el calculo de ruta óptima dentro del Sistema de Transporte Colectivo (METRO)”

Integrantes:
Aldaco Martínez Aldaco Iván
Coconi Tovar Dafne Natalia
Vargas Téllez Axel Dalí

Profesor: Álvaro Anzueto Ríos

Ciudad de México al 14 de diciembre del 2020

Presentación

Se realiza el algoritmo Ants Colony Optimization para encontrar la ruta óptima entre dos estaciones del sistema de transporte colectivo Metro de la CDMX.

Marco Teórico

Los problemas de optimización son muy importantes tanto en el campo científico como en el industrial. Algunos ejemplos de la vida real de estos problemas de optimización son la programación de horarios, la programación de la distribución del tiempo de enfermería, la programación de trenes, la planificación de la capacidad, los problemas de los vendedores ambulantes, los problemas de las rutas de los vehículos, el problema de la programación de las tiendas de grupo, la optimización de la cartera, etc. Muchos algoritmos de optimización se desarrollan por esta razón. La optimización de la colonia de hormigas es uno de ellos. La optimización de las colonias de hormigas es una técnica probabilística para encontrar rutas óptimas. En la informática y la investigación, el algoritmo de optimización de colonias de hormigas se utiliza para resolver diferentes problemas de computación.

La optimización de las colonias de hormigas (ACO) fue introducida por primera vez por Marco Dorigo en los años 90 en su tesis doctoral. Este algoritmo se introduce basado en el comportamiento de búsqueda de alimento de una hormiga para buscar un camino entre su colonia y la fuente de alimento. Inicialmente, fue usado para resolver el conocido problema del vendedor viajero. Más tarde, se utiliza para resolver diferentes problemas de optimización.

Las hormigas son insectos sociales. Viven en colonias. El comportamiento de las hormigas está controlado por el objetivo de buscar comida. Mientras buscan, las hormigas vagan por sus colonias. Una hormiga salta repetidamente de un lugar a otro para encontrar la comida. Mientras se mueve, deposita un compuesto orgánico llamado feromona en el suelo. Las hormigas se comunican entre sí a través de los rastros de feromonas. Cuando una hormiga encuentra alguna cantidad de comida, transporta todo lo que puede llevar. Al regresar deposita feromonas en los senderos según la cantidad y calidad del alimento. Las hormigas pueden oler las feromonas. Así que otras hormigas pueden olerla y seguir ese camino. Cuanto más alto sea el nivel de feromonas, mayor será la probabilidad de elegir ese camino y cuantas más hormigas lo sigan, la cantidad de feromonas también aumentará en ese camino.

Veamos un ejemplo de esto. Consideremos que hay dos caminos para llegar a la comida de la colonia. En primer lugar, no hay feromonas en el suelo. Por lo tanto, la probabilidad de elegir estos dos caminos es igual, es decir, el 50%. Consideremos que dos hormigas escogen dos caminos diferentes para llegar a la comida ya que la probabilidad de escoger estos caminos es del 50%. Fig1

Figura 1

Las distancias de estos dos caminos son diferentes. Las hormigas que sigan el camino más corto llegarán a la comida antes que el otro. Fig 2

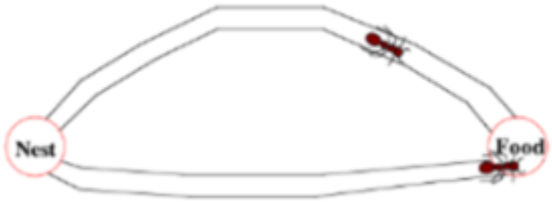

Figura 2

Después de encontrar comida, se lleva algo de comida consigo y regresa a la colonia. Cuando sigue el camino de regreso, deposita feromonas en el suelo. La hormiga que siga el camino más corto llegará antes a la colonia. Fig 3

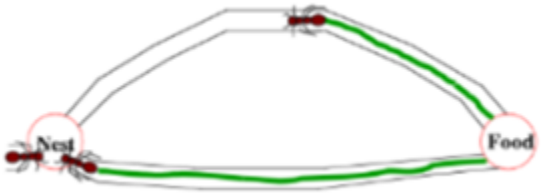

Figura 3

Cuando la tercera hormiga quiera salir a buscar comida seguirá el camino que tiene una distancia más corta según el nivel de feromonas en el suelo. Como el camino más corto tiene más feromonas que el más largo, la tercera hormiga seguirá el camino que tiene más feromonas. Fig 4

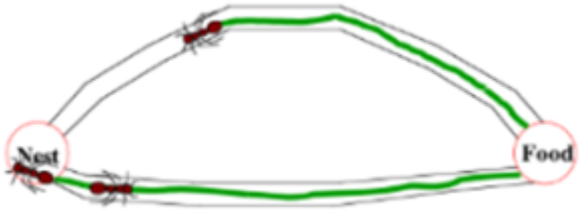

Figura 4

Para cuando la hormiga que seguía el camino más largo regresó a la colonia, más hormigas ya han seguido el camino con más nivel de feromonas. Entonces cuando otra hormiga intenta alcanzar el destino(comida) de la colonia encontrará que cada camino tiene el mismo nivel de feromonas. Así que elige uno al azar. Consideremos que escoge el anterior (Fig 5)

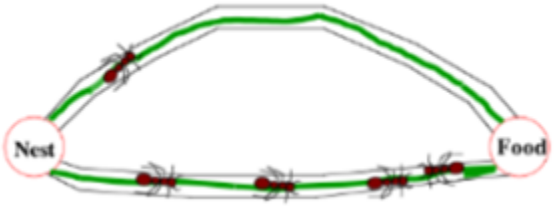

Figura 5

Repitiendo este proceso una y otra vez, después de algún tiempo, el camino más corto tiene un nivel de feromonas más alto que otros y tiene una mayor probabilidad de seguir el camino, y todas las hormigas la próxima vez seguirán el camino más corto. Fig 6

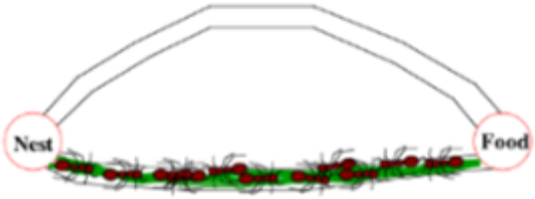

Figura 6

Para resolver los diferentes problemas con el ACO, hay tres diferentes versiones propuestas de Ant-System:

Densidad y cantidad de hormigas: La feromona se actualiza en cada movimiento de una hormiga de un lugar a otro.
Ciclo de la hormiga: La feromona se actualiza después de que todas las hormigas hayan completado su recorrido.

Veamos el pseudocódigo para aplicar el algoritmo de optimización de la colonia de hormigas. Una hormiga artificial está hecha para encontrar la solución óptima. En el primer paso para resolver un problema, cada hormiga genera una solución. En el segundo paso, se comparan los caminos encontrados por diferentes hormigas. Y en el tercer paso, se actualiza el valor del camino o feromona. Fig 7

```
procedure ACO_MetaHeuristic is
    while not_termination do
        generateSolutions()
        daemonActions()
        pheromoneUpdate()
    repeat
end procedure
```

Figura 7

Actualización de feromonas

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_{k} \Delta\tau_{xy}^{k}$$

Ec (1)

El lado izquierdo de la ecuación indica la cantidad de feromona en el borde dado x,y
ρ - la tasa de evaporación de feromonas
Y el último término en el lado derecho indicaba la cantidad de feromonas depositadas.

$$\Delta\tau_{xy}^{k} = \begin{cases} Q/L_k & \text{if ant } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

Donde L es el costo de una gira de hormigas y Q es una constante. [1]

Desarrollo

Código de distancia de cada estación

Se le otorgó un número a cada estación del metro, comenzando a numerarlo por la línea de metro. Esta numeración se muestra en la figura 8.

Figura 8. Numeración de cada estación.

Para encontrar la distancia de cada ecuación se realizó un código donde el usuario seleccionara las estaciones en las que se quiere medir la distanica.

1. Se ingresa y se muestra la imagen.

```
Ima=io.imread('NODOS.jpeg')
plt.figure()
plt.imshow(Ima)
```

2. Se seleccionan los puntos, se mide la distancia de píxeles entre ellos y se indica cuáles fueron. Después se coloca esa distancia calculada en una matriz.

```
matriz_distancia=np.load('Distancias_todos.npy')
k=1
while (k>0):
    print('Seleccione los dos nodos')
    punto=np.int32(plt.ginput(-1))
    distancia=np.sqrt((punto[0,0]-punto[1,0])**2+(punto[0,1]-punto[1,1])**2)
    n=int(input("Ingrese punto n: "))
    m=int(input("Ingrese punto m: "))
    if (n==0):
        break
    matriz_distancia[n,m]=distancia
```

Código de distancia entre nodos

Después con la matriz de distancia de cada estación se calculó la distancia entre los nodos, que serían las estaciones donde se conecta cada línea.

1. Se ingresa en una lista llamada **Nodos** las estaciones donde se unen las líneas. En un diccionario llamado **Lineas_metro** se colocan los números que cada línea del metro tiene.

```
Nodos=[2,8,9,11,13,14,20,23,30,31,35,40,42,44,48,52,55,57,59,60,62,68,71,88]
Lineas_metro={1:[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20],
              2:[21,22,23,24,25,36,27,28,29,30,31,32,33,11,34,35,36,37,38],
              3:[39,40,41,42,43,44,30,45,8,46,47,48,49,50,51],
              4:[52,53,54,55,56,57,13,58,59,60],
              5:[61,62,63,42,64,65,55,66,67,68,69,70,20],
              6:[71,72,73,74,75,76,62,77,40,78,52],
              7:[71,79,80,81,23,82,83,84,85,2,86,87],
              8:[88,31,89,9,90,91,35,92,60,93],
              9:[2,94,95,48,96,35,59,97,98,99,100,20],
              10:[20,101,102,103,104],
              11:[105,44,88,106,107,57,14,108,109,68,110,111,112,113,114,115,116,117,118,119]}
```

2. Se ingresó la matriz de distancias que se hizo en el código anterior. Se contabiliza cuántos nodos hay y se hace una matriz de distancia según la cantidad de nodos.

```
Matriz_01=np.load('Distancias_todos_completa.npy')
cant_n=len(Nodos)
M_distancia = np.zeros([cant_n,cant_n])
```

3. Se ingresa qué distancia de nodo se mide con la siguiente.

```
s=1
while s==1:
    suma=0
    nodo1=int(input('ingresa el 1er nodo: '))
    nodo2=int(input('ingresa el 2do nodo: '))
```

4. Dentro del ciclo while anterior se debe conocer en qué línea del metro están ambos nodos. Se busca cuál de ellas tiene una posición menor o mayor en la lista según su línea.

```
for i in range(11):
    if (nodo1 in Lineas_metro[i+1] and nodo2 in Lineas_metro[i+1]):
        linea = i+1
if Lineas_metro[linea].index(nodo1)<Lineas_metro[linea].index(nodo2):
    pos_enlinea = Lineas_metro[linea].index(nodo1)
    estacion=nodo1
    fin = nodo2
else:
    pos_enlinea = Lineas_metro[linea].index(nodo2)
    estacion=nodo2
    fin = nodo1
```

5. Se suma cada estación del metro que hay entre los nodos.

```
while estacion!=fin:
    suma += Matriz_01[Lineas_metro[linea][pos_enlinea],Lineas_metro[linea][pos_enlinea+1]]
    estacion=Lineas_metro[linea][pos_enlinea+1]
    pos_enlinea += 1
M_distancia[Nodos.index(nodo1),Nodos.index(nodo2)]=suma
s=int(input('Deseas otro nodo? Si:1 No:0 '))
```

6. Se suma la matriz final con su transpuesta. Se guardan la matriz de la distancia entre nodos.

```
M_dis=M_distancia+M_distancia.T
np.save('Distancias_nodos',M_dis)
```

Interfaz

1. Se realizó la interfaz con la librería Tkinter. No se puede modificar el tamaño.

```
raiz=Tk()
raiz.title ("Metro CDMX")
raiz.resizable(0,0)
raiz.iconbitmap("metro_logotipo.ico")
raiz.config(bg="orange")
```

2. Se hicieron tres frames: una para el título, otra para la imagen y la tercera para los botones de búsqueda y la respuesta.

```
#FRAME DEL TITULO
frameT=Frame(raiz,width="400",height="80")
frameT.pack()
frameT.config(bg="orange")


#FRAME DE LA IMAGEN
frameM=Frame(raiz,width="700",height="500")
frameM.pack(side="left")
frameM.config(bg="orange")

#FRAME DE LA BUSQUEDA
frameS=Frame(raiz,width="500",height="600")
frameS.pack(side="right", anchor="n")
frameS.config(bg="orange")
```

3. Se configura el texto de la respuesta, inicio y fin. Además que en el botón estará el algoritmo ACO.

```
#----------------- TEXTO DE RESPUESTA--------------------------
labelA=Label(frameS,text="RUTA A SEGUIR",font=("Arial Narrow",10))
labelA.place(x=50,y=200)
labelA.config(bg="orange")

texto=StringVar()
txtrespuesta=Text(frameS,width=40,height=14)#,state=DISABLED)
txtrespuesta.place(x=50,y=245)
txtrespuesta.config(bg="white")
#----------------- TEXTO DE INICIO ----------------------------
cuadroinicio=ttk.Combobox(frameS)#,state="readonly")
cuadroinicio["values"]=Lista_estaciones_alf
cuadroinicio.place(x=110,y=100)

labelinicio=Label(frameS,text="INICIO:",font=("Arial Narrow",10))
labelinicio.place(x=45,y=100)
labelinicio.config(bg="orange")

#----------------- TEXTO DE FIN ----------------------------
cuadrofin=ttk.Combobox(frameS)#,state="readonly")
cuadrofin["values"]=Lista_estaciones_alf
cuadrofin.place(x=110,y=140)

labelfin=Label(frameS,text="FIN:",font=("Arial Narrow",10))
labelfin.place(x=45,y=140)
labelfin.config(bg="orange")
```

```
#------------------ BOTON---------------------------------------
Bbuscar=Button(frameS,text="Buscar",command=boton_buscar)
Bbuscar.place(x=350,y=115)
Bbuscar.config(bg="white")


raiz.mainloop()
```

Algoritmo ACO

1. Se ingresa en orden alfabético las estaciones. Esto servirá para mostrar en la Combobox.

```
Lista_estaciones_alf=["Agricola Oriental","Allende","Aquiles Serdan","Aragon","Auditorio","Autobuses Norte","Azcapotzalco",
    "Balbuena","Balderas","Bosque Aragon","Bellas Artes","Blvd. Pto. Aereo","Bondojito","Buenavista","Camarones",
    "Canal del Norte","Canal de San Juan","Candelaria","Centro Medico","Chabacano","Chapultepec","Chilpancingo",
    "Ciudad Deportiva","Colegio Militar","Constituyentes","Consulado","Coyuya","Cuatro Caminos","Cuauhtemoc","Cuitlahuac",
    "Deportivo 18 de Marzo","Deportivo Oceania","Division del Norte","Doctores","Ecatepec","Eduardo Molina","El Rosario",
    "Etiopia","Eugenia","Ferreria","Fray Servando","Garibaldi","Gomez Farias","Guelatao","Guerrero","Hangares","Hidalgo",
    "Hospital General","Impulsora","Indios Verdes","Instituto del Petroleo","Insurgentes","Isabel la Catolica","Jamaica",
    "Juanacatlan","Juarez","Lagunilla","La Raza","La Viga","La Villa-Basilica","Lazaro Cardenas","Lindavista","Martin Carrera",
    "Merced","Misterios","Mixiuhca","M. Muzquiz","Moctezuma","Morelos","Nezahualcoyotl","Niños Heroes","Normal","Norte 45",
    "Obrera","Observatorio","Oceania","Olimpica","Panteones","Patitlan","Patriotismo","Pino Suarez","Plaza Aragon","Polanco",
    "Politecnico","Popotla","Potrero","Puebla","Refineria","Revolucion","R. Flores Magon","Rio de los Remedios","R. Rubio",
    "Salto del Agua","San Antonio","San Antonio Abad","San Cosme","San Joaquin","San Juan de Letran","San Pedro de los Pinos",
    "San Lazaro","Santa Anita","Sevilla","Tacuba","Tacubaya","Talisman","Tepalcates","Tepito","Terminal Aerea","Tezozomoc",
    "Tlatelolco","Valle Gomez","Vallejo","Velodromo","Viaducto","Villa de Aragon","Villa de Cortes","Xola","Zaragoza","Zocalo"]
```

2. El botón ingresa el algoritmo ACO. Se ingresa una lista de nodos y un diccionario con las estaciones que tiene cada línea de metro.

```
def boton_buscar():
    global M_tau
    txtrespuesta.delete('1.0', END)
    Nodos=[2,8,9,11,13,14,20,23,30,31,35,40,42,44,48,52,55,57,59,60,62,68,71,88]
    Lineas_metro={1:[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20],
                  2:[21,22,23,24,25,36,27,28,29,30,31,32,33,11,34,35,36,37,38],
                  3:[39,40,41,42,43,44,30,45,8,46,47,48,49,50,51],
                  4:[52,53,54,55,56,57,13,58,59,60],
                  5:[61,62,63,42,64,65,55,66,67,68,69,70,20],
                  6:[71,72,73,74,75,76,62,77,40,78,52],
                  7:[71,79,80,81,23,82,83,84,85,2,86,87],
                  8:[88,31,89,9,90,91,35,92,60,93],
                  9:[2,94,95,48,96,35,59,97,98,99,100,20],
                  10:[20,101,102,103,104],
                  11:[105,44,88,106,107,57,14,108,109,68,110,111,112,113,114,115,116,117,118,119]}
```

3. Se ingresa la lista de estaciones, se carga la matriz de distancia de nodos.

```
Lista_estaciones=["Observatorio","Tacubaya","Juanacatlan","Chapultepec","Se
    "Isabel la Catolica","Pino Suarez","Merced","Candelaria","San Lazaro","
    "Patitlan","Cuatro Caminos","Panteones","Tacuba","Cuitlahuac","Popotla"
    "Bellas Artes","Allende","Zocalo","San Antonio Abad","Chabacano","Viadu
    "Potrero","La Raza","Tlatelolco","Guerrero","Juarez","Niños Heroes","Hc
    "Martin Carrera","Talisman","Bondojito","Consulado","Canal del Norte","
    "Instituto del Petroleo","Autobuses Norte","Misterios","Valle Gomez","E
    "El Rosario","Tezozomoc","Azcapotzalco","Ferreria","Norte 45","Vallejo"
    "San Joaquin","Polanco","Auditorio","Constituyentes","San Pedro de los
    "La Viga","Coyuya","Patriotismo","Chilpancingo","Lazaro Cardenas","Mixi
    "Canal de San Juan","Tepalcates","Guelatao","Buenavista","Lagunilla","T
    "Villa de Aragon","Nezahualcoyotl","Impulsora","Rio de los Remedios","M

Matriz_01=np.load('Distancias_todos_completa.npy')
Matriz_nodos=np.zeros([26,26])
Matriz_nodos[0:24,0:24]=np.load('Distancias_nodos.npy')
```

4. Se obtienen los string del cuadro de inicio y cuadro del final.

```
inicio=cuadroinicio.get()
fin=cuadrofin.get()
Lista_estaciones.index(fin)+1
```

5. Si inicio y fin se encuentran a la lista de estaciones entonces entra al algoritmo ACO, si no, se manda un mensaje de error.

```
if (fin in Lista_estaciones) and (inicio in Lista_estaciones):
    num_inicio=Lista_estaciones.index(inicio)+1
    num_fin=Lista_estaciones.index(fin)+1

    misma_linea=0

    for i in range(11):
        if (num_inicio in Lineas_metro[i+1] and num_fin in Lineas_metro[i+1]):
            linea=i+1
            misma_linea=1
            break
```

6. Se busca si ambas están en la misma línea, si es así, se despliega el mensaje, sino, entra al algoritmo ACO.

```
misma_linea=0
for i in range(11):
    if (num_inicio in Lineas_metro[i+1] and num_fin in Lineas_metro[i+1]):
        linea=i+1
        misma_linea=1
        break
if misma_linea==1 and inicio!=fin:
    if linea==10:
        print("Viaja en la línea A del metro")
        txtrespuesta.insert(INSERT,"Viaja en la línea A del metro")
        txtrespuesta.insert(END,"")
    elif linea==11:
        print("Viaja en la línea B del metro")
        txtrespuesta.insert(INSERT,"Viaja en la línea B del metro")
        txtrespuesta.insert(END,"")
    else:
        print("Viaja en la línea ",linea," del metro")
        txtrespuesta.insert(INSERT,"Viaja en la línea "+str(linea)+" del metro")
        txtrespuesta.insert(END,"")
elif misma_linea==1 and inicio==fin:
```

7. Si es el mismo inicio y fin, se despliega el mensaje.

```
elif misma_linea==1 and inicio==fin:
    print("Estás donde quieres estar")
    txtrespuesta.insert(INSERT,"Estás donde quieres estar")
    txtrespuesta.insert(END,"")
```

8. La estación inicial se busca de qué línea de metro es y cuántos nodos hay en esa línea.

```
else:
    # INICIAA ##########################################
    valor2=0
    if not(num_inicio in Nodos):
        Nodos.append(num_inicio)
        for i in range(11):
            if num_inicio in Lineas_metro[i+1]:
                linea_ac=i+1
                break
        nodos_enlinea=[]
        for i in range(len(Lineas_metro[linea_ac])):
            if Lineas_metro[linea_ac][i] in Nodos:
                nodos_enlinea.append(i)
```

9. Se busca cuál es el nodo más cercano hacia atrás.

```
d_1=0
d_0=0
pos_inicio=Lineas_metro[linea_ac].index(num_inicio)
if nodos_enlinea[0]<pos_inicio:
    valor2+=1
    for j in range(pos_inicio,0,-1):
        nn=Lineas_metro[linea_ac][j]
        nn0=Lineas_metro[linea_ac][j-1]
        d_0 += Matriz_01[nn,nn0]
        if nn0 in Nodos:
            dali=Nodos.index(nn0)
            Matriz_nodos[len(Nodos)-1,dali]=d_0
            Matriz_nodos[dali,len(Nodos)-1]=d_0
            break
```

10. Se busca el nodo más cercano hacia delante de la línea del metro y se añade el valor en la matriz de nodos.

```
if nodos_enlinea[-1]>pos_inicio:
    valor2+=1
    for j in range(pos_inicio,len(Lineas_metro[linea_ac])-1):
        nn=Lineas_metro[linea_ac][j]
        nn0=Lineas_metro[linea_ac][j+1]
        d_1 += Matriz_01[nn,nn0]
        if nn0 in Nodos:
            dali=Nodos.index(nn0)
            Matriz_nodos[len(Nodos)-1,dali]=d_1
            Matriz_nodos[dali,len(Nodos)-1]=d_1
            break
```

11. Si la estación escogida no está entre nodos, es decir, sólo hay un nodo delante o atrás, se elimina esa estación de la lista de nodos y sus valores en las matrices para evitar que nuestra hormiga se encierre en un camino sin salida.

```
if valor2==1:
    ver_inicio=num_inicio
    num_inicio=nn0
    Nodos.pop()
    Matriz_nodos[len(Nodos),Nodos.index(nn0)]=0
    Matriz_nodos[Nodos.index(nn0),len(Nodos)]=0
```

12. Se realiza desde el paso 8 a 11 para la estación final.
13. Se hace inicializa la matriz tau según los valores de la matriz de nodos. Se inicializa el valor de alpha y beta, además del número de hormigas.

```
cant_n=len(Nodos)
M_dist=Matriz_nodos
M_tau=np.zeros([cant_n,cant_n]) # Matriz de tau (feromonas)

M_tau=np.where(M_dist==0,M_dist,0.1)
extra=np.where(M_dist!=0)
num_con = extra[0].shape #número de conexiones

alpha=1
beta=2
num_ants=cant_n+7


N=0
dist_op=np.sum(M_dist)
RUTA_op=[]
RUTAS_OP=[]
```

14. Se inicializan los valores para realizar las repeticiones, la actualización de feromonas y la cantidad de hormigas que buscan un camino,

```
for cambio_tau in range(5):
    rep=7
    for repeticiones in range(rep):
        RUTAS=[]
        DISTANCIAS=np.array([])
        for i in range(num_ants): # NUMERO DE HORMIGAS
            M_dist2=np.copy(M_dist)
            ruta=np.array([num_inicio])
            eleccion=Nodos.index(num_inicio)
            contador = 0
            dist = 0
```

15. Inicia un ciclo while hasta que cada hormiga encuentre una solución. Se busca en qué nodos se conectan con el nodo actual y se realiza un for para la suma de esos nodos.

```
while Nodos[eleccion]!=num_fin:
    suma=0
    Pxy=np.array([])
    sum_Pxy=np.array([])
    num_0 = Nodos.index(ruta[contador])
    conexion=np.where(M_dist2[:,num_0]!=0) #qué otros nodos se conectan al actual
    M_dist2=np.copy(M_dist)
    for i in conexion[0]:
        suma+=((1/M_dist[i,num_0])**beta)*(M_tau[i,num_0]**alpha)
    contador2=0
```

16. Se realiza el cálculo de probabilidades de cada nodo, se escoge de forma Random esa probabilidad y se añade a la ruta. Se modifica la matriz de tau para que la hormiga no “de un giro de 180 grados”.

```
contador2=0
for i in conexion[0]:
    Pxy=np.append(Pxy,((M_tau[i,num_0]**alpha)*((1/M_dist[i,num_0])**beta))/(suma))
    if contador2==0:
        sum_Pxy=np.append(sum_Pxy,Pxy[contador2])
    else:
        sum_Pxy=np.append(sum_Pxy,Pxy[contador2]+sum_Pxy[contador2-1]) #La suma de probabilidades
    contador2+=1
num_rand = float(np.random.rand(1))
P_elegida = np.min(np.where(sum_Pxy>num_rand))
eleccion = conexion[0][P_elegida]
ruta=np.append(ruta,Nodos[eleccion])
dist+=M_dist[eleccion,num_0]
M_dist2[eleccion,num_0]=0
M_dist2[num_0,eleccion]=0
contador+=1
```

17. La ruta obtenida de cada hormiga se guarda, al igual que su distancia. Se busca la ruta óptima global con **RUTA_op.**

```
        RUTAS.append(ruta)
        DISTANCIAS=np.append(DISTANCIAS,dist)
    if dist_op>np.min(DISTANCIAS):
        dist_op=np.min(DISTANCIAS)
        RUTA_op=RUTAS[np.argmin(DISTANCIAS)]
    RUTAS_OP.append(RUTAS[np.argmin(DISTANCIAS)])
```

18. Se actualizan las feromonas.

```
rho=0.5
NEW_Mtau=np.zeros([cant_n,cant_n])
for i in range(num_con[0]):
    delta_tau=0
    pp1=extra[0][i]
    pp2=extra[1][i]
    P1=Nodos[pp1]
    P2=Nodos[pp2]
    for j in range(rep): #para la SUMA
        A = np.where(RUTAS_OP[j]==P1)
        B = np.where(RUTAS_OP[j]==P2)
        aa = A[0].shape
        bb = B[0].shape
        if aa[0]>0 and bb[0]>0:
            if (A[0][0]+1==(B[0][0])) or (A[0][0]==(B[0][0]+1)):
                delta_tau+=M_dist[pp1,pp2]
    TAU = (1-rho)*M_tau[pp1,pp2]+delta_tau
    NEW_Mtau[pp1,pp2]=TAU
M_tau=NEW_Mtau
```

19. Se revisa cuál es la ruta más destacada según la matriz de tau

```
ruta_tau=[num_inicio]
seguimiento_tau=[Nodos.index(num_inicio)]
a=num_inicio
b=Nodos.index(num_inicio)
cont=0
M_tau2=np.copy(M_tau)
while a!=num_fin:
    if cont%2==0:
        pos=np.argmax(M_tau2[Nodos.index(ruta_tau[cont]),:])
        M_tau2[Nodos.index(ruta_tau[cont]),pos]=0
    else:
        pos=np.argmax(M_tau2[:,Nodos.index(ruta_tau[cont])])
        M_tau2[pos,Nodos.index(ruta_tau[cont])]=0
    a=Nodos[pos]
    b=pos
    ruta_tau.append(a)
    seguimiento_tau.append(b)
    cont+=1
    if cont>15:
        a=num_fin
```

20. Se obtiene la ruta óptima según la ruta de la matriz de tau o la ruta óptima guardada.

```
new_cont=0
impresion=[]
mov=""
if len(ruta_tau)>len(RUTA_op):
    RUTAFINAL=RUTA_op
else:
    RUTAFINAL=ruta_tau
for i in RUTAFINAL:
    if new_cont == 0:
        mov+=str(i)+" Inicia en "+str(Lista_estaciones[i-1])+"\n"
    elif new_cont>0 and new_cont<len(RUTA_op)-1:
        mov+=str(i)+" Viaja por "+str(Lista_estaciones[i-1])+"\n"
    else:
        mov+=str(i)+" Termina en "+str(Lista_estaciones[i-1])
    new_cont+=1
txtrespuesta.insert(INSERT,mov)
txtrespuesta.insert(END,"")
```

Resultados

Se muestra la interfaz gráfica en la siguiente imagen, con dos rutas escogidas.

Conclusiones

Aldaco Martínez Aldaco Iván

El moverse por el metro de la CDMX puede ser un verdadero dolor de cabeza, más cuando uno es un novato al usar este transporte público; ir en la dirección contraria, transbordos que parecen que nos llevan a caminar en círculos y la constante incertidumbre de no sabes si la ruta que tomamos es la más optima y rápida para llegar a nuestro destino, el programa que se desarrolló en esta práctica tiene el propósito de auxiliarnos con este último problema.

El programa calcula la distancia de estación a estación con el número de pixeles que las separa, esto quiere decir que si la imagen no está a escala la medición de las distancias es incorrecta. Para poder utilizar la fuerza de cómputo a nuestro favor se decidido solo tomar 24 nodos, solo en los puntos que se intersecan 2 líneas y que esa intersección no fuera una terminal. Dejando la posibilidad de que se agregaran otros 2 nodos, hasta 26 nodos, si fuera el caso de que el punto de inicio y el de final no coincidieran con los 24 anteriores.

Uno de los retos de este programa es que las rutas más sencillas, ir de estación a estación en la misma línea o ir en escuadra, las complicaba. Para el caso de ir en línea recta se le agregó la condicionante de que si las estaciones estaban en la misma línea no corriera el programa de ACO. Por otro lado, el cálculo al momento de hace una escuadra sigue dándonos resultados variables en ocasiones coincide y en otros nos da rutas en donde la distancia entre estaciones es más corta, sin embargo, no toma en cuenta el tiempo de los transbordes haciendo así que la ruta propuesta no sea las más optima.

Para poder dar una respuesta más cercana a la real, se tendríamos que revisar la correcta escala del mapa, los tiempos de espera en las estaciones con más flujo de personas y el tiempo de traslado en los transbordos. Con estos 3 factores extra el programa de ACO generaría mejores resultados y más cercanos a lo real, por ahora se desconoce cuánto mejoraría su efectividad.

Coconi Tovar Dafne Natalia

Para realizar el Ant Colony Optimization se decidió que para ahorrar tiempo de cómputo los únicos nodos fueran aquellos que tienen al menos tres uniones, es decir, cuando una línea cruz con otra. En total fueron 24 nodos y se añadieron dos filas y dos columnas extras que representarían aquellas estaciones que no están dentro de la lista de nodos, pero sí están entre dos nodos conocidos. Si no está entre dos nodos, significa que es un extremo de una línea y se tomará el inicio del nodo consecuente. Las distancias de las estaciones se midieron por los píxeles que hay entre ellos. El único cambio previo que se hizo fue que si ambas estaciones ecogidas se encontraban en la misma línea, no entrara al algoritmo ACO y sólo mostrara la estación.

El algoritmo ACO busca distintas rutas y se cambia la matriz de tau (feromonas), para escoger la ruta óptima. Se observa que este algoritmo no suele preferir las rutas perpendiculares. Por elegir repeticiones en lugar de que detener cuando los valores de tau lleguen a cierto límite, este código a veces tiene un error en elegir una ruta corta, si eso sucede, se elige la distancia más corta global guardada con anterioridad.

Vargas Téllez Axel Dali

En esta práctica se realizó la red ACO para conocer la mejor ruta posible del metro entre dos estaciones. Para realizar esto se consideraron las distancias entre estaciones de acuerdo a las medidas que se obtuvieran de la imagen de manera representativa, el sistema fue simplificado a tan solo 24 nodos iniciales y en caso de que se ingresara como punto de inicio o fin otra estación que no fuera un nodo esta era ingresada al sistema pudiendo haber hasta 26 nodos. De esta forma se requerían menor número de hormigas y con ello menor tiempo de cómputo para obtener la solución. Incluso se consideró que si uno de estos nuevos nodos solo se encontraba conectado a uno de estos 24 nodos iniciales entonces se recorría el punto de

referencia para trabajar en la red para evitar que las hormigas entraran a este camino puesto que no podían volver hacia atrás y quedaban encerradas.

También algo que se tomó en cuenta era que si la estación de inicio y la estación destino eran la misma entonces el programa no entraba a la red ACO porque resulta más óptimo seguir la misma línea sin transbordos a intentar buscar un camino virtualmente más rápido. Esto debido sobre todo a que no se tomaron en cuenta las distancias y tiempos entre transbordos, sin embargo, en la gran mayoría de los casos la red obtenía la mejor ruta posible aun cuando estos factores no eran considerados por el sistema pero si por nosotros al momento de evaluar si la ruta sería la mejor. Para poder para mejorar la precisión del sistema sería necesario tomar en cuenta las distancias reales entre estaciones y transbordos o quizá el tiempo de viaje entre ellas porque quizá dependiendo de la línea (si es más recurrida o menos) la velocidad de los trenes varié.

Referencias

[1] M. Pedemonte, «Ant Colony Optimization,» *Universidad de República de Montevideo,* 2007.