

## AG3 - Actividad Guiada 3 (2)

Nombre: Víctor Callejas Fuentes

Link: <https://colab.research.google.com/drive/1wM5YCKzs68hMPWpSY91rRpp30O4xhvX9?usp=sharing>

Github: [https://github.com/VictorCallejas/O3miar/blob/main/Algoritmos\\_AG3\(2\)\\_VictorCallejasFuentes.ipynb](https://github.com/VictorCallejas/O3miar/blob/main/Algoritmos_AG3(2)_VictorCallejasFuentes.ipynb)

### ✓ Carga de librerías

Double-click (or enter) to edit

```
!pip install requests
!pip install tabulate>=0.9 networkx>=3.0
!pip install tsplib95 --no-deps
!pip install deprecated
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (2.32.4)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests) (2026.1.4)
Collecting tsplib95
  Downloading tsplib95-0.7.1-py2.py3-none-any.whl.metadata (6.3 kB)
Downloading tsplib95-0.7.1-py2.py3-none-any.whl (25 kB)
Installing collected packages: tsplib95
Successfully installed tsplib95-0.7.1
Collecting deprecated
  Downloading deprecated-1.3.1-py2.py3-none-any.whl.metadata (5.9 kB)
Requirement already satisfied: wrapt<3,>=1.10 in /usr/local/lib/python3.12/dist-packages (from deprecated) (2.0.1)
Downloading deprecated-1.3.1-py2.py3-none-any.whl (11 kB)
Installing collected packages: deprecated
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the
tsplib95 0.7.1 requires Deprecated~=1.2.9, but you have deprecated 1.3.1 which is incompatible.
tsplib95 0.7.1 requires networkx~=2.1, but you have networkx 3.6.1 which is incompatible.
tsplib95 0.7.1 requires tabulate~=0.8.7, but you have tabulate 0.9.0 which is incompatible.
Successfully installed deprecated-1.3.1
```

```
!pip install deprecated
```

```
Requirement already satisfied: deprecated in /usr/local/lib/python3.12/dist-packages (1.3.1)
Requirement already satisfied: wrapt<3,>=1.10 in /usr/local/lib/python3.12/dist-packages (from deprecated) (2.0.1)
```

### ✓ Carga de los 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

#Descargamos el fichero de datos(Matriz de distancias)
file = "swiss42.tsp" ;
#urllib.request.urlretrieve("https://raw.githubusercontent.com/mastqe/tsplib/refs/heads/master/swiss42.tsp", file + '.gz')
urllib.request.urlretrieve("https://raw.githubusercontent.com/mastqe/tsplib/refs/heads/master/swiss42.tsp", file )
#!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

#http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95
#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/
```

```
('swiss42.tsp', <http.client.HTTPMessage at 0x7842c4dc5580>)
```

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

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

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

Nodos

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

Aristas

```
# https://github.com/mastqe/tsplib/blob/master/swiss42.tsp
```

(19, 29)

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

```
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 1
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
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
```

```
#Probamos algunas funciones del objeto problem
```

```
#Distancia entre nodos
problem.get_weight(3, 0)
```

```
#Todas las funciones
#Documentación: https://tsplib95.readthedocs.io/en/v0.6.1/modules.html
```

```
#dir(problem)
```

```
23
```

## Funcionas basicas

```
#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)

sol_temporal = crear_solucion(Nodos)

print(sol_temporal )
print(distancia_total(sol_temporal, problem))

[0, 4, 39, 3, 17, 36, 7, 34, 14, 19, 20, 28, 15, 18, 41, 25, 26, 37, 12, 8, 1, 27, 11, 38, 31, 32, 24, 21, 13, 9, 22, 10, 40, 30, 4411
```

## BUSQUEDA ALEATORIA

```
#####
# BUSQUEDA ALEATORIA
#####

def busqueda_aleatoria(problem, N):
    #N es el numero de iteraciones
    Nodos = list(problem.get_nodos())

    mejor_solucion = []
    #mejor_distancia = 10e100                                #Inicializamos con un valor alto
    mejor_distancia = float('inf')                          #Inicializamos con un valor alto

    for i in range(N):
        solucion = crear_solucion(Nodos)                    #Criterio de parada: repetir N veces pero podemos incluir otros
        distancia = distancia_total(solucion, problem)       #Genera una solucion aleatoria
                                                              #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, 5000)

Mejor solución: [0, 13, 19, 5, 37, 17, 36, 33, 35, 6, 27, 14, 15, 29, 9, 22, 24, 39, 21, 3, 34, 40, 8, 10, 16, 30, 23, 12, 26, 2, 4411
Distancia      : 3475
```

## BUSQUEDA LOCAL

```
#####
# BUSQUEDA LOCAL(1 paso)
#####
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-
        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,
print("Distancia Solucion Inicial:" , distancia_total(solucion, problem))

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

Distancia Solucion Inicial: 3475  
 Distancia Mejor Solucion Local: 3230

```
#Busqueda Local(iteraciones):
# - Sobre el operador de vecindad 2-opt(funcion genera_vecina)
# - Sin criterio de parada, se para cuando no es posible mejorar.
def busqueda_local(solucion, problem):
    mejor_solucion = []

    #Generar una solucion inicial de referencia(aleatoria)
    #solucion_referencia = crear_solucion(Nodos)
    solucion_referencia = solucion
    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 vecindad 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(nueva_solucion, problem )
```

En la iteracion 26 , la mejor solución encontrada es: [0, 13, 19, 16, 37, 17, 36, 35, 33, 20, 15, 14, 5, 9, 23, 40, 24, 21, 39,  
 Distancia : 1890

## SIMULATED ANNEALING

```
#####
# 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
```

```
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 pseudocódigo
    mejor_distancia = 10e100     #F* del pseudocódigo

    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, 1000000)
```

La mejor solución encontrada es [0, 16, 15, 37, 31, 5, 26, 8, 39, 38, 33, 20, 34, 22, 35, 36, 17, 32, 27, 28, 29, 9, 23, 21, 24, con una distancia total de 2312

## ✓ Mejoras Opcionales

A continuación se implementan dos mejoras sobre los algoritmos base:

1. Búsqueda Local con Entornos Variables (VNS)
2. Mejora del Recocido Simulado con generador de vecinos mejorado

## ✓ Búsqueda Local con Entornos Variables (VNS)

La idea es combinar dos operadores de vecindad (2-opt y 3-opt). Cuando la búsqueda se estanca en un mínimo local con un operador, cambiamos al otro operador para intentar escapar y seguir mejorando.

```
# Generador de vecinos con operador 3-opt
# Intercambia 3 nodos en lugar de 2, explorando mas combinaciones

def genera_vecina_3opt(solucion):
    mejor_solucion = solucion
    mejor_distancia = distancia_total(solucion, problem)

    n = len(solucion)
    for i in range(1, n-2):
        for j in range(i+1, n-1):
            for k in range(j+1, n):
                # Generamos varias permutaciones intercambiando los 3 nodos
                vecinos = []
                vecinos.append(solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + solucion[j+1:])
                vecinos.append(solucion[:j] + [solucion[k]] + solucion[j+1:k] + [solucion[j]] + solucion[k+1:])
                vecinos.append(solucion[:i] + [solucion[k]] + solucion[i+1:k] + [solucion[i]] + solucion[k+1:])

                for v in vecinos:
                    d = distancia_total(v, problem)
                    if d < mejor_distancia:
                        mejor_distancia = d
                        mejor_solucion = v

    return mejor_solucion

# Probamos el operador 3-opt
print("Distancia solucion inicial:", distancia_total(solucion, problem))
nueva_sol_3opt = genera_vecina_3opt(solucion)
print("Distancia mejor vecina 3-opt:", distancia_total(nueva_sol_3opt, problem))
```

Distancia solucion inicial: 3475  
Distancia mejor vecina 3-opt: 3230

```
# Busqueda Local con Entornos Variables
# Alterna entre 2-opt y 3-opt para escapar de minimos locales

def busqueda_local_vns(problem):
    Nodos = list(problem.get_nodes())
    solucion_referencia = crear_solucion(Nodos)
    mejor_distancia = distancia_total(solucion_referencia, problem)
    mejor_solucion = solucion_referencia

    entorno = 1 # 1 = 2-opt, 2 = 3-opt
    iteracion = 0
    sin_mejora = 0

    while sin_mejora < 2: # Probamos ambos entornos antes de parar
        iteracion += 1

        # Elegimos el operador segun el entorno actual
        if entorno == 1:
            vecina = genera_vecina(solucion_referencia)
        else:
            vecina = genera_vecina_3opt(solucion_referencia)
```

```

distancia_vecina = distancia_total(vecina, problem)

if distancia_vecina < mejor_distancia:
    mejor_solucion = vecina
    mejor_distancia = distancia_vecina
    solucion_referencia = vecina
    sin_mejora = 0
    entorno = 1 # Volvemos al primer entorno
else:
    sin_mejora += 1
    # Cambiamos de entorno
    entorno = 2 if entorno == 1 else 1

print("Iteraciones:", iteracion)
print("Mejor solucion (VNS):", mejor_solucion)
print("Distancia:", mejor_distancia)
return mejor_solucion

sol_vns = busqueda_local_vns(problem)

Iteraciones: 41
Mejor solucion (VNS): [0, 5, 13, 26, 6, 4, 18, 12, 11, 29, 30, 38, 22, 39, 24, 40, 21, 9, 23, 41, 25, 10, 8, 1, 7, 17, 36, 35, 3
Distancia: 1636

```

## ✓ Mejora del Recocido Simulado

Mejoramos la función `genera_vecina_aleatorio()` usando un enfoque semi-greedy: en lugar de generar un solo vecino aleatorio 2-opt, generamos varios candidatos con 3-opt y elegimos el mejor de ellos. Esto mantiene cierta aleatoriedad pero con una exploración mas inteligente.

```

# Generador de vecinos mejorado para Simulated Annealing
# Genera varios candidatos 3-opt aleatorios y elige el mejor

def genera_vecina_aleatorio_mejorado(solucion):
    mejor_solucion = solucion
    mejor_distancia = distancia_total(solucion, problem)

    # Generamos 3 candidatos aleatorios usando 3-opt
    for _ in range(3):
        # Elegimos 3 posiciones aleatorias
        indices = sorted(random.sample(range(1, len(solucion)), 3))
        i, j, k = indices

        # Generamos varias permutaciones con esos 3 nodos
        vecinos = []
        vecinos.append(solucion[:i] + [solucion[j]] + solucion[i+1:j] + [solucion[i]] + solucion[j+1:])
        vecinos.append(solucion[:j] + [solucion[k]] + solucion[j+1:k] + [solucion[j]] + solucion[k+1:])
        vecinos.append(solucion[:i] + [solucion[k]] + solucion[i+1:k] + [solucion[i]] + solucion[k+1:])

        for v in vecinos:
            d = distancia_total(v, problem)
            if d < mejor_distancia:
                mejor_distancia = d
                mejor_solucion = v

    return mejor_solucion

```

```

# Recocido Simulado con generador mejorado

def recocido_simulado_mejorado(problem, TEMPERATURA):
    Nodos = list(problem.get_nodes())
    solucion_referencia = crear_solucion(Nodos)
    distancia_referencia = distancia_total(solucion_referencia, problem)

    mejor_solucion = solucion_referencia
    mejor_distancia = distancia_referencia

    iteraciones = 0
    while TEMPERATURA > 0.0001:
        iteraciones += 1

        # Usamos el generador mejorado
        vecina = genera_vecina_aleatorio_mejorado(solucion_referencia)

```



```

distancia_vecina = distancia_total(vecina, problem)

# Si es la mejor de todas, la guardamos
if distancia_vecina < mejor_distancia:
    mejor_solucion = vecina
    mejor_distancia = distancia_vecina

# Aceptamos si mejora o con cierta probabilidad si empeora
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("Iteraciones:", iteraciones)
print("Mejor solucion (SA mejorado):", mejor_solucion)
print("Distancia:", mejor_distancia)
return mejor_solucion

sol_sa_mejorado = recocido_simulado_mejorado(problem, 10000000)

Iteraciones: 2521
Mejor solucion (SA mejorado): [0, 3, 1, 7, 20, 33, 35, 36, 31, 17, 37, 15, 16, 14, 12, 11, 25, 41, 23, 40, 24, 21, 9, 39, 22, 38]
Distancia: 1624

```

## ▼ Comparativa de resultados

Comparamos las distancias obtenidas por cada metodo para evaluar las mejoras:

```

# Comparativa de distancias
print("=" * 50)
print("COMPARATIVA DE RESULTADOS")
print("=" * 50)
print(f"Busqueda Aleatoria:      {distancia_total(solucion, problem)}")
print(f"Busqueda Local 2-opt:    {distancia_total(sol, problem)}")
print(f"Recocido Simulado basico:  {distancia_total(sol, problem)}")
print(f"Busqueda Local VNS:       {distancia_total(sol_vns, problem)}")
print(f"Recocido Simulado mejorado:{distancia_total(sol_sa_mejorado, problem)}")
print("=" * 50)

```

```

=====
COMPARATIVA DE RESULTADOS
=====
Busqueda Aleatoria:      3475
Busqueda Local 2-opt:    2312
Recocido Simulado basico: 2312
Busqueda Local VNS:     1636
Recocido Simulado mejorado:1624
=====

```

## ▼ Representación en un grafo a partir de la matriz de distancias( Optimización de posiciones usando escalado multidimensional (MDS)

Multidimensional scaling problem(MDS): [https://en.wikipedia.org/wiki/Multidimensional\\_scaling](https://en.wikipedia.org/wiki/Multidimensional_scaling)

```

import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from sklearn.manifold import MDS # Multidimensional Scaling o Escalado Multidimensional

def plot_tsp_solution(distance_matrix, tsp_solution):
    """
    Dibuja el grafo de un TSP con las posiciones calculadas mediante MDS y muestra
    solo las aristas correspondientes a la solución del TSP.

    :param distance_matrix: np.ndarray, matriz de distancias entre nodos
    :param tsp_solution: list, lista de nodos en el orden de la solución del TSP
    """
    # Crear el grafo completo
    G = nx.Graph()

```

```

num_nodes = len(distance_matrix)
for i in range(num_nodes):
    for j in range(i + 1, num_nodes):
        G.add_edge(i, j, weight=distance_matrix[i][j])

# Usar MDS para calcular posiciones de los nodos
mds = MDS(n_components=2, dissimilarity="precomputed", random_state=42)
positions = mds.fit_transform(distance_matrix)

# Convertir las posiciones en un diccionario para networkx
pos = {i: positions[i] for i in range(num_nodes)}

# Crear un subgrafo con las aristas del camino TSP
TSP_G = nx.Graph()
for i in range(len(tsp_solution) - 1):
    u = tsp_solution[i]
    v = tsp_solution[i + 1]
    TSP_G.add_edge(u, v, weight=distance_matrix[u][v])

# Dibujar el grafo
plt.figure(figsize=(8, 6))

# Dibujar nodos
nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=500)

# Dibujar las aristas del camino TSP
nx.draw_networkx_edges(TSP_G, pos, edge_color='red', width=2)

# Añadir etiquetas a los nodos y pesos de las aristas
nx.draw_networkx_labels(G, pos, font_size=10, font_weight='bold')
edge_labels = nx.get_edge_attributes(TSP_G, 'weight')
nx.draw_networkx_edge_labels(TSP_G, pos, edge_labels=edge_labels, font_size=8)

plt.title("Grafo TSP con solución específica")
plt.show()

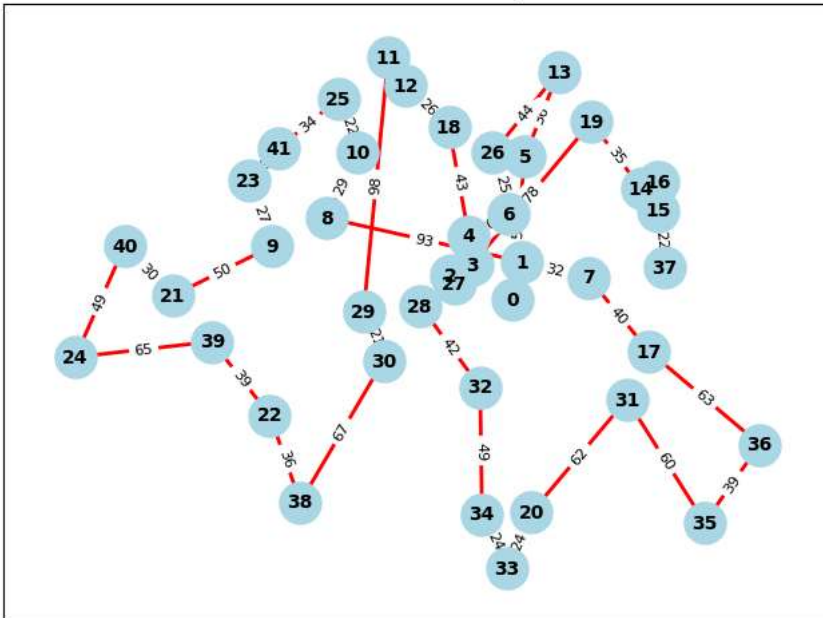
```

```

# Visualizacion de la solucion VNS
plot_tsp_solution(problem.edge_weights, sol_vns)

```

Grafo TSP con solución específica

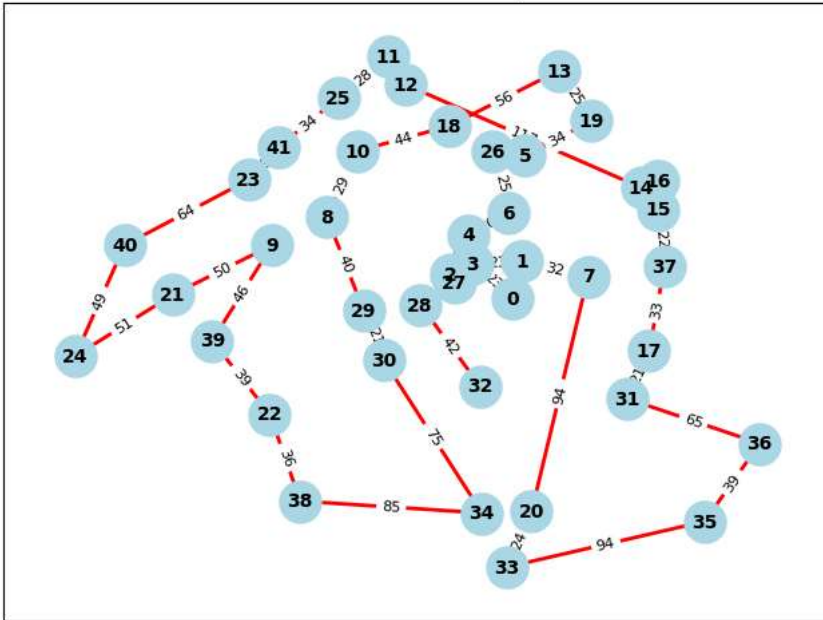


```

# Visualizacion de la solucion SA mejorado
plot_tsp_solution(problem.edge_weights, sol_sa_mejorado)

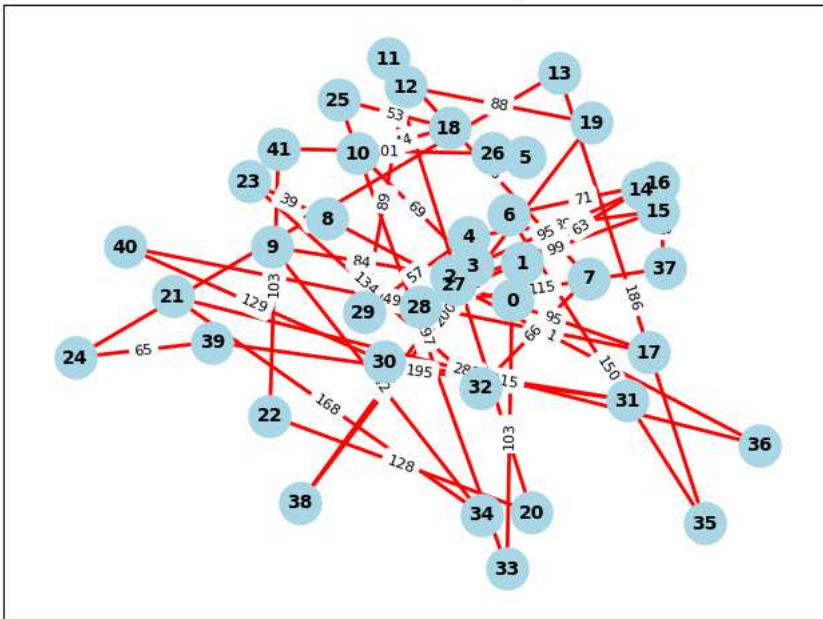
```

Grafo TSP con solución específica



```
plot_tsp_solution(problem.edge_weights, crear_solucion(Nodos))
```

Grafo TSP con solución específica



```
plot_tsp_solution(problem.edge_weights, solucion)
```

Grafo TSP con solución específica

