

Nombre: Salvador Gimeno

Actividad Guiada 3 - AG3

Google Drive: <https://colab.research.google.com/drive/1fhcXrSxm6PMEXQpFgo0K2VXqP9IO-1tl?usp=sharing>

Github: <https://github.com/salvagimeno-ai/03MAIR-Algoritmos-de-optimizacion/tree/master/AG3>

▼ 1) Problema del Viajero (TSP)

```
!pip install request # libreria para hacer llamadas http (descargar págonas y ficheros)
```

```
↳ Requirement already satisfied: request in /usr/local/lib/python3.6/dist-packages (0.0.0)
Requirement already satisfied: post in /usr/local/lib/python3.6/dist-packages (from request) (0.0.0)
Requirement already satisfied: get in /usr/local/lib/python3.6/dist-packages (from request) (0.0.0)
Requirement already satisfied: query-string in /usr/local/lib/python3.6/dist-packages (from get->request) (0.0.0)
```

```
!pip install tsplib95
```

```
↳ Requirement already satisfied: tsplib95 in /usr/local/lib/python3.6/dist-packages (0.7.1)
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.6/dist-packages (from tsplib95) (7.1.2)
Requirement already satisfied: Deprecated~=1.2.9 in /usr/local/lib/python3.6/dist-packages (from tsplib95) (1.2.10)
Requirement already satisfied: networkx~=2.1 in /usr/local/lib/python3.6/dist-packages (from tsplib95) (2.4)
Requirement already satisfied: tabulate~=0.8.7 in /usr/local/lib/python3.6/dist-packages (from tsplib95) (0.8.7)
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.6/dist-packages (from Deprecated~=1.2.9->tsplib95) (1.12.1)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.6/dist-packages (from networkx~=2.1->tsplib95) (4.4.2)
```

```
import urllib.request
file='swiss42.tsp'
urllib.request.urlretrieve('http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/swiss42.tsp', file)
```

```
↳ ('swiss42.tsp', <http.client.HTTPMessage at 0x7efdb11559b0>)
```

Saved successfully!



```
import tsplib95
```

```
#definimos la variable problema
problem = tsplib95.load_problem(file)
# Obtenemos los nodos
Nodos = list(problem.get_nodes())
# Obtenemos las aristas
Aristas = list(problem.get_edges())
```

```
↳
```

Nodos

```
↳
```

Saved successfully!



Aristas



Saved successfully!



...os devuelve la distancia entre dos puntos

Saved successfully!



▼ Metaheurísticas de búsquedas: Búsqueda Aleatoria

```
# Se genera una solución aleatoria con comienzo 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

# Calcula la distancia entre nodos
def distancia(a,b, problem):
    return problem.get_weight(a,b)

# Devuelve la distancia total de un recorrido
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 )

solucion_prueba = crear_solucion(Nodos)
print(solucion_prueba)

print(distancia_total(solucion_prueba, problem), solucion_prueba )
```



Saved successfully!



la distancia a partir de una Busqueda Aleatoria
: #N = numero de iteraciones

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

```
mejor_solucion = []
```

```
mejor_distancia = 10e10    #colocamos un valor muy alto
```

```
for i in range(N):
```

```
    solucion = crear_solucion(Nodos)
```

```
    distancia_solucion = distancia_total(solucion, problem)
```

```
    if distancia_solucion < mejor_distancia:
```

```
        mejor_solucion = solucion
```

```
        mejor_distancia = distancia_solucion
```

```
print(mejor_distancia, mejor_solucion)
```

```
return mejor_distancia, mejor_solucion
```

```
busqueda_aleatoria(problem, 100)
```



Saved successfully!



▼ Metaheurísticas de búsquedas: Búsqueda Local

```
# Búsqueda Local, Función para generar los vecinos
def generar_vecina(solucion, problem):
    mejor_solucion = []
    mejor_distancia = 10e10
    for i in range(1, len(solucion)-1): #recorremos desde el 2o nodo hasta el penultimo
        for j in range(i+1, len(solucion)): #recorremos desde el siguiente nodo hasta el final
            vecina = solucion[i+1] + [solucion[i+1] + solucion[i+1+1] + [solucion[i+1] + solucion[i+1+1]
```

```

vecina = solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + solucion[j+1:]
distancia_vecina = distancia_total(vecina, problem)
if distancia_vecina < distancia:
    distancia = distancia_vecina
return mejor_solucion

```

Saved successfully!



```

solucion_prueba = crear_solucion(Nodos)
solucion_vecina = generar_vecina(solucion_prueba, problem)

print(distancia_total(solucion_prueba,problem),solucion_prueba)
print(distancia_total(solucion_vecina,problem),solucion_vecina)

```



```

## Busqueda Local, Algoritmo (2019)
def busqueda_local(problem,N):
    mejor_solucion = []
    mejor_distancia = 10e10

    Nodos = list(problem.get_nodes())

    solucion_referencia = crear_solucion(Nodos)

    for i in range(N):
        vecina = generar_vecina(solucion_referencia, problem)
        distancia_vecina = distancia_total(vecina, problem)
        if distancia_vecina < mejor_distancia:
            mejor_solucion = vecina
            mejor_distancia = distancia_vecina

    solucion_referencia = mejor_solucion

    print(distancia_total(mejor_solucion, problem), mejor_solucion)

    return mejor_solucion

```



```
busqueda_local(problem,100)
```

Saved successfully!

```
def busqueda_local(problem):
    solucion_referencia = crear_solucion(Nodos)
    mejor_distancia = 10e100

    iteracion = 0
    while(1):
        iteracion+=1
        vecina = generar_vecina(solucion_referencia,problem)
        distancia_vecina = distancia_total(vecina,problem)
        if distancia_vecina < mejor_distancia:
            mejor_solucion = vecina
            mejor_distancia = distancia_vecina
        else:
            print('En la iteracion ' + str(iteracion)+
                  ' encontramos la solucion:'+str(mejor_distancia))
            print("Distancia :",mejor_distancia)
            print(mejor_solucion)
            return mejor_solucion
        solucion_referencia = vecina
sol = busqueda_local(problem)
```



▼ Metaheurísticas de búsquedas: Recocido Simulado

```
#METAHEURISTICAS: Recocido Simulado
def probabilidad(T,d): # T = temperatura, d = distancia
    r=random.random() #generamos un numero aleatorio
```

```
if r >= (e**(-1*d)/((T * .5*10**(-5)))):
```

Saved successfully!



```
return False
```

```
def bajar_temperatura(T):
```

```
    return T*.9
```

```
    #return T-1
```

```
def recocido_simulado(problema, TEMPERATURA):
```

```
    solucion_referencia = crear_solucion(Nodos)
```

```
    distancia_referencia = distancia_total(solucion_referencia, problem)
```

```
    mejor_solucion = []
```

```
    mejor_distancia = 10e10
```

```
    while TEMPERATURA > 1:
```

```
        vecina = generar_vecina(solucion_referencia, problem)
```

```
        distancia_vecina = distancia_total(vecina, problem)
```

```
        if distancia_vecina < mejor_distancia:
```

```
            mejor_solucion = vecina
```

```
            mejor_distancia = distancia_vecina
```

```
        if distancia_vecina < distancia_referencia or probabilidad(TEMPERATURA, abs(distancia_referencia - distancia_vecina)):
```

```
            solucion_referencia = vecina
```

```
            distancia_referencia = distancia_vecina
```

```
    TEMPERATURA = bajar_temperatura(TEMPERATURA)
```

```
    print(mejor_distancia, mejor_solucion)
```

```
    return mejor_solucion
```

```
recocido_simulado(problem,100)
```

Saved successfully!



Saved successfully!



▼ Metaheurísticas de búsquedas: Método Constructivo. Colonia de hormigas.

```
# Algoritmo COLONIA DE HORMIGAS:
```

```
# 1) Funciones Auxiliares:
```

```
def Add_Nodo(problem, H ,T ) :  
    #Mejora:Establecer una funcion de probabilidad para  
    #añadir un nuevo nodo dependiendo de los nodos mas cercanos y  
    #de las feromonas depositadas  
    Nodos = list(problem.get_nodes())  
    return random.choice( list(set(range(1,len(Nodos))) - set(H) ) )
```

```
def Incrementa_Feromona(problem, T, H ) :  
    #Incrementa segun la calidad de la solución.  
    #Añadir una cantidad inversamente proporcional a la distancia total  
    for i in range(len(H)-1):  
        T[H[i]][H[i+1]] += 1000/distancia_total(H, problem)  
    return T
```

```
def Evaporar_Feromonas(T ):  
    #Evapora 0.3 el valor de la feromona, sin que baje de 1  
    #Mejora:Podemos elegir diferentes funciones de evaporación  
    #dependiendo de la cantidad actual y de la suma total de feromonas depositadas,...  
    T = [[ max(T[i][j] - 0.3 , 1) for i in range(len(Nodos)) ]  
          for j in range(len(Nodos))]  
    return T
```

```
# 2) Método Constructivo. Colonia de Hormigas
```

```
def hormigas(problem, N) :
```

Saved successfully!



)

```
#Nodos
```

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

```
#Aristas
```

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

```
#Inicializa las aristas con una cantidad inicial de feromonas:1
```

```
#Mejora: inicializar con valores diferentes dependiendo diferentes criterios
```

```
T = [[ 1 for _ in range(len(Nodos)) ] for _ in range(len(Nodos))]
```

```
#Se generan los agentes(hormigas) que serán estructuras de caminos desde 0
```

```
Hormiga = [[0] for _ in range(N)]      # se inicializa todo con ceros
```

```
#Recorre cada agente construyendo la solución
```

```
for h in range(N) :      #N = numero de hormigas
```

```
    #Para cada agente se construye un camino
```

```
    for i in range(len(Nodos)-1) :
```

```
        #Elige el siguiente nodo
```

```
        Nuevo_Nodo = Add_Nodo(problem, Hormiga[h] ,T )      # Añade un nuevo nodo
```

```
        Hormiga[h].append(Nuevo_Nodo)
```

```
#Incrementa feromonas en esa arista
```

```
T = Incrementa_Feromona(problem, T, Hormiga[h] )      # T = lista de aristas
```

```
#print("Feromonas(1)", T)
```

```
#Evapora Feromonas
```

```
T = Evaporar_Feromonas(T)
```

```
#print("Feromonas(2)", T)
```

```
#Seleccionamos el mejor agente
```

```
mejor_solucion = []
```

```
mejor_distancia = 10e100
```

```
for h in range(N) :
```

```
distancia_actual = distancia_total(Hormiga[h], problem)
if distancia_actual < mejor_distancia:
```

Saved successfully!



actual

```
print(mejor_solucion)
print(mejor_distancia)
```

```
hormigas(problem, 100)
```



```
#-----
# Fin AG3 (Salvador Gimeno)
#-----
```

Saved successfully!

