Universidad de Castilla-La Mancha



Álgebra y Matemática Discreta

Practica Incremental 2023-24 (Grupo Español)

# Optimización en grafos

*Profesores de la asignatura:*
**José Ángel Martín Baos** [Grupo A] (joseangel.martin@uclm.es)
**José Luis Espinosa Aranda** [Prácticas de los Grupos C y D] (josel.espinosa@uclm.es)
**Ricardo García Ródenas** [Grupo B] (ricardo.garcia@uclm.es)
**José Ángel López Mateos** [Teoría de los Grupos C y D]

## A tener en cuenta:

1. La realización de la práctica es individual.
2. Esta práctica se evalúa sobre 2.5 puntos del global de la asignatura y vale aproximadamente un $70\,\%$ de la parte de prácticas de la asignatura.
3. La práctica es una actividad obligatoria, por lo que será necesario obtener al menos un $40\,\%$ de la puntuación (1 punto) para poder aprobar la asignatura.
4. No es necesario realizar todos los hitos para que la práctica pueda ser entregada.
5. Para la evaluación de cada uno de los apartados se tendrán en cuenta tanto la claridad del código como los comentarios incluidos en éste.
6. Para la corrección de la práctica se utilizará un programa de detección de copia. En caso de que la similitud entre dos o más prácticas se encuentren fuera de lo permisible, todas ellas serán calificadas con un $0$.
7. Fecha límite de entrega: Hasta el 19 de Mayo. Se habilitará una tarea en Campus Virtual para poder subirla.
8. Todos los códigos se entregaran en formato script de MATLAB (.m). No se aceptarán otros formatos, como por ejemplo cuadernos de MATLAB (.mlx). Cualquier entrega en otro formato será calificado con 0 puntos.
9. La versión final de los códigos debe entregarse comprimida en un único fichero .zip con el nombre del alumno.

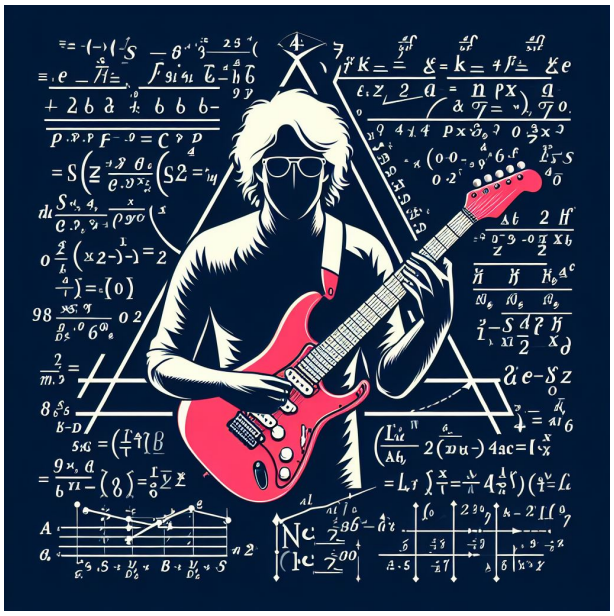

Figura 1: Cartel del grupo *Isomorphic Idols*

## Objetivos de la práctica:

El objetivo de esta práctica es aplicar la teoría de grafos para modelar el siguiente problema de optimización. Supón que el grupo musical *Isomorphic Idols*[1] (Figura 1) que ha alcanzado un gran éxito entre los estudiantes de Ingeniería Informática en España desea realizar una gran gira por todas las ciudades españolas de más de $100,000$ habitantes (Ciudad Real, tristemente no estará en esta gira). El objetivo es diseñar la gira de modo que la distancia recorrida por el grupo sea lo menor posible. En terminología de grafos este problema consiste en encontrar un circuito hamiltoniano de mínimo coste. En esta práctica se plantean varios algoritmos heurísticos para este problema y se analizan numéricamente.

# Hito 1: Generación del problema (0.5 puntos)

El fichero CSV *worldcities.csv* contiene los datos de las ciudades del mundo. Las columnas $3$ y $4$ (lat y lng) contienen la latitud y longitud de la ciudad. La columna $6$ es el código ISO2 del país al que pertenece. En el caso de España este código tiene el valor `ES`. La columna $10$ (population) contiene el número de habitantes de la ciudad.

Usando la plantilla `Milestone1Template.m`, se pide implementar un script de MATLAB, llamado `Milestone1.m`, que realice los siguientes apartados:

a) Obtenga los datos claves (los enunciados anteriormente) de las ciudades españolas de más de $100,000$ habitantes.

[1] También tenemos algunos temas del grupo por si los queréis escuchar: https://soundraw.io/edit_music?m=65fca05fa86213002c104e8d, https://soundraw.io/edit_music?m=65fca0a60f901e0026f34daf, https://soundraw.io/edit_music?m=65fca1433ae9110026c38773

b) Empleando las coordenadas de latitud y longitud obtenga una representación gráfica de las ciudades seleccionadas sobre un mapa del mundo.

**Nota.** Se sugiere explorar las funciones `geobasemap` y `geoscatter` de MATLAB para lograr una representación geográfica precisa, dado que estas herramientas son más apropiadas para este tipo de problema en comparación con el método tradicional de coordenadas cartesianas.

## Hito 2: Cálculo de la matriz distancia (0.25 puntos)

Supondremos que las ciudades están situadas en una esfera idealizada de radio $6,378.388$ kilómetros. Las coordenadas geográficas de longitud y latitud corresponden a puntos de dicha esfera cuyo formato es DDD.MM donde DDD son los grados y MM los minutos. Una latitud positiva es asignada al *Norte*, latitud negativa significa *Sur* (está referenciado con respecto al paralelo principal, que es el ecuador $(0°)$). Longitud positiva significa *Este* y longitud negativa *Oeste* (con respecto al meridiano principal, que es el meridiano de Greenwich $(0°)$). Para calcular la distancia entre la ciudad $i$ y $j$ dentro de esta esfera idealizada realizamos el siguiente proceso. Primero transformamos las coordenadas a radianes mediante el pseudocódigo proporcionado en el Algoritmo 1.

**Algorithm 1:** Algoritmo para convertir las coordenadas de la ciudad $i$ en radianes

**Input** : Coordenadas $x[i]$, $y[i]$ de una determinada ciudad $i$
**Output:** $latitude[i]$ y $longitude[i] \equiv$ Latitud y longitud de la ciudad $i$

$PI \leftarrow 3.141592$;
$deg\_x \leftarrow \text{floor}(x[i])$;
$min\_x \leftarrow x[i] - deg\_x$;
$latitude[i] \leftarrow PI \times (deg\_x + \frac{5.0}{3.0} \times min\_x)/180.0$;
$deg\_y \leftarrow \text{floor}(y[i])$;
$min\_y \leftarrow y[i] - deg\_y$;
$longitude[i] \leftarrow PI \times (deg\_y + \frac{5.0}{3.0} \times min\_y)/180.0$;

La distancia entre dos ciudades $i$ y $j$ en kilómetros es calculada mediante el pseudocódigo proporcionado en el Algoritmo 2. NOTA: La función "acos" es la inversa de la función coseno.

**Algorithm 2:** Algoritmo para calcular la distancia entre las ciudades $i$ y $j$

**Input** : $latitude$ y $longitude$ de dos ciudades $i$ y $j$
**Output:** $d\_ij \equiv$ Distancia entre las ciudades $i$ y $j$

$R \leftarrow 6378.388$ ; `// Radio de la tierra en kilómetros`
$q1 \leftarrow \cos(longitude[i] - longitude[j])$;
$q2 \leftarrow \cos(latitude[i] - latitude[j])$;
$q3 \leftarrow \cos(latitude[i] + latitude[j])$;
$d\_ij \leftarrow R \times \text{acos}\,(0.5 \times ((1.0 + q1) \times q2 - (1.0 - q1) \times q3))$;

Se pide, tomando como partida el fichero `Milestone1.m` generado en el hito anterior, implementar un nuevo script de MATLAB llamado `Milestone2.m` que:

a) Convierta las coordenadas de cada ciudad en radianes.

b) Calcule la matriz de distancia $d$, dónde la fila $i$ y columna $j$ representa la distancia entre las ciudades $i$ y $j$.

## Hito 3: Algoritmo de inserción (0.75 puntos)

Recordemos que el objetivo de la práctica es diseñar una gira de forma que la distancia recorrida por el grupo sea la menor posible. En este caso, se tratará por tanto de un ciclo hamiltoniano de mínimo coste. Sin embargo, obtener una solución para este problema por fuerza bruta resulta muy costoso cuando el numero de ciudades crece de unas pocas decenas.

Supongamos que tenemos un problema más sencillo con únicamente tres ciudades, que denominamos $1, 2, 3$. Una forma sencilla de codificar un ciclo $C$ es construir un vector donde las $n$ primeras componentes son una permutación de los $n$ elementos y la componente $n+1$ es idéntica a la primera (ya que debemos volver a la ciudad de origen al final del tour). Por ejemplo, un posible ciclo sería $C = [2, 1, 3, 2]$, que representaría la gira $2 \to 1 \to 3 \to 2$.

En este hito, proponemos la implementación de un algoritmo heurístico (con un coste computacional menor que el algoritmo de fuerza bruta) que permite aproximar el ciclo hamiltoniano de mínimo coste. El algoritmo heurístico de inserción parte de un ciclo inicial formado por un único vértice (la primera ciudad de la gira), de forma que $C = [1, 1]$. En cada iteración se inserta un nuevo vértice, no presente en el ciclo, en el mejor lugar posible. El Algoritmo 3 muestra el pseudocódigo de este procedimiento.

**Algorithm 3:** Algoritmo de inserción para aproximadamente encontrar un ciclo hamiltoniano de mínimo coste

**Input** : $d \equiv$ Matriz de distancias entre ciudades, $v_1 \equiv$ Vértice (ciudad) inicial y $n \equiv$ Número total de ciudades.

**Output:** $C \equiv$ Ciclo aproximado al ciclo hamiltoniano de mínimo coste.

$C_1 \leftarrow [v, v]$ ; `// Ciclo inicial formado por un solo vértice,` $v$

**for** $i \leftarrow 1 : n-1$ **do**

Se elige un vértice $v_i$ no perteneciente al ciclo en la iteración $i$, es decir, $v_i \notin C_i$;

Si el ciclo en la iteración $i$ fuera $C_i = [w_1, w_2, \cdots w_i, w_{i+1}]$. Se calcula la posición $k^*$ del ciclo dónde insertar el $v_i$ de la manera más óptima. Para ello, se puede usar la expresión:

$$k^* = \underset{1 \le k \le i}{\text{arg minimizar}} \ \left(d(w_k, v_i) + d(v_i, w_{k+1})\right), \tag{1}$$

donde arg minimizar se refiere al valor del índice k donde la expresión $d(w_k, v_i) + d(v_i, w_{k+1})$ alcanza el mínimo;

Se añade el vértice $v_i$ en la posición $k^*$ del ciclo. Es decir, se obtiene un nuevo ciclo $C_{i+1} = [w_1, w_2, \cdots, w_{k^*}, v_i, w_{k^*+1}, \cdots, w_{i+1}]$;

**end**

Es decir, el Algoritmo 3 comienza con un ciclo que contiene únicamente el índice de la ciudad inicial. En cada iteración se va seleccionando una ciudad que no se encuentre en la gira

(en el ciclo) y procede a buscar el punto más adecuado del ciclo en el que añadirla. Para ello, se comprueba en que posición del ciclo la distancia entre la ciudad anterior, la ciudad que se quiere añadir y la ciudad siguiente, es la mínima posible.

Pongamos un ejemplo, si tenemos el ciclo [Madrid, Barcelona, Málaga, Madrid] y queremos añadir Sevilla, debemos comprobar en que posición se debe añadir para minimizar el coste de viajar desde la ciudad anterior a Sevilla y luego ir a la siguiente ciudad. En este caso, $k^* = 3$ y el ciclo resultante es [Madrid, Barcelona, Málaga, Sevilla, Madrid].

Se pide:

a) Implementa una función de MATLAB llamada `insertionAlgorithm(d,v,n)` que contenga el algoritmo de inserción. Dada la matriz $d$ y un vértice inicial como input esta función debe devolver el ciclo $C$ como output.

b) Usando como base el hito anterior, crea un script de MATLAB denominado `Milestone3.m` que calcule la solución con el algoritmos inserción y represente gráficamente el circuito obtenido. Además, imprime por consola la lista de las ciudades visitadas y en que orden. **Nota.** Usa la función MATLAB `geoplot` para la representación gráfica.

c) Implementar en MATLAB una función denominada `cost(C,d)` que tome como entrada un circuito $C$ y la matriz distancias $d$ y devuelva como salida la longitud del circuito $C$ (expresada en kilómetros). A continuación, modifica el script `Milestone3.m` para que imprima el número total de kilómetros recorridos por la banda durante la gira.

# Hito 4: Algoritmo de intercambio (0.5 puntos)

El azar es un elemento que se incorpora en el diseño de ciertos algoritmos de optimización. En este hito introducimos un algoritmo probabilístico que denominamos de *intercambio*. Este algoritmo se basa en la idea de modificar aleatoriamente los componentes de una solución para encontrar configuraciones más óptimas. Esta técnica introduce variabilidad y flexibilidad en el proceso de búsqueda, incrementando las posibilidades de hallar la solución más favorable en espacios de solución complejos.

El algoritmo de intercambio parte de un ciclo inicial $C$ (una gira completa). A continuación, selecciona aleatoriamente dos vértices, $v_i$ y $v_j$, cumpliendo la condición de que el vértice $v_i$ aparezca en el ciclo $C$ antes que el vértice $v_j$. Seleccionados los vértices, se invierte el orden de todos los vértices comprendidos entre $v_i$ y $v_j$ (ambos incluidos). En este caso el nuevo ciclo se expresa:

$$C_{\text{new}} = [v_1, v_2, \cdots, v_{i-1}, \text{flip}([v_i, \cdots, v_j]), v_{j+1}, \cdots, v_n, v_1] \tag{2}$$

donde la función flip(x) invierte el orden de las componentes del vector $x$. Si la longitud del nuevo circuito (coste) es menor que el coste del ciclo original entonces el algoritmo acepta como actual el nuevo ciclo y vuelve a repetir el proceso.

Por ejemplo, supongamos que tenemos el ciclo $C = [1, 2, 3, 4, 5, 6, 7, 8, 1]$ y seleccionamos aleatoriamente los vértices $v_i = 3$ y $v_j = 6$. Aplicando el procedimiento anterior, generaremos un nuevo ciclo $C_{\text{new}} = [1, 2, 6, 5, 4, 3, 7, 8, 1]$. Finalmente, comprobaremos si el nuevo ciclo tiene un coste inferior al ciclo original, y en dicho caso mantendremos el nuevo ciclo generado.

El Algoritmo 4 recoge un pseudocódigo del algoritmo de intercambio.

**Algorithm 4:** Algoritmo de intercambio para aproximadamente encontrar un ciclo hamiltoniano de mínimo coste en $G = (V, E, d)$

**Input** : $C = [v_1, \cdots, v_n, v_1] \equiv$ Ciclo inicial, $Niter \equiv$ Número de iteraciones a realizar, $d \equiv$ Matriz de distancias entre ciudades, y $n \equiv$ Número total de ciudades.
**Output:** $C \equiv$ Ciclo aproximado al ciclo hamiltoniano de mínimo coste.
**for** $k \leftarrow 1 : Niter$ **do**
  Se eligen aleatoriamente dos vértices $v_i$ y $v_j$;
  Calcular $C_{\text{new}}$ usando la expresión (2);
  **if** $cost(C_{new}) < cost(C)$ **then**
    $C = C_{\text{new}}$;
  **end**
**end**

En este hito se pide:

a) Implementar en MATLAB una función denominada `flipAlgorithm(C,Niter,d,n)` que contenga el algoritmo de intercambio, de forma similar a lo realizado en el hito anterior. Usa la función `cost(C,d)` implementada en el hito anterior.

b) Crea un script de MATLAB denominado `Milestone4.m` que calcule la solución con el algoritmo de intercambiando haciendo uso de $Niter = 10000$ y considerando como ciclo inicial $C = [1, 2, \cdots, n, 1]$ (repitiendo la primera ciudad al final para cerrar el ciclo). Represente gráficamente el circuito obtenido. Además, imprime por consola la lista de las ciudades visitadas y en que orden, así como el número total de kilómetros recorridos por la banda durante la gira. **Nota.** Usa la función MATLAB `geoplot` para la representación gráfica.

## Hito 5: Un algoritmo heurístico de tu invención (hasta 0.5 puntos extras)

Este hito es ópcional. Desarrolla un algoritmo heurístico que sea más eficiente que los dos anteriores. Este algoritmo puede consistir en una mejora de los anteriores, una hibridación de ambos o cualquier otra idea que pueda mejorar el rendimiento del mismo. Para que este hito pueda ser valorado, la solución propuesta por el estudiante debe ser original. Con este nuevo algoritmo realiza los siguientes puntos:

a) Implementar una función MATLAB con el algoritmo propuesto por el alumno de forma similar a lo realizado en los hitos anteriores.

b) Genera un fichero .txt o .pdf con el nombre "ExplicacionMilestone5" que contenga la explicación detallada del algoritmo propuesto por el alumno, así como los pasos seguidos para llegar a él. Enumera las ventajas y/o inconvenientes que pueda tener este nuevo algoritmo con respecto los anteriores. Cita todas las fuentes que hayas consultado para su elaboración.

c) Crea un script de MATLAB denominado `Milestone5.m` que calcule la solución con el algoritmo del alumno, imprima por pantalla el coste obtenido y represente gráficamente el circuito hallado.

## Hito 6: Comparación computacional (0.5 puntos)

El objetivo de este hito es realizar una comparativa de los tres (o dos) algoritmos anteriores, analizando el valor de la solución alcanzada (coste) y el tiempo computacional empleado (medido en segundos de CPU).

Se pide implementar un script de MATLAB llamado `Milestone6.m` que realice los siguientes apartados:

a) Genere los problemas para los países de España, Alemania, Francia e Italia.

b) Rellene automáticamente la tabla 1 y la muestre por consola.

| | ESPAÑA | | ALEMANIA | | FRANCIA | | ITALIA | |
|---|---|---|---|---|---|---|---|---|
| | cost(km) | CPU(s) | cost(km) | CPU(s) | cost(km) | CPU(s) | cost(km) | CPU(s) |
| Insertion Alg. | | | | | | | | |
| Flip Alg. | | | | | | | | |
| Student Alg. | | | | | | | | |

Cuadro 1: Comparativa computacional entre algoritmos heurísticos