

# AG3 – Actividad Guiada 3

## 03MIAR – Algoritmos de optimización

Abrir el cuaderno de google colab:

[https://colab.research.google.com/drive/1ynPeKMTJDJPQ9ctRwCIn\\_LA9Kk1ycXS9](https://colab.research.google.com/drive/1ynPeKMTJDJPQ9ctRwCIn_LA9Kk1ycXS9)

# Errores en las fechas de Examen y Entrega de Actividades

Enero 2024

LUN	MAR	MIÉ	JUE	VIE	SÁB	DOM
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				

Febrero 2024

LUN	MAR	MIÉ	JUE	VIE	SÁB	DOM
			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*			

Marzo 2024

LUN	MAR	MIÉ	JUE	VIE	SÁB	DOM
				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

● Tutoría colectiva a cargo del Director/a de título

● Videoconferencia de primeros pasos (orientador)

● Python para la inteligencia artificial

● Algoritmos de Optimización

● Aprendizaje Supervisado

● Razonamiento aproximado

● Redes Neuronales y Deep Learning

● Aprendizaje por refuerzo

● Festivos

# Rúbrica del trabajo práctico

Niveles de rendimiento ↕			
Criterios ↕	Principiante ↕	Competente ↕	Muy competente ↕
<b>Modelo</b> ↕ Ponderación 30,00 %	Porcentaje 4,00 <ul style="list-style-type: none"><li>- Representación básica y poco clara del espacio de soluciones.</li><li>- Función objetivo no claramente definida.</li></ul> abc ✓	Porcentaje 70,00 <ul style="list-style-type: none"><li>- Representación adecuada del espacio de soluciones.</li><li>- Función objetivo definida con claridad.</li><li>- Implementación correcta de</li></ul> abc ✓	Porcentaje 100,00 <ul style="list-style-type: none"><li>- Representación detallada y eficiente del espacio de soluciones.</li><li>- Función objetivo definida con precisión y detalle.</li><li>- Implementación completa y correcta de todas las restricciones.</li></ul> abc ✓
<b>Análisis</b> ↕ Ponderación 30,00 %	Porcentaje 4,00 <ul style="list-style-type: none"><li>- Identificación limitada de la complejidad.</li><li>- Contabilización del espacio de soluciones poco clara o incorrecta.</li></ul> abc ✓	Porcentaje 70,00 <ul style="list-style-type: none"><li>- Análisis correcto de la complejidad del problema.</li><li>- Buena contabilización del espacio de soluciones.</li></ul> abc ✓	Porcentaje 100,00 <ul style="list-style-type: none"><li>- Análisis detallado y preciso de la complejidad del problema.</li><li>- Contabilización exhaustiva y precisa del espacio de soluciones.</li></ul> abc ✓
<b>Diseño</b> ↕ Ponderación 30,00 %	Porcentaje 4,00 <ul style="list-style-type: none"><li>- Elección de técnica poco adecuada o mal justificada.</li></ul> abc ✓	Porcentaje 70,00 <ul style="list-style-type: none"><li>- Buena elección de técnica con justificación adecuada.</li></ul> abc ✓	Porcentaje 100,00 <ul style="list-style-type: none"><li>- Elección excelente de técnica con justificación detallada y bien fundamentada.</li></ul> abc ✓
<b>Presentación y Redacción</b> ↕ Ponderación 10,00 %	Porcentaje 4,00 <ul style="list-style-type: none"><li>- Presentación básica y redacción con errores significativos.</li><li>- Falta de claridad en la exposición.</li></ul> abc ✓	Porcentaje 70,00 <ul style="list-style-type: none"><li>- Presentación clara y redacción con pocos errores.</li><li>- Exposición ordenada y comprensible.</li></ul> abc ✓	Porcentaje 100,00 <ul style="list-style-type: none"><li>- Presentación y redacción excelentes.</li><li>- Exposición muy clara, estructurada y atractiva.</li></ul> abc ✓

## Actividad. Borrador del trabajo práctico(no evaluable)



### **Borrador Trabajo práctico(no evaluable)**

- Desarrollar, modelar y analizar algoritmos según diferentes técnicas para resolver el problema planteado en la asignatura.
- Utiliza la siguiente plantilla de Google Colab como base("Guardar una copia en Drive")

Fecha Límite : lunes, 12 de febrero

## Agenda

1. Librería TSPLIB para el agente viajero - TSP
2. Resolución por búsqueda aleatoria
3. Resolución por Búsqueda local
4. Resolución por recocido simulado(Simulated Annealing -SA)
5. Planteamiento por Colonia de Hormigas – ACO (no evaluable)

# AG3 – Actividad Guiada 3

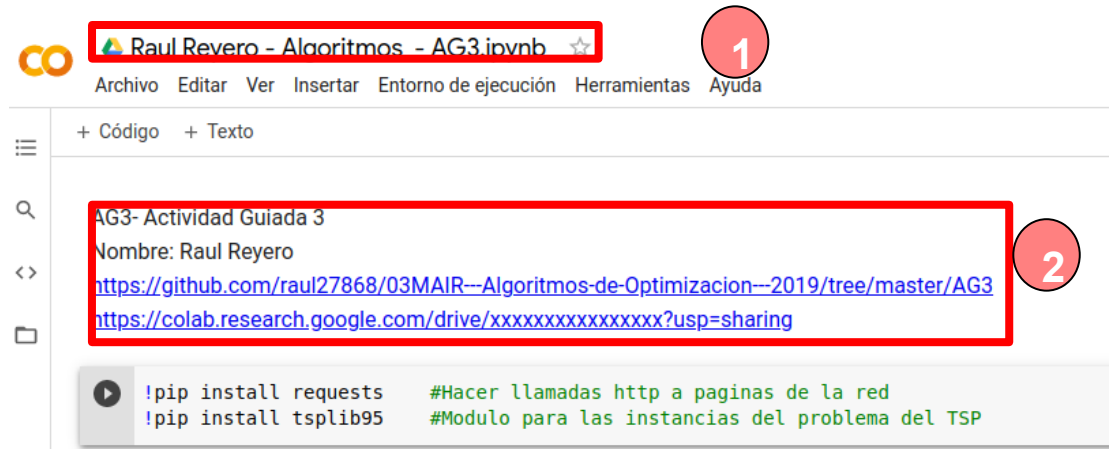
## Librería TSPLIB

### Agenda

1. Librería TSPLIB para el agente viajero - TSP
2. Resolución por búsqueda aleatoria
3. Resolución por Búsqueda local
4. Resolución por recocido simulado
5. Planteamiento por Colonia de Hormigas - ACO

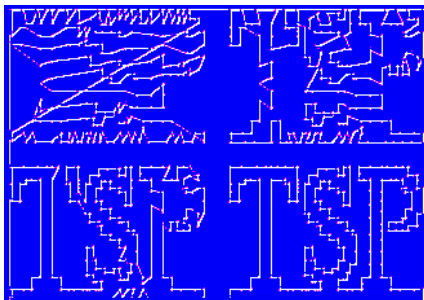
## Preparar la actividad en Google Colaboratory.

- Copiar con notebook en Google Colaboratory  
[https://colab.research.google.com/drive/1ynPeKMTJDJPQ9ctRwCln\\_LA9Kk1ycXS9?usp=sharing](https://colab.research.google.com/drive/1ynPeKMTJDJPQ9ctRwCln_LA9Kk1ycXS9?usp=sharing)
- Renombra el documento python : **<nombre apellido>-AG3**
- Crear un texto con:
  - \* AG3- Actividad Guiada 3
  - \* Nombre Apellidos
  - \* Url a la carpeta **AG3** de GitHub



## El problema del agente viajero – TSP. TSPLIB

Juegos de datos para poner a prueba nuestros diseños para resolver el problema del TSP



<http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/>

### Symmetric traveling salesman problem (TSP)

Given a set of  $n$  nodes and distances for each pair of nodes, find a roundtrip of minimal total length visiting each node exactly once. The distance from node  $i$  to node  $j$  is the same as from node  $j$  to node  $i$ .

-> TSP data

Best known solutions for symmetric TSPs





## El problema del agente viajero - TSP

- Es el problema más estudiado.
- Sirve de test para los diseños de nuevos algoritmos o técnicas.
- Para simplificar, suponemos todos los nodos conectados y comenzamos y terminamos por el nodo 0.



## Preparación de los datos

Instalación del módulo `tsplib` y otras librerías.

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

```
import tsplib95      #Modulo para las instancias del problema del TSP
import random        #Modulo para generar números aleatorios
from math import e    #constante e
import copy          #Para copia profunda de estructuras de datos(en python la asignación es por referencia)
|
```



## Preparación de los datos

### Cargar una instancia del problema: swiss42.tsp

```
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

#http://elib.zib.de/pub/mp-testdata/tsp/tsplib/
#Documentacion :
# http://comopt.if1.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)
file = "swiss42.tsp" ;
urllib.request.urlretrieve("http://comopt.if1.uni-heidelberg.de/software/TSPLIB95/tsp/swiss42.tsp.gz", file + '.gz')
!gzip -d swiss42.tsp.gz      #Descomprimir el fichero de datos

#Coordenadas 51-city problem (Christofides/Eilon)
#file = "eil51.tsp" ; urllib.request.urlretrieve("http://comopt.if1.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.if1.uni-heidelberg.de/software/TSPLIB95/tsp/att48.tsp.gz", file)
```



## Preparación de los datos

### Cargar datos del problema

```
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 6
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 9
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
```

```
problem = tsplib95.load_problem(file)
```

```
#Nodos
```

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

```
#Aristas
```

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



## Preparación de los datos

Probamos algunas funciones del problema

```
#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)
```



## Preparación de los datos

### Datos del problema

Para  $n=42$  ciudades(nodos) el total de soluciones es:

$$(n-1)!/2 = 33452526613163807108170062053440751665152000000000/2$$

La distancia para la mejor solución encontrada al problema swiss42.tsp es:

1273

Symmetric traveling salesman problem (TSP)

Given a set of  $n$  nodes and distances for each pair of nodes, find a roundtrip of minimal total length visiting each node exactly once. The distance from node  $i$  to node  $j$  is the same as from node  $j$  to node  $i$ .

-> TSP data

Best known solutions for symmetric TSPs

<http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html>



## Preparación de los datos

### Algunas funciones generales

```
#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
```

Toma un nodo no elegido anteriormente

```
#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)
```

Se puede mejorar con **functools.reduce** (programación funcional)



# AG3 – Actividad Guiada 3

## Búsqueda aleatoria

### Agenda

1. Librería TSPLIB para el agente viajero - TSP
2. Resolución por búsqueda aleatoria
3. Resolución por Búsqueda local
4. Resolución por recocido simulado
5. Planteamiento por Colonia de Hormigas - ACO



# Metaheurísticas de búsquedas

## Búsqueda aleatoria

**Definición:** Es un proceso por el que se van generando soluciones aleatorias en cada iteración y se devuelve la mejor.

### Inicio

GENERA(Solución Inicial)  
Solución Actual  $\leftarrow$  Solución Inicial;  
Mejor Solución  $\leftarrow$  Solución Actual;

### Repetir

GENERA(Solución Actual);  
**Si** Objetivo(Solución Actual) **es mejor que** Objetivo(Mejor Solución)  
**entonces** Mejor Solución  $\leftarrow$  Solución Actual;

**Hasta** (Criterio de parada);

DEVOLVER (Mejor Solución);

### Fin

Generación aleatoria



Repaso

# Metaheurísticas de búsquedas

## Búsqueda aleatoria

```
def busqueda_aleatoria(problem, N):  
    Nodos = list(problem.get_nodos())  
  
    mejor_solucion = []  
    #mejor_distancia = 10e100  
    mejor_distancia = float('inf')  
  
    for i in range(N):  
        solucion = crear_solucion(Nodos)  
        distancia = distancia_total(solucion, problem)  
  
        if distancia < mejor_distancia:  
            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, 31, 33, 34, 26, 38, 22, 18, 35, 15, 5, 19, 28, 25, 8, 24, 13, 12, 11, 32, 30, 39, 20, 6, 4, 3, 37, 17]  
Distancia : 3495

1273



# AG3 – Actividad Guiada 3

## Búsqueda local

### Agenda

1. Librería TSPLIB para el agente viajero - TSP
2. Resolución por búsqueda aleatoria
3. Resolución por Búsqueda local
4. Resolución por recocido simulado
5. Planteamiento por Colonia de Hormigas - ACO

# Metaheurísticas de búsquedas

## Búsqueda local. Generador de vecindad

### Algoritmos Heurísticos

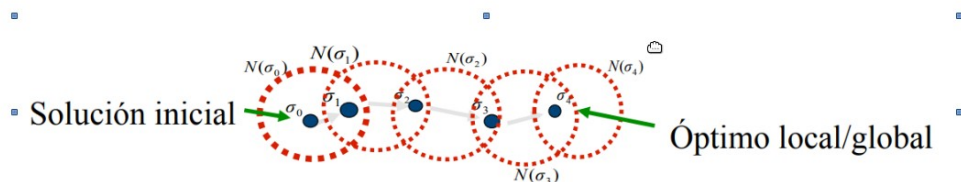
### Asignatura: Algoritmos de Optimización

## Metaheurísticas de búsquedas basadas en trayectorias

### Búsqueda local

#### Elementos básicos:

- ✓ - Establecer **solución inicial**
- ✓ - Establecer una **codificación** para las soluciones
- Establecer un **operador de generación de vecino** (estructura de entorno)
- Establecer un **criterio de parada** para finalizar la iteración



# Metaheurísticas de búsquedas

## Búsqueda local. Generador de vecindad

```
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  
    #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  
  
#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, 4  
print("Distancia Solucion Inicial:" , distancia_total(solucion, problem))  
  
nueva_solucion = genera_vecina(solucion)  
print("Distancia Solucion Local:", distancia_total(nueva_solucion, problem))
```

Prueba todos los posibles intercambios 2-opt  
Total  $41 \times 40 / 2 = 820$  posibilidades

Distancia Solucion Inicial: 3712  
Distancia Solucion Local: 3255

¿Cómo serían otros generadores de vecindad?

- se elige una sub-lista y se invierte el orden
- se elige una sub-lista y se baraja
- otros



# Metaheurísticas de búsquedas

## Búsqueda local.

```
#Búsqueda 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
        #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(problem )

En la iteracion 42, la mejor solución encontrada es: [0, 3, 4, 10, 25, 11, 12, 18, 31, 36, 35, 30, 22, 38, 34, 33, 20,
Distancia      : 1749
```

1273

## Metaheurísticas de búsquedas basadas en trayectorias

Búsqueda local. Desventajas(intensifica pero no diversifica)

- Escapar de máximos(mínimos) locales. 3 opciones:
  - Modificar la estructura de entornos búsqueda en **entornos variables(\*)**
  - Permitir movimientos peores respecto a la solución actual búsqueda tabú, **recocido simulado**
  - Volver a comenzar con otras soluciones iniciales búsquedas **multi-arranque**

Propuesta de mejora para aumentar nota(8/10)

+1 para mejorar

(\*)Búsqueda por Entornos Variables para Planificación Logística:

<https://jamoreno.webs.ull.es/www/papers/VNS2PL.pdf>

### Búsqueda por Entornos Variables para Planificación Logística\*

José Andrés Moreno Pérez  
DEIOC. Instituto Universitario de Desarrollo Regional  
Universidad de La Laguna,  
38271 La Laguna, España  
[jamoreno@ull.es](mailto:jamoreno@ull.es)

Nenad Mladenović  
School of Mathematics, Brunel University,  
London, Reino Unido  
[nenad.mladenovic@brunel.ac.uk](mailto:nenad.mladenovic@brunel.ac.uk)

#### Resumen

La Búsqueda por Entornos Variables (Variable Neighbourhood Search, VNS) es una metaheurística reciente para resolver problemas de optimización cuya idea básica es el cambio sistemático de entorno dentro de una búsqueda local. En este artículo presentamos las reglas básicas de la VNS y sus extensiones para la resolución heurística de una variedad de problemas de optimización. Se ofrece un breve recorrido por los aspectos más relevantes de la aplicación de esta metaheurística en problemas de planificación logística. Finalmente se incluyen algunas reflexiones sobre aspectos esenciales de las metaheurísticas y del análisis de los procesos de solución heurística, y la contribución de los trabajos con la VNS para estas cuestiones.

# AG3 – Actividad Guiada 3

## Recocido Simulado(Simulated Annealing -SA)

### Agenda

1. Librería TSPLIB para el agente viajero - TSP
2. Resolución por búsqueda aleatoria
3. Resolución por Búsqueda local
4. Resolución por recocido simulado
5. Planteamiento por Colonia de Hormigas - ACO



## Metaheurísticas: Recocido simulado - SA

### Esquema básico

Criterio de parada:  
 $T=0$   
ó  
n.º de iteraciones

$T \leftarrow T_0$

Temperatura inicial alta

Generar una solución inicial  $x_1$  en  $X$ ;

$F^* \leftarrow F(x_1)$

$x^* \leftarrow x_1$

**While** la condición de parada no se satisfaga **do**

Generar aleatoriamente un  $x$  en el entorno  $V(x_n)$  de  $x_n$

**if**  $F(x) \leq F(x_n)$ , **then**  $x_{n+1} \leftarrow x$

Criterio de aceptación  
se solución actual

**if**  $F(x) \leq F^*$ , **then**  $F^* \leftarrow F(x)$  y  $x^* \leftarrow x$

**else**, generamos un número  $p$  aleatorio entre  $[0,1]$

**end if**

**if**  $p \leq p(n)$  **then**  $x_{n+1} \leftarrow x$

También se acepta con  
probabilidad  $p(n)$

**end if**

Se disminuye la temperatura según el programa de enfriamiento

**end do**

Revisión

## Metaheurísticas: Recocido simulado - SA

Función de probabilidad  $p(n)$  para aceptar soluciones peores

- Depende la temperatura( $T$ ) y de de la diferencia de costes de las soluciones

$$P_{\text{aceptación}} = \exp(-\delta/T)$$

- A mayor temperatura => mayor probabilidad de aceptar peores soluciones
- A menor diferencia de costes => mayor probabilidad de aceptar peores soluciones

$$\delta = C(s') - C(s)$$

**s = solución actual**  
**s' = solución vecina**

## Metaheurísticas: Recocido simulado - SA

### Generar una vecina aleatoria con operador 2-opt



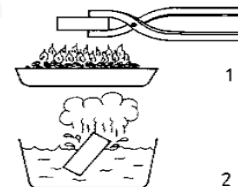
```
#Generador de 1 solución vecina 2-opt aleatoria (intercambiar 2 nodos)
def genera_vecina_aleatorio(solucion):

    #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:]

genera_vecina_aleatorio(solucion)
```

Se eligen aleatoriamente 2 nodos y se intercambian



# Metaheurísticas: Recocido simulado - SA

## Funciones para Probabilidad, Temperatura

```
#Funcion de probabilidad para determinar si se cambia
# a una solución peor respecto a la de referencia(exponencial)
def probabilidad(T,d):
    if random.random() <= math.exp(-1*d / T) :
        return True
    else:
        return False

def bajar_temperatura(T):
    return T*.999
```

AG3 - Actividad Guiada 3 Asignatura: Algoritmos de Optimización

**Metaheurísticas: Recocido simulado - SA**

Función de probabilidad  $p(j)$  para aceptar soluciones peores

- Depende la temperatura ( $T$ ) y de la diferencia de costes de las soluciones

$$P_{\text{aceptacion}} = \exp(-\delta/T)$$

- A mayor temperatura  $\Rightarrow$  mayor probabilidad de aceptar peores soluciones
- A menor diferencia de costes  $\Rightarrow$  mayor probabilidad de aceptar peores soluciones

$\delta = C(s') - C(s)$   
 $s$  = solución actual  
 $s'$  = solución vecija

viu

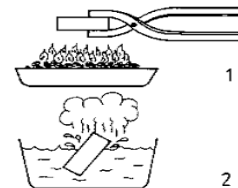
### Algoritmos Heurísticos

### Asignatura: Algoritmos de Optimización

## Metaheurísticas: Recocido simulado - SA

Mecanismos de enfriamiento. Descenso de la temperatura

- Descenso constante
- Basado en descensos sucesivos por tramos dependiendo de la iteración
- Descenso exponencial  $T_{k+1} = \alpha \cdot T_k$
- Criterio de Boltzmann:  $T_k = T_0 / (1 + \log(k))$
- Esquema de Cauchy:  $T_k = T_0 / (1 + k)$

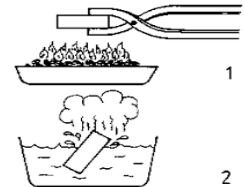


# Metaheurísticas: Recocido simulado - SA

## Recocido simulado(I)

```
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 = []  
    mejor_distancia = 10e100
```

↓  
continua



# Metaheurísticas: Recocido simulado - SA

## Recocido simulado(II)

```
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)
        distancia_referencia = distancia_vecina

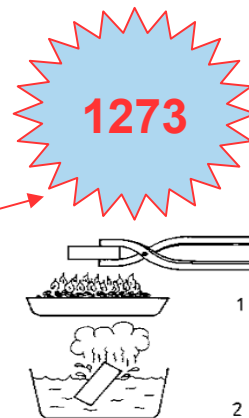
    #Bajamos la temperatura
    TEMPERATURA = bajar_temperatura(TEMPERATURA)

    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, 10000000)
```

Criterio de parada

Nos quedamos con una solución peor en algunas ocasiones



La mejor solución encontrada es [0, 17, 15, 14, 6, 1, 27, 3, 4, 5, 26, 11, 12, 18, 13, 19, 16, 36, 35, 31, 37, 7, 20, 33, 34, :  
con una distancia total de 1906

## Heurísticas. Práctica individual.

Tareas opcionales para **mejorar nota**:

- Búsqueda local con Entornos variables.

¿Se puede mejorar con otros operadores de vecindad variables?

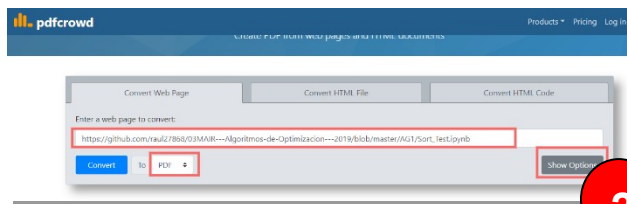
- Recocido simulado

¿Se puede mejorar con otra elección no tan aleatoria (función `genera_vecina_aleatorio()`) ?



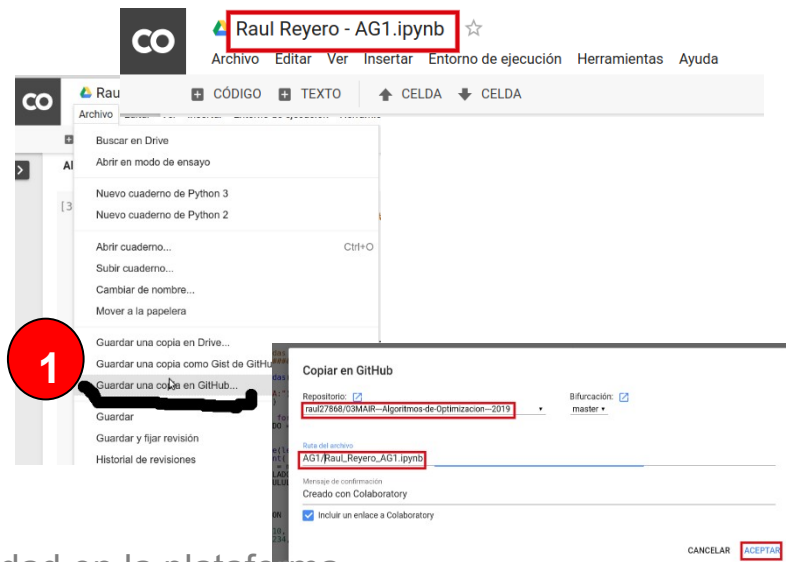
## Finalizar la actividad. Grabar, subir a GitHub, Generar pdf (I)

- Guardar en GitHub  
Repositorio: 03MIAR ---Algoritmos de Optimizacion  
Ruta de Archivo con **AG3**
- Generar pdf con <https://pdfcrowd.com>



- Descargar pdf y adjuntar el documento generado a la actividad en la plataforma
  - Adjuntar .pdf en la actividad
  - URL GitHub en el texto del mensaje de la actividad

3





## Heurísticas. Prácticas para compartir en el Foro

- **Búsqueda Tabú** Usar diferentes tipo de listas
- **GRASP**. ¿Se puede mejorar con un algoritmo voraz, aleatorio y adaptativo?



# AG3 – Actividad Guiada 3

## Colonia de Hormigas - ACO

### Agenda

1. Librería TSPLIB para el agente viajero - TSP
2. Resolución por búsqueda aleatoria
3. Resolución por Búsqueda local
4. Resolución por recocido simulado
5. Planteamiento por Colonia de Hormigas - ACO

# Metaheurísticas. Método Constructivo. Colonia de Hormigas

## Esquema básico

Depositar una cantidad de feromona inicial en todas las aristas

Crear  $m$  hormigas

Repetir:

- Reiniciar hormigas (borrar memoria)

- Cada hormiga: Construir solución usando feromonas y coste de las aristas

- Cada hormiga: Depositar feromonas en aristas de la solución

- Evaporar feromona en las aristas

Devolver: la mejor solución encontrada



## Metaheurísticas. Método Constructivo. Colonia de Hormigas

### Esquema básico (I) (1ª iteración)

```
def hormigas(problem, N) :  
    #problem = datos del problema  
    #N = Número de agentes(hormigas)  
  
    #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)]
```

Propuesta de mejora

Se inicializa todo con ceros

↓  
continua



# Metaheurísticas. Método Constructivo. Colonia de Hormigas

## Esquema básico (II) (1ª iteración)

```
#Recorre cada agente construyendo la solución
for h in range(N) :
    #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 )
        Hormiga[h].append(Nuevo_Nodo)

    #Incrementa feromonas en esa arista
    T = Incrementa_Feromona(problem, T, Hormiga[h] )
    #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:
            mejor_solucion = Hormiga[h]
            mejor_distancia = distancia_actual
```

N=Nº de hormigas

Añade un nuevo nodo

T=Lista de aristas



# Metaheurísticas. Método Constructivo. Colonia de Hormigas

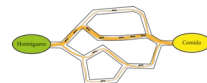
## Funciones auxiliares

```
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_nodos())  
    return random.choice( list(set(range(1,len(Nodos))) - set(H) ) )
```

Poco eficiente, demasiado aleatoria

```
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
```

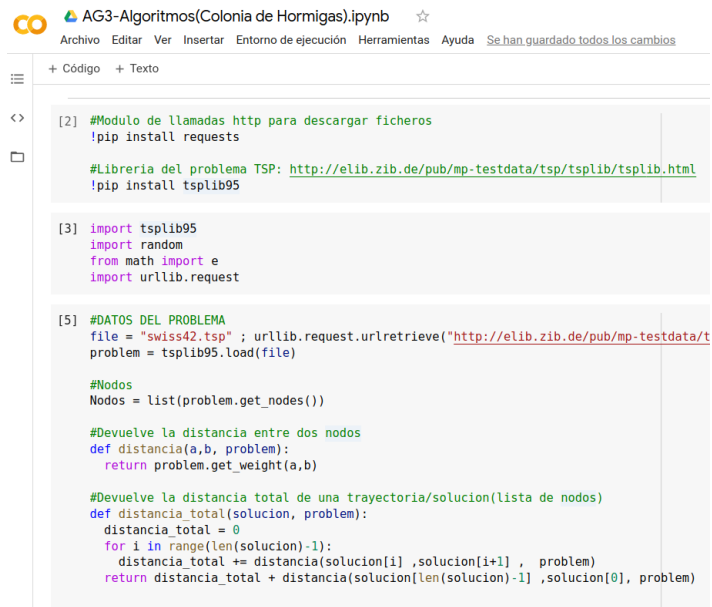
```
def Evaporar_Feromonas(T ):  
    #Evapora 0.3 el valor de la feromona, sin que baje de 1  
    #Mejora:Podemos elegir diferentes funciones de evaporación dependiendo de la cantidad actual y de la suma total de feromonas depositadas,...  
    T = [[ max(T[i][j] - 0.3 , 1) for i in range(len(Nodos))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



# Metaheurísticas. Método Constructivo. Colonia de Hormigas

## Trabajar sobre

<https://colab.research.google.com/drive/1nX4FZJGCCmh31ps8znXMSxbdgimGW0T2?usp=sharing>



```
[2] #Modulo de llamadas http para descargar ficheros
!pip install requests

#Libreria del problema TSP: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html
!pip install tsplib95

[3] import tsplib95
import random
from math import e
import urllib.request

[5] #DATOS DEL PROBLEMA
file = "swiss42.tsp" ; urllib.request.urlretrieve("http://elib.zib.de/pub/mp-testdata/tsp/tsplib95.tsp", file)
problem = tsplib95.load(file)

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

#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(lista de nodos)
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)
```

▼ Algoritmo de colonia de hormigas

## Metaheurísticas. Método Constructivo. Colonia de Hormigas

Función de probabilidad para elegir el siguiente nodo

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha [\nu_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}(t)]^\alpha [\nu_{il}]^\beta}, \text{ si } j \in J_i^k$$

Cantidad de feromonas en la arista (i,j)

Inversa de la distancia de i a j

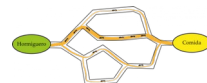
$$p_{ij}^k(t) = 0, \text{ si } j \notin J_i^k \longrightarrow \text{No se elije si no está en los nodos pendientes}$$

$\alpha$       Peso relativo que se da a la elección por el rastro

$\beta$       Peso relativo que se da a la elección por la distancia entre dos nodos

*Si  $\alpha$  es igual a 0 , se asemeja a una técnica voraz*

*Si  $\beta$  es igual a 0 , solo intervine la feromona que puede no ser lo ideal*





# Metaheurísticas. Método Constructivo. Colonia de Hormigas

## Algoritmo ACO aplicado al TSP: Resumen de una experiencia práctica

Benjamin Arenas F., benarenas@hotmail.com  
Alejandro Pavez S., apavez@mi.terra.cl  
Rodrigo Vidal K., rovikak@yahoo.com  
Universidad Técnica Federico Santa María  
Departamento de Informática

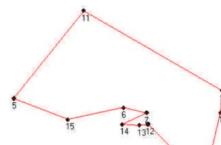
**Abstract:** La metaheurística Ant Colony Optimization (ACO) [4] es uno de los nuevos paradigmas de resolución de problemas combinatorios del tipo NP. En este artículo se presentan los resultados obtenidos al aplicar un algoritmo variante de la metaheurística ACO en torno al mundialmente conocido Traveling Salesman Problem (TSP) [10], se discuten las oportunidades de desarrollo futuro y se presentan nuevos tours solución con mejores resultados que los mundialmente encontrados hasta hoy.

**Palabras claves:** TSP, ACO, Simulated Annealing, Metaheurística

### 1. Introducción

La importancia del TSP dentro del mundo de problemas NP radica en que es utilizado como conjunto de pruebas para los nuevos algoritmos asociados a la resolución de esta clase de problemas. Su naturaleza NP-Completo hace posible que se puedan resolver una gran familia de problemas equivalentes mediante transformaciones particulares para cada caso [8]. Además, como el TSP presenta una gran cantidad de aplicaciones en el mundo real, se ha convertido en uno de los problemas más estudiados por la comunidad científica mundial. Las características de la nueva metaheurística ACO, aplicadas al TSP [2], [3], [7], nos pueden ayudar a encontrar soluciones

dos veces una misma ciudad. El objetivo es minimizar la longitud de la secuencia de visita de las ciudades, bajo el criterio de minimización (maximización) de la función objetivo. Un ejemplo de un tour es el que se muestra a continuación en el figura 1.



[https://www.academia.edu/6676146/Algoritmo\\_ACO\\_aplicado\\_al\\_TSP\\_Resumen\\_de\\_una\\_experiencia\\_pr%C3%A1ctica](https://www.academia.edu/6676146/Algoritmo_ACO_aplicado_al_TSP_Resumen_de_una_experiencia_pr%C3%A1ctica)



## Sesión práctica en grupo

<b>SESIÓN 9</b>	24/01/24	AG3 – Actividad Guiada(1ª parte)
<b>SESIÓN 10</b>	29/01/24	VC6 – Algoritmos genéticos
<b>SESIÓN 11</b>	31/01/24	AG3 – Actividad Guiada(2ª parte)
<b>SESIÓN 12</b>	05/02/24	TC1 - Tutoría Colectiva Final

- práctica de Algoritmo Genético(no evaluable)
- propuestas de problemas prácticos
- dudas
- repaso

## Actividad. Encuesta de satisfacción

- Disponible

**03MIAR**  
**ALGORITMO**  
**S DE**  
**OPTIMIZACI**  
**ÓN**  
INICIO

**INFORMACIÓN GENERAL**

Bienvenida  
Guía didáctica  
Calendario

**ACTIVIDAD FORMATIVA**

Videoconferencias  
Recursos y materiales  
Actividades

### Actividades

 **Encuesta satisfacción alumno con la asignatura y el profesor**  
La Universidad Internacional de Valencia tiene como objetivo conocer y analizar el grado de satisfacción y expectativas del alumnado con respecto a las titulaciones que ofrece con la finalidad de incrementar el grado de satisfacción en cada edición y contribuir con vuestra información a la mejora continua.  
A través de esta encuesta recogeremos vuestra opinión sobre el desarrollo de cada una de las asignaturas y el desempeño de los docentes que han impartido las mismas.  
Vuestra valoración es un elemento fundamental del sistema de calidad y mejora permanente de la Universidad, por lo que solicitamos vuestra colaboración.

 **AG1- Actividad Guiada 1(2ª convocatoria)**  
Desarrollar algoritmos voraces para resolver problemas  
Desarrollar algoritmos con la técnica de vuelta atrás(backtracking) para resolver problemas  
Desarrollar algoritmos con la técnica de divide y vencerás para resolver problemas  
Desarrollar algoritmos con la técnica de programación dinámica para resolver problemas

 **AG2 - Actividad Guiada 2(2ª convocatoria)**  
Desarrollar algoritmos de búsqueda en amplitud para resolver problemas  
Desarrollar algoritmos de búsqueda en profundidad para resolver problemas  
Desarrollar algoritmos con la técnica de ramificación y poda para resolver problemas  
Desarrollar algoritmos con la técnica del descenso del gradiente

# Ampliación de conocimientos y habilidades

## Investigar

### ■ Bibliografía

- Duarte, A. (2008). Metaheurísticas. Madrid: Dykinson.

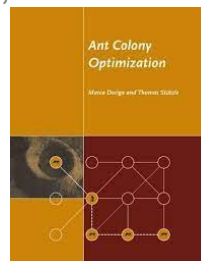
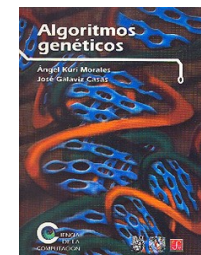
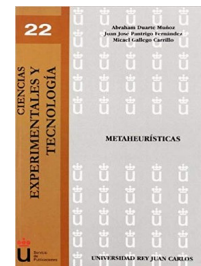
<https://elibro-net.universidadviu.idm.oclc.org/es/ereader/universidadviu/35696>

- Kuri, Á. (2002). Algoritmos genéticos. Instituto Politécnico Nacional.

<https://elibro-net.universidadviu.idm.oclc.org/es/ereader/universidadviu/71925?page=1>

- Dorigo, Marco, et al. Ant Colony Optimization, MIT Press, 2004. ProQuest Ebook Central,

<https://ebookcentral.proquest.com/lib/universidadviu/detail.action?docID=3338887>



## Practicar



## Próximo día:

1. Algoritmos genéticos y evolutivos(\*)
  - Introducción conceptos evolucionistas
  - Fundamentos de los algoritmos evolutivos y genéticos
  - Características de los algoritmos evolutivos y genéticos
  - Componentes de los algoritmos evolutivos y genéticos
2. Problemas del trabajo práctico(repaso)
3. Artículo científico

(\*)Visionar: BIOBOTS y Algoritmos Evolutivos(DotCSV)  
<https://www.youtube.com/watch?v=ULh2IXR-6O4>

## ¿Preguntas?



# Gracias

[raul.reyero@professor.universidadviu.com](mailto:raul.reyero@professor.universidadviu.com)