

Algoritmos de optimización - Seminario

Nombre y Apellidos: Rubén Cuenca Carbonell

Url:

Problema:<https://github.com/RCuencaOrbita/algoritmos-optimizacion/tree/main/Practica>

1. Sesiones de doblaje

Descripción del problema:

(*) La respuesta es obligatoria

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 toda 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.

Toma\Actor	1	2	3	4	5	6	7	8	9	10	Total
1 1 1 1 1 1 0 0 0 0 0 5	2 0 0 1 1 1 0 0 0 0 0 3	3 0 1 0 0 1 0 0 0 0 0 3	4 1 1 0 0 0 1 1 0 0 0 4	5 0 1 0 1 0 0 0 1 0 0 3	6 1 1 0 1 1 0 0 0 0 0 4	7 1 1 0 1 1 0 0 0 0 0 4	8 1 1 0 0 0 1 0 0 0 0 3	9 1 1 0 1 0 0 0 0 0 0 3	10 1 1 0 0 0 1 0 0 1 0 4	11 1 1 1 0 1 0 0 1 0 0 5	12 1 1 1 1 0 1 0 0 0 0 5
13 1 0 0 1 1 0 0 0 0 0 3	14 1 0 1 0 0 1 0 0 0 0 3	15 1 1 0 0 0 0 1 0 0 0 3	16 0 0 0 1 0 0 0 0 0 0 1 2	17 1 0 1 0 0 0 0 0 0 0 2	18 0 0 1 0 0 1 0 0 0 0 2	19 1 0 1 0 0 0 0 0 0 0 2	20 1 0 1 1 1 0 0 0 0 0 2	21 0 0 0 0 1 0 1 0 0 0 2	22 1 1 1 1 0 0 0 0 0 0 4	23 1 0 1 0 0 0 0 0 0 0 2	24 0 0 1 0 0 1 0 0 0 0 2
25 1 1 0 1 0 0 0 0 0 1 4	26 1 0 1 0 0 0 0 1 0 0 4	27 0 0 0 1 1 0 0 0 0 0 2	28 1 0 0 1 0 0 0 0 0 0 2	29 1 0 0 0 1 1 0 0 0 0 3	30 1 0 0 1 0 0 0 0 0 0 2						TOTAL 22 14 13 15 11 8 3 4 2 2

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

¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.

Respuesta

```
In [36]: import math
#si no tenemos en cuenta las restricciones, todos los posibles subconjuntos del conjunto de tomas
#el cardinal del conjunto potencia es |P|=2^|Tomas|
print('|P|=',2**30)

#Con las restricciones, hay 30 tomas, tendremos que hacer 6 sesiones de 5 tomas
#las combinaciones posibles son C(n, k) = n! / (k! * (n-k)! ) (n=30 tomas, k=6 tomas)
print('C(30,6)=',math.factorial(30)/(math.factorial(6)*math.factorial(30-6)))

|P|= 1073741824
C(30,6)= 593775.0
```

Modelo para el espacio de soluciones

(*) ¿Cuál es la estructura de datos que mejor se adapta al problema? Argumentalo.(Es posible que hayas elegido una al principio y veas la necesidad de cambiar, argumentalo)

Respuesta

```
In [37]: import numpy as np

#Modelo el problema con una matriz de tomas Las tomas en filas (x) y los actores en
tomas=[ \
[1,1,1,1,1,0,0,0,0,0], \
[0,0,1,1,1,0,0,0,0,0], \
[0,1,0,0,1,0,1,0,0,0], \
[1,1,0,0,0,0,1,1,0,0], \
[0,1,0,1,0,0,0,1,0,0], \
[1,1,0,1,1,0,0,0,0,0], \
[1,1,0,1,1,0,0,0,0,0], \
[1,1,0,0,0,1,0,0,0,0], \
[1,1,0,1,0,0,0,0,0,0], \
[1,1,0,0,0,1,0,0,1,0], \
[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], \
[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], \
[1,0,0,1,0,0,0,0,0,0] \
]

#la solución será una matriz de 5 días con 6 tomas en cada día
#ejemplo:
dia1=[tomas[0],tomas[1],tomas[2],tomas[3],tomas[4],tomas[5]]
dia2=[tomas[6],tomas[7],tomas[8],tomas[9],tomas[10],tomas[11]]
dia3=[tomas[12],tomas[13],tomas[14],tomas[15],tomas[16],tomas[17]]
dia4=[tomas[18],tomas[19],tomas[20],tomas[21],tomas[22],tomas[23]]
dia5=[tomas[24],tomas[25],tomas[26],tomas[27],tomas[28],tomas[29]]

solucion = [dia1,dia2,dia3,dia4,dia5]
display(solucion)
```

```
[[[1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
 [0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 1, 0, 1, 0, 0, 0],
 [1, 1, 0, 0, 0, 1, 1, 0, 0, 0],
 [0, 1, 0, 1, 0, 0, 0, 1, 0, 0],
 [1, 1, 0, 1, 1, 0, 0, 0, 0, 0]],
 [[1, 1, 0, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 0, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
 [1, 1, 0, 0, 0, 1, 0, 0, 1, 0],
 [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],
 [1, 0, 0, 1, 0, 0, 0, 0, 0, 0]]]
```

Según el modelo para el espacio de soluciones

(*) ¿Cuál es la función objetivo?

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

Respuesta

```
In [38]: #La función objetivo es el coste total de la solución
#defino el coste como la suma de los actores distintos que participan en las tomas
def coste_dia(dia):
    # Aplicar la operación AND bit a bit entre todos los elementos
    return np.sum(np.bitwise_or.reduce(dia, axis=0))
def coste_solucion(solucion):
    return np.sum(np.apply_along_axis(coste_dia, axis=1, arr=solucion))
print(coste_dia(dia1))
print(coste_solucion(solucion))

#Es un problema de minimización de la función objetivo
```

7

38

Diseña un algoritmo para resolver el problema por fuerza bruta

Respuesta

```
In [39]: from itertools import permutations
import numpy as np

#Genero todas las posibles permutaciones de los elementos del array de tomas, y luego
def generar_combinaciones_iterador(array, tomas_dia, actores_dia):
```

```
for mezcla in permutations(array,len(array)):
    yield(np.array(mezcla).reshape(len(array)//tomas_dia,tomas_dia,actores_dia))

def mejor_planificacion(tomas,tomas_por_dia,actores):
    #Inicializo las variables
    mejor_solucion=[]
    mejor_coste=math.inf
    actual=0
    #Utilizando el generador de combinaciones voy evaluando las soluciones
    for iteracion in generar_combinaciones_iterador(tomas,tomas_por_dia,actores):
        coste = coste_solucion(iteracion)
        if coste<mejor_coste:
            mejor_coste=coste
            mejor_solucion = iteracion
            print(iteracion)
            solucion = np.zeros((len(mejor_solucion), len(mejor_solucion[0])))
            for i in range(len(mejor_solucion)):
                for j in range(len(mejor_solucion[i])):
                    for k in range(len(tomas)):
                        if np.array_equal(tomas[k],mejor_solucion[i][j]):
                            if not np.any(solucion == k):
                                solucion[i][j] = k
                                break

            yield(solucion,mejor_solucion,mejor_coste,actual)
            #print(coste,mejor_coste)
            actual+=1
            #print(i)
    #Localizo los números de toma para mostrar la solución
```

```
In [40]: for solucion in (mejor_planificacion(tomas,6,10)):
    print(solucion)
```

```

[[[1 1 1 1 1 0 0 0 0 0]
 [0 0 1 1 1 0 0 0 0 0]
 [0 1 0 0 1 0 1 0 0 0]
 [1 1 0 0 0 0 1 1 0 0]
 [0 1 0 1 0 0 0 1 0 0]
 [1 1 0 1 1 0 0 0 0 0]]

[[1 1 0 1 1 0 0 0 0 0]
 [1 1 0 0 0 1 0 0 0 0]
 [1 1 0 1 0 0 0 0 0 0]
 [1 1 0 0 0 1 0 0 1 0]
 [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]
 [1 0 0 1 0 0 0 0 0 0]]]

(array([[ 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.]]), array([[[1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
        [0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 1, 0, 1, 0, 0, 0],
        [1, 1, 0, 0, 0, 1, 1, 0, 0, 0],
        [0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
        [1, 1, 0, 0, 0, 0, 1, 0, 0, 0]]])

```

```
[0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0]],

[[[1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1],
 [1, 0, 1, 0, 1, 0, 0, 0, 1, 0],
 [0, 0, 1, 1, 0, 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],
 [1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
 [1, 0, 1, 1, 0, 0, 0, 0, 0, 0]]], 38, 0)

[[[1 1 1 1 1 0 0 0 0 0]
 [0 0 1 1 1 0 0 0 0 0]
 [0 1 0 0 1 0 1 0 0 0]
 [1 1 0 0 0 0 1 1 0 0]
 [0 1 0 1 0 0 0 1 0 0]
 [1 1 0 1 1 0 0 0 0 0]]]

[[[1 1 0 1 1 0 0 0 0 0]
 [1 1 0 0 0 1 0 0 0 0]
 [1 1 0 1 0 0 0 0 0 0]
 [1 1 0 0 0 1 0 0 1 0]
 [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]
 [1 1 0 1 0 0 0 0 0 1]]]

[[[0 0 1 0 0 1 0 0 0 0]
 [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]
 [1 0 0 1 0 0 0 0 0 0]]]

(array([[ 0.,  1.,  2.,  3.,  4.,  5.],
 [ 6.,  7.,  8.,  9., 10., 11.],
 [12., 13., 14., 15., 16., 17.],
 [18., 19., 20., 21., 22., 24.],
 [23., 25., 26., 27., 28., 29.]]), array([[1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
 [0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0],
 [1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0],
 [0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0],
 [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1]]),

[[[1, 1, 0, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 0, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
 [1, 1, 0, 0, 0, 0, 1, 0, 0, 0],
 [1, 1, 1, 0, 1, 0, 0, 0, 0, 0],
 [1, 1, 1, 1, 0, 0, 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],
 [0, 1, 0, 0, 1, 0, 0, 0, 0, 0],
 [1, 0, 0, 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, 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],  
 [1, 1, 0, 1, 0, 0, 0, 0, 0, 1]],  
  
[[0, 0, 1, 0, 0, 1, 0, 0, 0, 0],  
 [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],  
 [1, 0, 0, 1, 0, 0, 0, 0, 0, 0]])), 37, 720)
```

```

-----
KeyboardInterrupt                                     Traceback (most recent call last)
<ipython-input-40-f103dde2042b> in <cell line: 1>()
----> 1 for solucion in (mejor_planificacion(tomas,6,10)):
      2     print(solucion)

<ipython-input-39-21a1cbda7c1d> in mejor_planificacion(tomas, tomas_por_dia, actores):
    14     #Utilizando el generador de combinaciones voy evaluando las soluciones
    15     for iteracion in generar_combinaciones_iterador(tomas,tomas_por_dia,actores):
----> 16         coste = coste_solucion(iteracion)
    17         if coste<mejor_coste:
    18             mejor_coste=coste

<ipython-input-38-517f7a492f2c> in coste_solucion(solucion)
    5     return np.sum(np.bitwise_or.reduce(dia, axis=0))
    6 def coste_solucion(solucion):
----> 7     return np.sum(np.apply_along_axis(coste_dia, axis=1, arr=solucion))
    8 print(coste_dia(dia1))
    9 print(coste_solucion(solucion))

/usr/local/lib/python3.10/dist-packages/numpy/lib/shape_base.py in apply_along_axis(func1d, axis, arr, *args, **kwargs)
    400     buff[ind0] = res
    401     for ind in inds:
--> 402         buff[ind] = asanyarray(func1d(inarr_view[ind], *args, **kwargs))
    403
    404     if not isinstance(res, matrix):

<ipython-input-38-517f7a492f2c> in coste_dia(dia)
    3 def coste_dia(dia):
    4     # Aplicar la operación AND bit a bit entre todos los elementos
----> 5     return np.sum(np.bitwise_or.reduce(dia, axis=0))
    6 def coste_solucion(solucion):
    7     return np.sum(np.apply_along_axis(coste_dia, axis=1, arr=solucion))

/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py in sum(a, axis, dtype, out, keepdims, initial, where)
    2311         return res
    2312
-> 2313     return _wrapreduction(a, np.add, 'sum', axis, dtype, out, keepdims=keepdims,
    2314                             initial=initial, where=where)
    2315

/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py in _wrapreduction(obj, ufunc, method, axis, dtype, out, **kwargs)
    84         return reduction(axis=axis, dtype=dtype, out=out, **passkwargs)
args)
    85     else:
--> 86         return reduction(axis=axis, out=out, **passkwargs)
    87
    88     return ufunc.reduce(obj, axis, dtype, out, **passkwargs)

/usr/local/lib/python3.10/dist-packages/numpy/core/_methods.py in _sum(a, axis, dtype, out, keepdims, initial, where)
    47 def _sum(a, axis=None, dtype=None, out=None, keepdims=False,
    48         initial=_NoValue, where=True):
----> 49     return umr_sum(a, axis, dtype, out, keepdims, initial, where)
    50
    51 def _prod(a, axis=None, dtype=None, out=None, keepdims=False,

```

KeyboardInterrupt:

Calcula la complejidad del algoritmo por fuerza bruta

Respuesta

In [41]: *#Dado que Las soluciones se van a generar a base de permutaciones del conjunto*

(*) Diseña un algoritmo que mejore la complejidad del algoritmo por fuerza bruta.

Argumenta porque crees que mejora el algoritmo por fuerza bruta

Respuesta

In [42]:

```

import random
#Pruebo una solución con búsqueda aleatoria
def generar_combinaciones_random_iterador(tomas_input, tomas_dia, actores, maximo_iteraciones):
    num_dias = len(tomas_input) // tomas_dia
    #Inicializo un array con las dimensiones de la solución
    solucion = np.empty((num_dias, tomas_dia, actores), dtype=int)
    #itero según el parámetro de iteraciones máximas
    for iteracion_actual in range(maximo_iteraciones):
        copia_array_tomas = tomas_input.copy()
        tomas_seleccionadas = 0
        #voy asignando las tomas de una en una de forma aleatoria y reduciendo
        for dia in range(num_dias):
            for toma in range(tomas_dia):
                indice = random.randint(0, len(tomas_input) - 1 - tomas_seleccionadas)
                toma_actual = copia_array_tomas[indice]
                copia_array_tomas = np.delete(copia_array_tomas, indice, axis=0)
                solucion[dia][toma] = toma_actual
                tomas_seleccionadas += 1
        #devuelvo cada solución para que se vaya mostrando el progreso
        yield(solucion)

def mejor_planificacion_random(tomas_entrada, tomas_por_dia, actores, maximo_iteraciones):
    #ahora no funciona bien, repite tomas, debe haber copiado algún array puta mierda
    #Inicializo las variables
    mejor_solucion = []
    mejor_coste = math.inf
    actual = 0
    #Utilizando el generador de combinaciones random voy evaluando las soluciones
    for iteracion in generar_combinaciones_random_iterador(tomas_entrada, tomas_por_dia):
        coste = coste_solucion(iteracion)
        if coste < mejor_coste:
            mejor_coste = coste
            mejor_solucion = iteracion
            print(iteracion)
            solucion = np.zeros((len(mejor_solucion), len(mejor_solucion[0])))
            #Localizo los números de toma para mostrar la solución
            for i in range(len(mejor_solucion)):
                for j in range(len(mejor_solucion[i])):
                    for k in range(len(tomas_entrada)):
                        if np.array_equal(tomas_entrada[k], mejor_solucion[i][j]):
                            if not np.any(solucion == k):
                                solucion[i][j] = k
                                break
            yield(solucion, mejor_solucion, mejor_coste, actual)
            #print(coste, mejor_coste)
            actual += 1
            #print(i)

#pruebo la solución con búsqueda aleatoria

```

```
for solucionx in (mejor_planificacion_random(tomas,6,10,10000)):  
    print(solucionx)
```

```

[[[1 1 0 1 0 0 0 0 0 1]
 [1 0 1 0 0 0 0 0 0 0]
 [0 0 1 1 1 0 0 0 0 0]
 [0 0 0 0 0 1 0 1 0 0]
 [1 1 0 1 0 0 0 0 0 0]
 [1 1 1 1 1 0 0 0 0 0]]

[[1 1 0 0 0 0 1 0 0 0]
 [1 1 0 1 1 0 0 0 0 0]
 [0 0 1 0 0 1 0 0 0 0]
 [1 1 1 0 1 0 0 1 0 0]
 [1 0 1 0 0 0 0 0 0 0]
 [1 1 0 0 0 1 0 0 0 0]]

[[0 1 0 0 1 0 1 0 0 0]
 [0 0 1 0 0 1 0 0 0 0]
 [0 0 0 1 1 0 0 0 0 0]
 [0 1 0 1 0 0 0 1 0 0]
 [1 1 0 0 0 1 0 0 1 0]
 [1 0 1 0 1 0 0 0 1 0]]

[[1 0 1 0 0 1 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 1]
 [1 1 0 0 0 0 1 1 0 0]
 [1 0 1 0 0 0 0 0 0 0]
 [1 1 1 1 0 0 0 0 0 0]
 [1 1 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]
 [1 0 0 1 0 0 0 0 0 0]
 [1 0 1 1 1 0 0 0 0 0]
 [1 0 0 1 1 0 0 0 0 0]
 [1 1 1 1 0 1 0 0 0 0]]]

(array([[24., 16., 1., 20., 8., 0.],
 [14., 5., 17., 10., 18., 7.],
 [2., 23., 26., 4., 9., 25.],
 [13., 15., 3., 22., 21., 6.],
 [27., 28., 29., 19., 12., 11.]]), array([[1, 1, 0, 1, 0, 0, 0, 0, 0, 1],
 [1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
 [1, 1, 0, 1, 1, 0, 0, 0, 0, 0]]], 

[[1, 1, 0, 0, 0, 0, 1, 0, 0, 0],
 [1, 1, 0, 1, 1, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 1, 0, 1, 0, 0, 1, 0, 0],
 [1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [1, 1, 0, 0, 1, 0, 0, 0, 0, 0],
 [1, 1, 1, 0, 1, 0, 0, 0, 0, 0]],

[[0, 1, 0, 0, 1, 0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
 [0, 1, 0, 1, 0, 0, 0, 1, 0, 0],
 [1, 1, 0, 0, 1, 0, 0, 0, 1, 0],
 [1, 0, 1, 0, 1, 0, 0, 0, 0, 0],
 [1, 1, 0, 1, 0, 0, 0, 0, 0, 0]],

[[1, 0, 1, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
 [1, 1, 0, 0, 0, 0, 1, 1, 0, 0],
 [1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
 [1, 1, 0, 1, 1, 0, 0, 0, 0, 0]]]

```



```

[0, 0, 0, 0, 0, 1, 0, 1, 0, 0],
[0, 0, 1, 0, 0, 1, 0, 0, 0, 0]],

[[1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
 [1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 0, 1, 0, 0, 0, 0, 0, 1],
 [1, 1, 1, 0, 1, 0, 0, 0, 0, 0]],

[[0, 0, 1, 1, 0, 0, 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, 0, 0, 0, 0, 0, 0],
 [1, 0, 1, 1, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 1, 0, 0, 0, 0, 0]]], 37, 1)

[[[1 0 1 0 0 0 0 0 0 0]
 [1 1 0 1 0 0 0 0 0 1]
 [1 1 1 1 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 1]
 [1 0 0 1 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0 0 0]]
 
 [[0 0 0 1 1 0 0 0 0 0]
 [0 0 1 0 0 1 0 0 0 0]
 [1 1 1 0 1 0 0 1 0 0]
 [1 0 0 1 0 0 0 0 0 0]
 [1 0 1 0 1 0 0 0 1 0]
 [1 1 0 0 0 0 1 1 0 0]]
 
 [[1 0 1 0 0 1 0 0 0 0]
 [0 1 0 1 0 0 0 1 0 0]
 [1 0 1 0 0 0 0 0 0 0]
 [1 0 0 0 1 1 0 0 0 0]
 [1 0 1 1 1 0 0 0 0 0]
 [1 0 0 1 1 0 0 0 0 0]]
 
 [[1 1 1 1 0 1 0 0 0 0]
 [0 0 1 0 0 1 0 0 0 0]
 [1 1 0 1 1 0 0 0 0 0]
 [0 0 1 1 1 0 0 0 0 0]
 [1 1 0 1 0 0 0 0 0 0]
 [1 1 0 0 0 1 0 0 1 0]]
 
 [[0 1 0 0 1 0 1 0 0 0]
 [1 1 0 0 0 0 1 0 0 0]
 [1 1 0 1 1 0 0 0 0 0]
 [0 0 0 0 0 1 0 1 0 0]
 [1 1 0 0 0 1 0 0 0 0]
 [1 1 1 1 1 0 0 0 0 0]]]

(array([[16., 24., 21., 15., 27., 18.],
        [26., 17., 10., 29., 25., 3.],
        [13., 4., 22., 28., 19., 12.],
        [11., 23., 5., 1., 8., 9.],
        [2., 14., 6., 20., 7., 0.]]), array([[1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
        [1, 1, 0, 1, 0, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
        [1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
        [1, 0, 1, 0, 0, 0, 0, 0, 0, 0]]),

[[0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 1, 0, 1, 0, 0, 0, 0, 0],
 [1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
 [1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 1, 1, 1, 1, 0, 0, 0, 0]]]

```

```

[1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[1, 0, 1, 0, 1, 0, 0, 0, 1, 0],
[1, 1, 0, 0, 0, 0, 1, 1, 0, 0]],

[[1, 0, 1, 0, 0, 1, 0, 0, 0, 0],
 [0, 1, 0, 1, 0, 0, 0, 1, 0, 0],
 [1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [1, 0, 0, 1, 1, 0, 0, 0, 0, 0],
 [1, 0, 1, 1, 0, 0, 0, 0, 0, 0],
 [1, 0, 0, 1, 1, 0, 0, 0, 0, 0]],

[[1, 1, 1, 0, 1, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 0, 1, 1, 0, 0, 0, 0, 0],
 [0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
 [1, 1, 0, 0, 1, 0, 0, 1, 0, 0]],

[[0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
 [1, 1, 0, 0, 0, 0, 1, 0, 0, 0],
 [1, 1, 0, 1, 1, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
 [1, 1, 0, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]], 36, 2)

[[[1 1 0 0 0 1 0 0 0 0]
 [1 1 0 1 1 0 0 0 0 0]
 [0 0 0 0 0 1 0 1 0 0]
 [1 0 1 0 0 0 0 0 0 0]
 [1 1 0 1 0 0 0 0 0 0]
 [0 0 1 0 0 1 0 0 0 0]]

[[0 1 0 1 0 0 0 1 0 0]
 [1 1 0 0 0 0 1 1 0 0]
 [1 0 1 0 0 0 0 0 0 0]
 [1 1 1 0 1 0 0 1 0 0]
 [1 1 1 1 1 0 0 0 0 0]
 [0 1 0 0 1 0 1 0 0 0]]

[[0 0 0 1 1 0 0 0 0 0]
 [1 1 0 0 0 0 1 0 0 0]
 [0 0 1 1 1 0 0 0 0 0]
 [1 0 0 1 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 1]
 [1 0 0 1 0 0 0 0 0 0]]

[[1 0 0 0 1 1 0 0 0 0]
 [0 0 1 0 0 1 0 0 0 0]
 [1 1 1 1 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0 0 0]
 [1 0 0 1 1 0 0 0 0 0]
 [1 1 0 1 0 0 0 0 0 1]]

[[1 1 0 0 0 1 0 0 1 0]
 [1 0 1 0 1 0 0 0 1 0]
 [1 1 1 1 0 1 0 0 0 0]
 [1 0 1 1 1 0 0 0 0 0]
 [1 1 0 1 1 0 0 0 0 0]
 [1 0 1 0 0 1 0 0 0 0]]]

(array([[ 7.,  5., 20., 16.,  8., 17.],
        [ 4.,  3., 18., 10.,  0.,  2.],
        [26., 14.,  1., 27., 15., 29.],
        [28., 23., 21., 22., 12., 24.],
        [ 9., 25., 11., 19.,  6., 13.]]), array([[1, 1, 0, 0, 0, 1, 0, 0, 0, 0],
        [1, 1, 0, 1, 1, 0, 0, 0, 0, 0],
        [1, 1, 0, 0, 1, 1, 0, 0, 0, 0],
        [1, 1, 0, 0, 0, 1, 1, 0, 0, 0],
        [1, 1, 0, 0, 0, 0, 1, 1, 0, 0],
        [1, 1, 0, 0, 0, 0, 0, 1, 1, 0],
        [1, 1, 0, 0, 0, 0, 0, 0, 1, 1],
        [1, 1, 0, 0, 0, 0, 0, 0, 0, 1]]),

```

```

[0, 0, 0, 0, 0, 1, 0, 1, 0, 0],
[1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 1, 0, 0, 0, 0]],

[[0, 1, 0, 1, 0, 0, 0, 1, 0, 0],
[1, 1, 0, 0, 0, 0, 1, 1, 0, 0],
[1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[1, 1, 1, 0, 1, 0, 0, 1, 0, 0],
[1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 1, 0, 1, 0, 0, 0]],

[[0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
[1, 1, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
[1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
[1, 0, 0, 1, 0, 0, 0, 0, 0, 0]],

[[1, 0, 0, 1, 1, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
[1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
[1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 1, 1, 0, 0, 0, 0, 0],
[1, 1, 0, 1, 0, 0, 0, 0, 0, 1]],

[[1, 1, 0, 0, 0, 1, 0, 0, 1, 0],
[1, 0, 1, 0, 1, 0, 0, 0, 1, 0],
[1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
[1, 0, 1, 1, 1, 0, 0, 0, 0, 0],
[1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
[1, 0, 1, 0, 0, 1, 0, 0, 0, 0]], 35, 17)

[[[1 1 1 1 0 0 0 0 0 0]
[1 1 0 1 1 0 0 0 0 0]
[1 1 1 0 1 0 0 1 0 0]
[1 0 0 1 1 0 0 0 0 0]
[0 0 0 1 1 0 0 0 0 0]
[0 1 0 1 0 0 0 1 0 0]]

[[0 0 0 1 0 0 0 0 0 1]
[0 0 1 1 1 0 0 0 0 0]
[1 1 0 0 0 1 0 0 1 0]
[1 0 1 0 0 0 0 0 0 0]
[0 0 1 0 0 1 0 0 0 0]
[1 1 0 0 0 1 0 0 0 0]]

[[1 1 0 1 0 0 0 0 0 0]
[1 1 0 0 0 0 1 0 0 0]
[1 1 0 1 0 0 0 0 0 1]
[1 1 0 1 1 0 0 0 0 0]
[0 1 0 0 1 0 1 0 0 0]
[1 1 0 0 0 0 1 1 0 0]]

[[0 0 0 0 0 1 0 1 0 0]
[1 0 1 0 0 1 0 0 0 0]
[1 0 0 1 0 0 0 0 0 0]
[1 0 0 0 1 1 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 0 1 0 0 0 1 0]
[1 0 1 0 0 0 0 0 0 0]
[1 1 1 1 1 0 0 0 0 0]
[1 1 1 1 0 1 0 0 0 0]]]
```

```

[1 0 1 1 1 0 0 0 0 0]
[1 0 0 1 0 0 0 0 0 0]]]
(array([[21., 5., 10., 12., 26., 4.],
       [15., 1., 9., 16., 17., 7.],
       [8., 14., 24., 6., 2., 3.],
       [20., 13., 27., 28., 23., 18.],
       [25., 22., 0., 11., 19., 29.]]], array([[[1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
          [1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
          [1, 1, 1, 0, 1, 0, 0, 1, 0, 0],
          [1, 0, 0, 1, 1, 0, 0, 0, 0, 0],
          [0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
          [0, 1, 0, 1, 0, 0, 1, 0, 0, 0]],

         [[0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
          [0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
          [1, 1, 0, 0, 0, 1, 0, 0, 1, 0],
          [1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
          [1, 1, 0, 0, 0, 1, 0, 0, 0, 0]],

         [[1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
          [1, 1, 0, 0, 0, 0, 1, 0, 0, 0],
          [1, 1, 0, 1, 0, 0, 0, 0, 0, 1],
          [1, 1, 0, 1, 1, 0, 0, 0, 0, 0],
          [0, 1, 0, 0, 1, 0, 1, 0, 0, 0],
          [1, 1, 0, 0, 0, 0, 1, 1, 0, 0]],

         [[0, 0, 0, 0, 0, 1, 0, 1, 0, 0],
          [1, 0, 1, 0, 0, 1, 0, 0, 0, 0],
          [1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
          [1, 0, 0, 0, 1, 1, 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, 0, 1, 0, 0, 0, 1, 0],
          [1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
          [1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
          [1, 1, 1, 0, 1, 0, 0, 1, 0, 0],
          [0, 0, 0, 1, 1, 0, 0, 0, 0, 0]]], 34, 62)

[[[1 0 0 1 0 0 0 0 0 0]
  [1 0 0 1 0 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 1 1 0 1 0 0 1 0 0]
  [0 0 0 1 1 0 0 0 0 0]]]

[[1 0 0 1 1 0 0 0 0 0]
  [0 0 1 0 0 1 0 0 0 0]
  [1 0 1 0 1 0 0 0 1 0]
  [1 1 0 1 0 0 0 0 0 0]
  [1 1 1 1 0 1 0 0 0 0]
  [1 0 0 0 1 1 0 0 0 0]]]

[[1 0 1 0 0 1 0 0 0 0]
  [1 0 1 0 0 0 0 0 0 0]
  [1 1 0 0 0 0 1 0 0 0]
  [0 0 1 0 0 1 0 0 0 0]
  [1 1 0 0 0 1 0 0 1 0]
  [1 0 1 0 0 0 0 0 0 0]]]

[[0 0 0 1 0 0 0 0 0 1]
  [1 1 0 0 0 1 0 0 0 0]
  [1 1 0 1 0 0 0 0 0 1]]

```

```
[0 1 0 1 0 0 0 1 0 0]
[0 1 0 0 1 0 1 0 0 0]
[1 1 0 0 0 0 1 1 0 0]

[[1 0 1 0 0 0 0 0 0 0]
 [1 0 1 1 1 0 0 0 0 0]
 [1 1 1 1 1 0 0 0 0 0]
 [1 1 0 1 1 0 0 0 0 0]
 [0 0 1 1 1 0 0 0 0 0]
 [1 1 0 1 1 0 0 0 0 0]]

(array([[27., 29., 20., 21., 10., 26.],
       [12., 17., 25., 8., 11., 28.],
       [13., 16., 14., 23., 9., 18.],
       [15., 7., 24., 4., 2., 3.],
       [22., 19., 0., 5., 1., 6.]]), array([[[1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                                                [1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                                                [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
                                                [1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
                                                [1, 1, 0, 1, 1, 0, 0, 0, 0, 0],
                                                [0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
                                                [1, 1, 0, 1, 1, 0, 0, 0, 0, 0]]],

[[1, 0, 0, 1, 1, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
 [1, 0, 1, 0, 1, 0, 0, 0, 1, 0],
 [1, 1, 0, 1, 0, 0, 0, 0, 0, 0],
 [1, 1, 1, 1, 0, 0, 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, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [1, 1, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 0, 0, 1, 0, 0, 0, 1, 0],
 [1, 0, 0, 1, 1, 0, 0, 0, 0, 0]],

[[1, 0, 1, 0, 0, 0, 1, 0, 0, 0],
 [1, 1, 0, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 0, 0, 1, 0, 0, 0, 1, 0],
 [1, 0, 0, 1, 1, 0, 0, 0, 0, 0]],

[[0, 0, 1, 0, 0, 0, 0, 0, 0, 1],
 [1, 1, 0, 0, 0, 1, 0, 0, 0, 0],
 [1, 1, 0, 1, 0, 0, 0, 0, 0, 1],
 [0, 1, 0, 1, 0, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 1, 0, 1, 0, 0, 0],
 [1, 1, 0, 0, 0, 1, 1, 0, 0, 0]],

[[1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [1, 0, 1, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 0, 1, 1, 0, 0, 0, 0, 0],
 [0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 0, 1, 1, 0, 0, 0, 0, 0]]], 33, 818)
```

(*)Calcula la complejidad del algoritmo

Con esta búsqueda aleatoria la complejidad es del orden del parámetro maximo_iteraciones multiplicado por una constante relativa al número de operaciones de cada iteración

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

Respuesta

```
In [43]: import numpy as np
def tomas_random(tomas_in, actores):
    #Creo un array vacío
    array = np.empty((tomas_in, actores), dtype=int)
```

```
#Genero números aleatorios entre 0 y 1 para cada posición del array
aleatorios = np.random.rand(tomas_in, actores)

#Relleno el array con 1 si el número aleatorio es mayor o igual a 0.5, y con 0
array[aleatorios >= 0.5] = 1
array[aleatorios < 0.5] = 0
print(array)
return(array)
```

Aplica el algoritmo al juego de datos generado

Respuesta

```
In [52]: num_tomas=8
actores=5
tomas_por_dia=2
tomas_rand=tomas_random(num_tomas,actores)
#planificacion_dias, detalle_dias, coste=
for parcial in mejor_planificacion(tomas_rand,tomas_por_dia,actores):
    print(parcial)
for _ in range(10):
    print('#####')
for parcial in mejor_planificacion_random(tomas_rand,tomas_por_dia,actores,10000):
    print(parcial)
#print('Tomas por día de rodaje:\n',planificacion_dias, '\nActores por día\n',detalle_dias)
```

```

[[[0 0 0 0 0]
 [1 0 0 1 0]
 [0 1 1 1 1]
 [0 1 1 0 0]
 [0 0 1 0 0]
 [1 0 1 1 0]
 [1 0 0 1 0]
 [0 1 0 1 0]]
 [[[0 0 0 0 0]
 [1 0 0 1 0]]]

[[[0 1 1 1 1]
 [0 1 1 0 0]]
 [[[0 0 1 0 0]
 [1 0 1 1 0]]]

[[[1 0 0 1 0]
 [0 1 0 1 0]]]
(array([[0., 1.],
 [2., 3.],
 [4., 5.],
 [6., 7.]]), array([[[0, 0, 0, 0, 0],
 [1, 0, 0, 1, 0]],
 [[0, 1, 1, 1, 1],
 [0, 1, 1, 0, 0]],
 [[1, 0, 1, 1, 0],
 [1, 0, 0, 1, 0]]]),
 [[0, 1, 1, 1, 1],
 [0, 1, 1, 0, 0]],
 [[1, 0, 1, 1, 0],
 [1, 0, 0, 1, 0]]],
 [[0, 0, 1, 0, 0],
 [1, 0, 1, 1, 0]]], 12, 0)
[[[0 0 0 0 0]
 [1 0 0 1 0]]]

[[[0 1 1 1 1]
 [0 1 0 1 0]]
 [[[0 1 1 0 0]
 [0 0 1 0 0]]]

[[[1 0 1 1 0]
 [1 0 0 1 0]]]
(array([[0., 1.],
 [2., 7.],
 [3., 4.],
 [5., 6.]]), array([[[0, 0, 0, 0, 0],
 [1, 0, 0, 1, 0]],
 [[0, 1, 1, 1, 1],
 [0, 1, 0, 1, 0]],
 [[1, 0, 1, 1, 0],
 [1, 0, 0, 1, 0]]]),
 [[0, 1, 1, 1, 1],
 [0, 1, 0, 1, 0]],
 [[1, 0, 1, 1, 0],
 [1, 0, 0, 1, 0]]],
 [[0, 1, 1, 0, 0],
 [0, 0, 1, 0, 0]],
 [[1, 0, 1, 1, 0],
 [1, 0, 0, 1, 0]]], 11, 96)
#####
#####
```

```
#####
#####
#####
#####
[[[1 0 0 1 0]
 [1 0 1 1 0]]

[[0 1 1 0 0]
 [0 1 1 1 1]]

[[1 0 0 1 0]
 [0 0 1 0 0]]

[[0 1 0 1 0]
 [0 0 0 0 0]]
(array([[1., 5.],
 [3., 2.],
 [6., 4.],
 [7., 0.]]), array([[1, 0, 0, 1, 0],
 [1, 0, 1, 1, 0]],

 [[0, 1, 1, 0, 0],
 [0, 1, 1, 1, 1]],

 [[1, 0, 0, 1, 0],
 [0, 0, 1, 0, 0]],

 [[0, 1, 0, 1, 0],
 [0, 0, 0, 0, 0]]], 12, 0)
[[[0 1 1 1 1]
 [0 1 0 1 0]]

[[1 0 0 1 0]
 [1 0 0 1 0]]

[[0 0 1 0 0]
 [0 0 0 0 0]]

[[1 0 1 1 0]
 [0 1 1 0 0]]
(array([[2., 7.],
 [1., 6.],
 [4., 0.],
 [5., 3.]]), array([[0, 1, 1, 1, 1],
 [0, 1, 0, 1, 0]],

 [[1, 0, 0, 1, 0],
 [1, 0, 0, 1, 0]],

 [[0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0]],

 [[1, 0, 1, 1, 0],
 [0, 1, 1, 0, 0]]], 11, 1)
```

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

Documentación de python sobre itertools: <https://docs.python.org/3/library/itertools.html>

Describe brevemente las líneas 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

Para optimizar un problema de planificación como el de este trabajo, se tendría que buscar una función de selección de las mejores soluciones. En este caso particular, una posible mejora sería tratar de agrupar en un mismo día tomas donde se repitan el máximo número de actores. Una implementación podría ser seleccionar una toma para cada día y luego recorrer el listado clasificando las tomas en el día con mayor coincidencia de actores.