Algoritmos de optimización - Seminario

Nombre y Apellidos: Francisco González Ramos

Url: https://github.com/fgramos/SEMINARIO

Problema:

> ### 1. Sesiones de doblaje

Descripción del problema:

Se precisa coordinar el doblaje de una película. Los actores del doblaje deben coincidir en las tomas en las que sus personajes aparecen juntos en las diferentes tomas. Los actores de doblaje cobran todos la misma cantidad por cada día que deben desplazarse hasta el estudio de grabación independientemente del número de tomas que se graben. No es posible grabar más de 6 tomas por día. El objetivo es planificar las sesiones por día de manera que el gasto por los servicios de los actores de doblaje sea el menor posible. Los datos son:

> Número de actores: 10 Número de tomas : 30 Actores/Tomas : https://bit.ly/36D8IuK

- 1 indica que el actor participa en la toma
- 0 en caso contrario

(*) La respuesta es obligatoria

Tenemos que generar un máximo de 6 sesiones de doblaje diarias.

```
Las tomas son diferentes, pueden intervenir los mismos actores o distintos.

Para abordar el problema, podríamos agrupar en un mismo día 6 tomas diferentes.

Pero si coinciden todos los actores en las mismas tomas, el coste es menor, ya que si no coinciden los mismos actores
en las tomas de ese día, hay que pagar a todos los actores, con lo que podemos tener costes de 1 dia de trabajo para un
actor que ha hecho una única toma
```

La solución ideal sería que en un día de trabajo (6 tomas) participaran los mismos actores en todas las tomas. Y si pudieramos agrupar esta solución todos los días hasta que se acaben las tomas, sería perfecto.

Veamos las combinaciones de tomas que podemos hacer en cada día de trabajo

(*)¿Cuantas posibilidades hay sin tener en cuenta las restricciones?

Las posibilidades que hay **teniendo en cuenta las restricciones**:

1. Podemos grabar 6 escenas como máximo por día
2. Tenemos 30 escenas en total en el ejemplo de doblaje
3. Finalizar el doblaje cuanto antes (no es útil alargar rodaje con dias de menos de 6 sesiones)
4. Minimizar el coste total de doblajes (a un actor se le paga lo mismo por una sesión que por seis, dentro del mismo día)

> Tendremos que hacer 5 grupos de 6 sesiones para tener las 30 tomas en el mejor caso (hacer menos sesiones por día supone mas días)

> Son combinaciones sin repetición, es decir: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

> En nuestro problema la complejidad será $\binom{Nº Total Sesiones}{Maximo Diario} = \frac{Nº Total Sesiones!}{Maximo Diario!(Nº Total Sesiones - Maximo Diario)!}$

> > $\binom{30}{6} = \frac{30!}{6!(30-6)!} = \frac{30 \cdot 29 \cdot 28 \cdot 27 \cdot 26 \cdot 25 \cdot 24!}{6! \cdot 24!} = \frac{30 \cdot 29 \cdot 28 \cdot 27 \cdot 26 \cdot 25}{6!} = \frac{427.518.000}{6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = 593.775$

> 593.775 Soluciones distintas con sus costes asociados.

Si **no tenemos restricciones**, no tendremos que hacer grupos de 6 escenas, con lo cual, podemos hacer todas las escenas en un día (suponiendo que la suma de tiempos no llegue a 24 horas y que nos de igual que los actores trabajen más de 8 horas y sin pausa para el bocata) Por lo tanto, tendremos que eliminar las restricciones

- No nos importa el orden de grabación de las escenas.

- Podemos grabarlas alternativamente, ya que se montan por separado
- Aunque nos sobra el dinero y no nos importa el coste, le pagamos un día de trabajo a cada uno

> 1 solución = pagamos **10** sesiones, 1 a cada actor (10 actores)

(*) ¿Cual es la estructura de datos que mejor se adapta al problema? Argumentalo.

Vamos a plantear el problema como un árbol de búsqueda en profundidad *(DFS depth first search)*, para obtener grupos de 6 sesiones en el descenso.

Utilizaremos una lista de listas para simular una matriz de Sesiones x Actores ordenada, con sesiones en filas y actores en columnas para almacenar las sesiones.

Sesion = [1,2,3,4...9,10] = Sesion con actores 1,2,3,...N en posiciones 0,1,2,...,n-1 respectivamente, donde en cada posición habrá un 1 o un 0 dependiendo si el actor participa o no en la sesión de doblaje.

Según el modelo para el espacio de soluciones

(*)¿Cual es la función objetivo?

(*)¿Es un problema de maximización o minimización?

$$F(x) = Min(coste(s1 + s2 + s3 + \dots s6) + coste(s7 + s8 + \dots s12) + coste(s13 + s14 + \dots s18) + coste(s19 + s20 + \dots s24) + coste(s25 + s26 + \dots s$$

Siendo $Si = sesión de grabación$ sin importar el orden, pero no se repiten

Es un problema de Minimización de costes

Diseña un algoritmo para resolver el problema por fuerza bruta

In [12]:
```
import math           #Modulo de funciones matematicas. Se usa para exp
import random         #Para generar valores aleatorios

# inicializamos datos de tomas del problema en lista de listas TOMAS

TOMAS =[[1,1,1,1,1,0,0,0,0,0], #1
        [0,0,1,1,1,0,0,0,0,0], #2
        [0,1,0,0,1,0,1,0,0,0], #3
        [1,1,0,0,0,0,1,1,0,0], #4
```

```
        [0,1,0,1,0,0,0,1,0,0], #5
        [1,1,0,1,1,0,0,0,0,0], #6
        [1,1,0,1,1,0,0,0,0,0], #7
        [1,1,0,0,0,1,0,0,0,0], #8
        [1,1,0,1,0,0,0,0,0,0], #9
        [1,1,0,0,0,1,0,0,1,0], #10
        [1,1,1,0,1,0,0,1,0,0],
        [1,1,1,1,0,1,0,0,0,0],
        [1,0,0,1,1,0,0,0,0,0],
        [1,0,1,0,0,1,0,0,0,0],
        [1,1,0,0,0,0,1,0,0,0],
        [0,0,0,1,0,0,0,0,0,1],
        [1,0,1,0,0,0,0,0,0,0],
        [0,0,1,0,0,1,0,0,0,0],
        [1,0,1,0,0,0,0,0,0,0],
        [1,0,1,1,1,0,0,0,0,0],
        [0,0,0,0,0,1,0,1,0,0],
        [1,1,1,1,0,0,0,0,0,0],
        [1,0,1,0,0,0,0,0,0,0],
        [0,0,1,0,0,1,0,0,0,0],
        [1,1,0,1,0,0,0,0,0,1],
        [1,0,1,0,1,0,0,0,1,0],
        [0,0,0,1,1,0,0,0,0,0],
        [1,0,0,1,0,0,0,0,0,0],
        [1,0,0,0,1,1,0,0,0,0], #29
        [1,0,0,1,0,0,0,0,0,0]] #30

# Funciones auxiliares

def coste_diario( Mat_Tomas,a,b,c,d,e,f):
    # Devuelve el coste de 6 sesiones especificadas en a,b,c,d,e,f del 0 al 29
    coste_total=0
    for i in range(len(Mat_Tomas[a])): # comparamos las posiciones 0 a n actores y si participan, sumo 1
        #print( i)
        if Mat_Tomas[a][i] == 1 or Mat_Tomas[b][i] == 1  or Mat_Tomas[c][i] == 1 or Mat_Tomas[d][i] == 1 or Mat_Tomas[e][i] == 1  or Mat_To
            coste_total += 1
    return coste_total

# función busqueda en profundidad sin costes

def selecciona_todos(TOMAS,usadas):
    # realiza búsqueda en profundidad y devuelve combinacion de sesiones
    contador = 0
    ciclos = len(TOMAS)
    for i in range(ciclos):
        if i not in usadas: # Si ya he seleccionado la sesión en una combinacion anterior, no lo uso
```

```
            for j in range(i+1,ciclos):
                if j not in usadas: # Si ya he seleccionado la sesión en una combinacion anterior, no lo uso
                    for k in range(j+1,ciclos):
                        if k not in usadas: # Si ya he seleccionado la sesión en una combinacion anterior, no lo uso
                            for m in range(k+1,ciclos):
                                if m not in usadas: # Si ya he seleccionado la sesión en una combinacion anterior, no lo uso
                                    for n in range(m+1,ciclos):
                                        if n not in usadas: # Si ya he seleccionado la sesión en una combinacion anterior, no lo uso
                                            for o in range(n+1,ciclos):
                                                if o not in usadas: # Si ya he seleccionado la sesión en una combinacion anterior, no lo us
                                                    contador += 1 #contador de operaciones
                                                    coste_6_sesiones = coste_diario(TOMAS,i,j,k,m,n,o)
                                                    combinacion = [i,j,k,m,n,o]
    return combinacion, coste_6_sesiones, contador


def fuerza_bruta(TOMAS):
    #bucle que recorrerá todas las posibilidades
    ciclos = len(TOMAS)
    coste_total = 0
    sesiones_selectas = [] #Guardaremos las sesiones que hemos seleccionado, para no repetirlas
    for dias in (range(5)):
        c1,c2,c3 = selecciona_todos(TOMAS,sesiones_selectas)
        sesiones_selectas.extend(c1)
        coste_total += c2
        print(f"Combinación elegida:{c1} coste:{c2} contador:{c3}")
    print(f"Coste total:{coste_total}  Sesiones consumidas:{sesiones_selectas}")
    return

fuerza_bruta(TOMAS)
```

```
Combinación elegida:[24, 25, 26, 27, 28, 29] coste:8 contador:593775
Combinación elegida:[18, 19, 20, 21, 22, 23] coste:7 contador:134596
Combinación elegida:[12, 13, 14, 15, 16, 17] coste:8 contador:18564
Combinación elegida:[6, 7, 8, 9, 10, 11] coste:8 contador:924
Combinación elegida:[0, 1, 2, 3, 4, 5] coste:7 contador:1
Coste total:38  Sesiones consumidas:[24, 25, 26, 27, 28, 29, 18, 19, 20, 21, 22, 23, 12, 13, 14, 15, 16, 17, 6, 7, 8, 9, 10, 11, 0, 1, 2, 3,
4, 5]
```

Calcula la complejidad del algoritmo por fuerza bruta

El algoritmo "*fuerza bruta*" se le llama 5 veces (de 0 a 4) y en cada ocurrencia, tiene complejidad distinta:

- en la primera llamada $\binom{30}{6}$
- en la segunda llamada $\binom{30-6}{6}$

- en la tercera llamada $\binom{30-12}{6}$
- en la cuarta llamada $\binom{30-18}{6}$
- en la última llamada $\binom{30-24}{6}$

Siendo n el número de sesiones a realizar (n=30 en nuestro caso). Se eliminan 6 sesiones en cada llamada (i·6) siendo i el número de llamadas, (de 0 a 4)

> Por lo que son $= \binom{30}{6} + \binom{30-6}{6} + \binom{30-12}{6} + \binom{30-18}{6} + \binom{30-24}{6}$

> Haciendo las restas $= \binom{30}{6} + \binom{24}{6} + \binom{18}{6} + \binom{12}{6} + \binom{6}{6}$

> Operando $= 593775 + 134596 + 18564 + 924 + 1$ = **747.860** es la complejidad

(*)Diseña un algoritmo que mejore la complejidad del algortimo por fuerza bruta. Argumenta porque crees que mejora el algoritmo por fuerza bruta

In [13]:
```
# función busqueda en profundidad con cálculo de costes mínimos

def selecciona_mejores(TOMAS,usadas):
    contador = 0
    coste_minimo = float('inf')                    #Inicializamos con un valor alto
    combinacion_minima = []
    ciclos = len(TOMAS)

    for i in range(ciclos):
        if i not in usadas: # Si ya he seleccionado la sesión en una combinacion anterior, no lo uso
            for j in range(i+1,ciclos):
                if j not in usadas: # Si ya he seleccionado la sesión en una combinacion anterior, no lo uso
                    for k in range(j+1,ciclos):
                        if k not in usadas: # Si ya he seleccionado la sesión en una combinacion anterior, no lo uso
                            for m in range(k+1,ciclos):
                                if m not in usadas: # Si ya he seleccionado la sesión en una combinacion anterior, no lo uso
                                    for n in range(m+1,ciclos):
                                        if n not in usadas: # Si ya he seleccionado la sesión en una combinacion anterior, no lo uso
                                            for o in range(n+1,ciclos):
                                                if o not in usadas: # Si ya he seleccionado la sesión en una combinacion anterior, no lo usa
                                                    contador += 1 #contador de operaciones
                                                    coste_6_sesiones = coste_diario(TOMAS,i,j,k,m,n,o)
                                                    if coste_minimo > coste_6_sesiones :
                                                        coste_minimo = coste_6_sesiones
                                                        combinacion_minima = [i,j,k,m,n,o]
    return combinacion_minima, coste_minimo, contador
```

```python
def genera_jornadas(TOMAS):
    #bucle que recorrerá todas las posibilidades
    ciclos = len(TOMAS)
    coste_total = 0
    sesiones_selectas = [] #Guardaremos las sesiones que hemos seleccionado, para no repetirlas
    for dias in (range(5)):
        c1,c2,c3 = selecciona_mejores(TOMAS,sesiones_selectas)
        sesiones_selectas.extend(c1)
        coste_total += c2
        print(f"Combinación mínima:{c1} coste:{c2} contador:{c3}")
    print(f"Coste total:{coste_total}  Sesiones consumidas:{sesiones_selectas}")
    return

genera_jornadas(TOMAS)
```

```
Combinación mínima:[13, 16, 17, 18, 22, 23] coste:3 contador:593775
Combinación mínima:[1, 12, 19, 26, 27, 29] coste:4 contador:134596
Combinación mínima:[0, 2, 5, 6, 8, 14] coste:6 contador:18564
Combinación mínima:[3, 4, 7, 11, 20, 21] coste:7 contador:924
Combinación mínima:[9, 10, 15, 24, 25, 28] coste:9 contador:1
Coste total:29  Sesiones consumidas:[13, 16, 17, 18, 22, 23, 1, 12, 19, 26, 27, 29, 0, 2, 5, 6, 8, 14, 3, 4, 7, 11, 20, 21, 9, 10, 15, 24, 2
5, 28]
```

(*)Calcula la complejidad del algoritmo

El algoritmo *genera_jornadas* tiene prácticamente la misma complejidad que el de fuerza bruta, solamente hay que añadir una condición y 2 asignaciones en caso de que se encuentre una buena combinación con coste mínimo

El algoritmo "*genera_jornadas*" se le llama 5 veces (de 0 a 4) y en cada ocurrencia, tiene complejidad distinta:

- en la primera llamada $\binom{30}{6}$
- en la segunda llamada $\binom{30-6}{6}$
- en la tercera llamada $\binom{30-12}{6}$
- en la cuarta llamada $\binom{30-18}{6}$
- en la última llamada $\binom{30-24}{6}$

Siendo n el número de sesiones a realizar (n=30 en nuestro caso). Se eliminan 6 sesiones en cada llamada (i·6) siendo i el número de llamadas, (de 0 a 4)

> Por lo que son $= \binom{30}{6} + \binom{30-6}{6} + \binom{30-12}{6} + \binom{30-18}{6} + \binom{30-24}{6}$

> Haciendo las restas $= \binom{30}{6} + \binom{24}{6} + \binom{18}{6} + \binom{12}{6} + \binom{6}{6}$

> Operando $= 593775 + 134596 + 18564 + 924 + 1$ = **747.860** es la complejidad

Según el problema (y tenga sentido), diseña un juego de datos de entrada aleatorios

In [17]:
```
TOMAS2=[[0,0,0,0,0,0,0,0,0,1], #1
        [0,0,0,0,0,0,0,0,1,0], #2
        [0,0,0,0,0,0,0,1,0,0], #3
        [0,0,0,0,0,0,1,0,0,0], #4
        [0,0,0,0,0,1,0,0,0,0], #5
        [0,0,0,0,1,0,0,0,0,0], #6
        [0,0,0,1,0,0,0,0,0,0], #7
        [0,0,1,0,0,1,0,0,0,0], #8
        [0,1,0,0,0,0,0,0,0,0], #9
        [1,0,0,0,0,0,0,0,0,0], #10
        [1,0,1,0,1,0,1,0,1,0],
        [0,1,0,1,0,1,0,1,0,1],
        [1,1,1,1,1,0,0,0,0,0],
        [0,0,0,0,0,1,1,1,1,1],
        [0,0,0,0,0,1,1,0,0,0],
        [0,0,1,1,0,0,0,0,0,1],
        [1,0,1,0,1,0,0,0,0,0],
        [0,0,1,0,0,1,0,0,1,0],
        [1,0,1,0,0,0,0,0,0,0],
        [1,0,1,1,1,0,0,0,0,0],
        [0,0,1,0,0,1,0,0,0,0],
        [1,0,0,1,0,0,0,0,0,0],
        [1,0,1,0,0,0,0,0,0,0],
        [1,1,0,1,1,0,1,1,0,0],
        [0,0,1,0,0,0,0,0,1,0],
        [0,1,0,1,0,0,0,0,1,0],
        [0,0,0,0,0,1,1,0,0,0],
        [0,1,0,1,0,0,0,0,0,0],
        [1,0,1,0,1,0,0,0,0,0], #29
        [1,0,0,0,0,1,0,1,0,1]] #30
```

Aplica el algoritmo al juego de datos generado

In [18]:
```
genera_jornadas(TOMAS2)
```

```
Combinación mínima:[1, 4, 7, 17, 20, 24] coste:3 contador:593775
Combinación mínima:[5, 9, 16, 18, 22, 28] coste:3 contador:134596
Combinación mínima:[3, 6, 8, 14, 26, 27] coste:4 contador:18564
Combinación mínima:[0, 2, 11, 15, 21, 29] coste:7 contador:924
Combinación mínima:[10, 12, 13, 19, 23, 25] coste:10 contador:1
Coste total:27  Sesiones consumidas:[1, 4, 7, 17, 20, 24, 5, 9, 16, 18, 22, 28, 3, 6, 8, 14, 26, 27, 0, 2, 11, 15, 21, 29, 10, 12, 13, 19, 2
3, 25]
```

Enumera las referencias que has utilizado(si ha sido necesario) para llevar a cabo el trabajo

- *Algoritmos de Optimización* . Raúl Reyero Díez . Guia Documentación Master UIA VIU
- *VIU-03MIAR-Sesion 06*- VC4 - Descenso del gradiente y Trabajo Práctico.pdf
- Algorítmia - Tema 5. Backtracking. N-Reinas - Andrés Muñoz Ortega
- Programación Dinámica: Introducción
- ¿Qué es el Descenso del Gradiente? Algoritmo de Inteligencia Artificial | DotCSV
- Branch And Bound—Why Does It Work?

Describe brevemente las lineas de como crees que es posible avanzar en el estudio del problema. Ten en cuenta incluso posibles variaciones del problema y/o variaciones al alza del tamaño

En este problema se necesitaría realizar una poda para disminuir la complejidad del problema, pero el problema hace una búsqueda en profundidad no para evaluar si los nodos inferiores son buenos caminos, sino para construir las distintas combinaciones de sesiones, por lo que no se cómo hacer una poda distinta.

Actualmente, cuando el algoritmo desciende un nivel para seleccionar un nodo, descarta como siguiente nodo las sesiones que ya han sido seleccionadas anteriormente, por lo que no recorre esos nodos. No puede considerarse como Poda ya que no evalua el nodo como un máximo o un mínimo para descartar esa rama, sino que directamente no la explora

Este algoritmo podría mejorarse buscando las combinaciones de nodos desde un valor aleatorio, y seguir explorando cada nodo aleatoriamente o por orden Es decir, en vez de comenzar por la sesión 0 y explorar la 1, 2, etc como actualmente, podría empezar en la sesión p.ej.15 y seguir comparando por otra aleatoria p.ej. 22. Esto haría combinaciones no por orden y puede encontrarse mejoras. Este método nos requiere tener estructura tipo lista para pasarla entre los nodos para llevar el registro de los nodos visitados.

Es de destacar que el algoritmo "selecciona_mejores" y el "fuerza_bruta" son candidatos a una mejora mediante técnica de Backtraking.