Algoritmos de optimización - Seminario

Nombre y Apellidos: Daniel Paniagua Ares
Url: https://github.com/dpaniaguaa/algoritmos_optimizacion_MIAR_2024
Problema:

> 1. Sesiones de doblaje

Descripción del problema:(copiar enunciado) 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.

Nº actores = 10

Nº tomas = 30

(*) La respuesta es obligatoria

In [ ]:
```
import math
import numpy as np
import random
import copy
from itertools import combinations, chain
```

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

Si no hay ninguna restricción cada toma se puede asignar a uno de los días de grabación. Si no imponemos un límite en el número de tomas por día, podemos considerar que cada toma se puede grabar en un día independiente. Entonces, el número total de maneras en que se pueden organizar estas tomas es una permutación de n tomas.

En este caso al contar con 30 tomas las posibilidades son

In [ ]:
```
n_pos = math.factorial(30)

print(f"El número de posibilidades es: {n_pos}")
```

```
El número de posibilidades es: 265252859812191058636308480000000
```

¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.

Al tener en cuenta que sólo se pueden grabar 6 tomas por día eso reduce la manera de configruar cada día. Las posiblidades entonces son las posiblidades de cada día multiplicadas. Y cada día es una variación sin repetición por lo que:

In [ ]:
```
n_pos = 1

for i in reversed(range(6,36,6)):
  n_pos = n_pos * (math.factorial(i)/(math.factorial(6)*math.factorial(i-6)))

print(f"El número de posibilidades es: {int(n_pos)}")
```

```
El número de posibilidades es: 1370874167589326336
```

Modelo para el espacio de soluciones
(*) ¿Cual 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, arguentalo)

La mejor estructura es un array donde las columnas indiquen el actor y las filas la toma. De esta manera cada cela indica en qué toma es necesario qué actor/es. (Nótese que se ha hecho un array más pequeño para realizar pruebas sobre el algoritmo de fuerza bruta)

In [ ]:
```
# A C T O R E S
# T
# O
# M
# A
# S

data = np.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],
        [1,1,0,0,0,0,1,1,0,0],
        [0,1,0,1,0,0,0,1,0,0],
        [1,1,0,1,1,0,0,0,0,0],
        [1,1,0,1,1,1,0,0,0,0],
        [1,1,0,0,0,1,0,0,0,0],
        [1,1,0,1,0,0,0,0,0,0],
        [1,1,0,0,0,1,0,0,1,0],
        [1,1,1,1,0,1,0,0,1,0],
        [1,1,1,1,0,1,1,0,0,0],
        [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],
        ])

data_small = np.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],
        [1,1,0,0,0,0,1,1,0,0],
        [0,1,0,1,0,0,0,1,0,0],
        [1,1,0,1,1,0,0,0,0,0],
        [1,1,0,1,1,1,0,0,0,0],
        [1,1,0,0,0,1,0,0,0,0],
        [1,1,0,1,0,0,0,0,0,0],
        [1,1,0,0,0,1,0,0,1,0],
        [1,1,1,1,0,1,0,0,1,0],
        [1,1,1,1,0,1,1,0,0,0],

        ])
```

Según el modelo para el espacio de soluciones
(*)¿Cual es la función objetivo?

La función objetivo es el número de traslados de actores por día.

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

Es un problema de minimización, ya que lo que se busca es reducir el número de traslado de actores por día para pagar menos desplazamientos.

Diseña un algoritmo para resolver el problema por fuerza bruta

Respuesta

In [ ]:
```
# Función para calcular el número de actores necesarios para un conjunto de tomas
def actores_necesarios(tomas, data):
    if len(tomas) == 0:
        return 0
    return np.any(data[list(tomas)], axis=0).sum()

# Generar todas las combinaciones posibles de tomas por día
def generar_combinaciones(data, max_tomas_por_dia=6):
    indices = range(data.shape[0])
    todas_combinaciones = []
    for r in range(1, max_tomas_por_dia + 1):
        todas_combinaciones.extend(combinations(indices, r))
    return todas_combinaciones

# Evaluar todas las combinaciones posibles y encontrar la mejor
```

```
def encontrar_mejor_combinacion(data, n_tomas):
    mejor_combinacion = None
    min_actores = float('inf')
    todas_combinaciones = generar_combinaciones(data)

    # Generar todas las combinaciones posibles de grupos de tomas por día
    for combinacion in combinations(todas_combinaciones, (n_tomas + 5) // 6):
        # Verificar que todas las tomas estén cubiertas
        tomas_cubiertas = set(chain(*combinacion))
        if len(tomas_cubiertas) == n_tomas:
            actores = sum(actores_necesarios(tomas, data) for tomas in combinacion)

            if actores < min_actores:
                min_actores = actores
                mejor_combinacion = combinacion

    return mejor_combinacion, min_actores

# Número de tomas
n_tomas = data_small.shape[0]

mejor_combinacion, min_actores = encontrar_mejor_combinacion(data_small, n_tomas)
print("Mejor combinación de tomas por día:", mejor_combinacion)
print("Número mínimo de desplazamientos de actores necesarios:", min_actores)
```

```
Mejor combinación de tomas por día: ((0, 1, 2, 3, 4, 10), (5, 6, 7, 8, 9, 11))
Número mínimo de desplazamientos de actores necesarios: 14
```

Calcula la complejidad del algoritmo por fuerza bruta

La complejidad del algoritmo es cúbica ya que tiene 3 bucles anidados, el primero para iterar sobre las combinaciones, el segundo para iterar sobre las tomas dentro de las combinaciones, y el 3ro dentro de la función any de numpy que itera sobre el array.

o(n^3)

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

Respuesta

In [ ]:

```
# Función para calcular el número de actores necesarios para un conjunto de tomas
def calcular_actores_necesarios(solucion,data):
    return sum(actores_necesarios(tomas, data) for tomas in solucion)

def tomas_cubiertas(solucion):
    # Verificar si una solución cubre todas las tomas
    tomas_cubiertas = set()

    for dia in solucion:
        tomas_cubiertas.update(dia)

    return len(tomas_cubiertas)


def actores_necesarios(tomas, data):
    if len(tomas) == 0:
        return 0
    return np.any(data[list(tomas)], axis=0).sum()


def generar_solucion_aleatoria(data):
  # Número total de tomas
  n_tomas = data.shape[0]

  # Crear una lista de todas las tomas
  todas_las_tomas = list(range(n_tomas))

  # Barajar la lista de tomas aleatoriamente
  random.shuffle(todas_las_tomas)

  # Dividir las tomas en grupos de hasta 6 tomas por día
  solucion = [todas_las_tomas[i:i+6] for i in range(0, n_tomas, 6)]

  return solucion


def generar_poblacion(data,N):
  return [generar_solucion_aleatoria(data) for _ in range(N)]

def evaluar_poblacion(poblacion):
    mejor_solucion = None
    min_actores = float('inf')
    max_tomas = 0

    for solucion in poblacion:
        tomas_cubiertas_ = tomas_cubiertas(solucion)
        if tomas_cubiertas_ > max_tomas:
            actores = calcular_actores_necesarios(solucion,data)
            if actores < min_actores:
                max_tomas = tomas_cubiertas_
                min_actores = actores
                mejor_solucion = solucion

    return (mejor_solucion, min_actores, max_tomas)


def crossover(sol1, sol2):
    # Punto de corte aleatorio
    punto_corte = random.randint(1, min(len(sol1), len(sol2)) - 1)

    # Generar hijos cruzando los padres en el punto de corte
    hijo1 = sol1[:punto_corte] + sol2[punto_corte:]
    hijo2 = sol2[:punto_corte] + sol1[punto_corte:]

    return hijo1, hijo2

def mutar(solucion, factor_mutacion, n_tomas):
    solucion_mutada = solucion.copy()

    for dia in solucion_mutada:
        if random.random() < factor_mutacion:
            # Seleccionar una toma aleatoria dentro del día
            toma_a_remover = random.choice(dia)
            dia.remove(toma_a_remover)

            # Agregar una nueva toma aleatoria que no esté ya en el día
            nueva_toma = random.choice(list(set(range(n_tomas)) - set(dia)))
            dia.append(nueva_toma)

    return solucion_mutada

def generar_nuevas_soluciones(sol1, sol2, factor_mutacion, n_tomas):
    # Generar soluciones cruzadas
    hijo1, hijo2 = crossover(sol1, sol2)

    # Aplicar mutación a los hijos
    hijo_mutado1 = mutar(hijo1, factor_mutacion, n_tomas)
    hijo_mutado2 = mutar(hijo2, factor_mutacion, n_tomas)

    return [hijo1, hijo2, hijo_mutado1, hijo_mutado2]


def cruzar(poblacion, mutacion):
  poblacion_copia = copy.deepcopy(poblacion)

  poblacion_final =  []

  while len(poblacion_copia) > 1:
    padre1,padre2  = random.sample(poblacion_copia,2)
    poblacion_copia.remove(padre1)
    poblacion_copia.remove(padre2)

    poblacion_final.extend(generar_nuevas_soluciones(padre1, padre2, mutacion, n_tomas))


  return poblacion_final

def seleccionar(poblacion,N, elitismo):
  poblacion_ordenada= sorted(poblacion, key=lambda sol: (-tomas_cubiertas(sol), calcular_actores_necesarios(sol, data)))

  n_elements = int(N*elitismo)
```

```
  nueva_poblacion = poblacion_ordenada[:n_elements]

  for i in range(n_elements, N, 1):
    nueva_poblacion.extend(random.sample(poblacion_ordenada,1))
  return nueva_poblacion

def algoritmo_genetico(data,N=100,mutacion=.15,elitismo=.1,generaciones=100):

  #Genera la poblacion inicial
  poblacion = generar_poblacion(data,N)

  #Inicializamos valores para la mejor solucion
  (solucion, n_actores, n_tomas_cubiertas) = evaluar_poblacion(poblacion)


  #Condicion de parada
  parar = False
  n=0
  #Iniciamos el cliclo de generaciones
  while(parar == False) :

    #Cruce de la poblacion(incluye mutación)
    poblacion = cruzar(poblacion, mutacion)

    #Seleccionamos la población
    poblacion = seleccionar(poblacion, N, elitismo)

    #Evaluamos la nueva población
    (mejor_solucion, actores_min, tomas_cubiertas) = evaluar_poblacion(poblacion)

    if actores_min <= n_actores and tomas_cubiertas >= n_tomas_cubiertas:
        n_tomas_cubiertas = tomas_cubiertas
        n_actores = actores_min
        solucion = mejor_solucion

    print("Generacion #", n, "\nLa mejor solución es:" , mejor_solucion, "\ncon " , actores_min, " traslados de actores necesarios y cubre ",tomas_cubiertas, " tomas\n")

    #Numero de generaciones. Criterio de parada
    if n==generaciones:
      parar = True
    n +=1

  print("La mejor solucion encontrada ha sido: ",solucion)
  print("La cual necesita ", n_actores, " desplazamientos y cubre ", n_tomas_cubiertas," tomas")

  return solucion

algoritmo_genetico(data,N=1000,mutacion=0.2,elitismo=0.2,generaciones=100)
```

```
Generacion # 0
La mejor solución es: [[19, 13, 11, 3, 21, 22], [9, 15, 1, 0, 14, 23], [17, 28, 26, 25, 7, 12], [10, 8, 6, 16, 18, 29], [27, 4, 19, 2, 13, 24]]
con  39  traslados de actores necesarios y cubre  28  tomas

Generacion # 1
La mejor solución es: [[9, 7, 27, 8, 1, 0], [17, 25, 19, 26, 22, 28], [5, 12, 23, 24, 16, 20], [2, 11, 3, 9, 14, 8], [29, 13, 6, 7, 18, 15]]
con  37  traslados de actores necesarios y cubre  27  tomas

Generacion # 2
La mejor solución es: [[1, 7, 3, 0, 24, 19], [25, 9, 29, 5, 11, 6], [27, 12, 18, 20, 2, 8], [13, 8, 28, 22, 26, 11], [15, 27, 21, 23, 24, 10]]
con  38  traslados de actores necesarios y cubre  26  tomas

Generacion # 3
La mejor solución es: [[14, 25, 5, 11, 28, 4], [26, 15, 16, 23, 5, 8], [12, 6, 13, 27, 10, 7], [20, 17, 2, 1, 4, 9], [16, 22, 18, 24, 29, 0]]
con  38  traslados de actores necesarios y cubre  27  tomas

Generacion # 4
La mejor solución es: [[26, 14, 15, 18, 3, 16], [28, 25, 11, 0, 23, 2], [12, 6, 13, 27, 10, 7], [20, 17, 1, 8, 7, 10], [24, 19, 1, 10, 4, 8]]
con  37  traslados de actores necesarios y cubre  25  tomas

Generacion # 5
La mejor solución es: [[23, 27, 25, 17, 26, 18], [29, 8, 16, 9, 2, 4], [24, 18, 5, 21, 10, 0], [23, 11, 27, 7, 0, 6], [22, 10, 6, 1, 3, 11]]
con  36  traslados de actores necesarios y cubre  23  tomas

Generacion # 6
La mejor solución es: [[19, 24, 21, 9, 11, 8], [16, 12, 20, 0, 2, 6], [14, 4, 3, 10, 7, 5], [17, 28, 25, 0, 1, 10], [9, 23, 8, 6, 1, 0]]
con  39  traslados de actores necesarios y cubre  23  tomas

Generacion # 7
La mejor solución es: [[7, 22, 20, 8, 0, 3], [1, 17, 5, 11, 8, 0], [27, 19, 28, 2, 6, 10], [12, 23, 21, 5, 3, 7], [16, 29, 9, 5, 3, 7]]
con  39  traslados de actores necesarios y cubre  22  tomas

Generacion # 8
La mejor solución es: [[17, 5, 20, 10, 27, 6], [16, 18, 4, 8, 7, 10], [28, 22, 6, 0, 5, 10], [2, 12, 3, 11, 6, 7], [9, 19, 12, 1, 21, 5]]
con  36  traslados de actores necesarios y cubre  22  tomas

Generacion # 9
La mejor solución es: [[27, 0, 6, 1, 8, 5], [18, 6, 29, 7, 3, 1], [7, 2, 1, 4, 5, 3], [8, 26, 11, 0, 6, 2], [19, 10, 13, 22, 17, 9]]
con  36  traslados de actores necesarios y cubre  20  tomas

Generacion # 10
La mejor solución es: [[7, 27, 28, 4, 9, 6], [29, 20, 4, 0, 3, 2], [14, 17, 12, 5, 10, 0], [11, 1, 8, 2, 5, 10], [0, 5, 6, 8, 1, 10]]
con  37  traslados de actores necesarios y cubre  19  tomas

Generacion # 11
La mejor solución es: [[26, 10, 8, 0, 6, 4], [25, 14, 23, 9, 5, 3], [5, 1, 10, 11, 0, 2], [0, 20, 4, 11, 2, 8], [17, 5, 16, 7, 8, 6]]
con  37  traslados de actores necesarios y cubre  19  tomas

Generacion # 12
La mejor solución es: [[8, 6, 7, 1, 4, 3], [4, 24, 17, 9, 11, 10], [23, 2, 7, 0, 10, 11], [19, 21, 26, 7, 0, 5], [20, 1, 0, 5, 7, 6]]
con  38  traslados de actores necesarios y cubre  19  tomas

Generacion # 13
La mejor solución es: [[14, 22, 1, 11, 9, 7], [8, 12, 3, 6, 7, 2], [11, 8, 3, 1, 2, 7], [21, 1, 8, 10, 4, 5], [20, 1, 0, 4, 7, 10]]
con  36  traslados de actores necesarios y cubre  17  tomas

Generacion # 14
La mejor solución es: [[14, 22, 1, 11, 7, 2], [8, 12, 3, 7, 9, 11], [28, 3, 7, 10, 4, 8], [21, 8, 10, 7, 3, 6], [0, 8, 11, 5, 6, 10]]
con  39  traslados de actores necesarios y cubre  17  tomas

Generacion # 15
La mejor solución es: [[14, 11, 7, 3, 9, 6], [12, 3, 9, 11, 6, 8], [3, 1, 4, 5, 2, 6], [25, 8, 28, 1, 3, 10], [4, 11, 0, 8, 7, 5]]
con  41  traslados de actores necesarios y cubre  16  tomas

Generacion # 16
La mejor solución es: [[26, 11, 7, 1, 3, 5], [11, 0, 7, 10, 4, 3], [8, 19, 6, 10, 11, 9], [24, 3, 11, 5, 9, 7], [6, 8, 4, 10, 1, 2]]
con  41  traslados de actores necesarios y cubre  15  tomas

Generacion # 17
La mejor solución es: [[25, 7, 8, 2, 5, 0], [4, 3, 10, 1, 0, 5], [24, 0, 9, 11, 1, 3], [1, 8, 5, 9, 6, 2], [18, 8, 9, 4, 3, 7]]
con  41  traslados de actores necesarios y cubre  15  tomas

Generacion # 18
La mejor solución es: [[7, 2, 5, 0, 1, 8], [4, 3, 10, 8, 0, 1], [24, 0, 3, 8, 6, 1], [9, 1, 0, 11, 5, 2], [18, 9, 1, 5, 10, 8]]
con  38  traslados de actores necesarios y cubre  14  tomas

Generacion # 19
La mejor solución es: [[17, 3, 9, 8, 5, 6], [4, 7, 10, 3, 11, 2], [24, 0, 7, 10, 8, 4], [9, 0, 6, 10, 5, 2], [1, 5, 7, 11, 4, 10]]
con  41  traslados de actores necesarios y cubre  14  tomas

Generacion # 20
La mejor solución es: [[8, 5, 10, 3, 0, 1], [1, 4, 11, 6, 2, 0], [10, 5, 8, 4, 7, 9], [7, 20, 5, 6, 3, 1], [1, 5, 7, 11, 4, 10]]
con  38  traslados de actores necesarios y cubre  13  tomas

Generacion # 21
La mejor solución es: [[7, 6, 2, 1, 8, 0], [11, 6, 7, 4, 1, 10], [29, 2, 5, 1, 10, 7], [9, 7, 5, 6, 0, 1], [2, 5, 0, 6, 4, 3]]
con  36  traslados de actores necesarios y cubre  13  tomas

Generacion # 22
La mejor solución es: [[7, 11, 5, 4, 8, 6], [5, 8, 11, 0, 7, 10], [20, 6, 3, 10, 5, 2], [7, 2, 8, 1, 11, 9], [1, 7, 0, 11, 3, 2]]
con  38  traslados de actores necesarios y cubre  13  tomas

Generacion # 23
La mejor solución es: [[0, 7, 8, 1, 5, 9], [5, 1, 2, 7, 4, 0], [8, 11, 7, 9, 5, 6], [4, 8, 6, 1, 10, 0], [11, 3, 4, 6, 2, 0]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 24
La mejor solución es: [[1, 0, 7, 6, 8, 5], [1, 9, 10, 3, 7, 6], [6, 8, 10, 2, 4, 1], [7, 1, 5, 11, 6, 0], [5, 1, 2, 0, 3, 10]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 25
La mejor solución es: [[7, 8, 10, 2, 4, 5], [5, 1, 2, 7, 0, 8], [11, 0, 5, 1, 9, 8], [11, 1, 6, 5, 7, 0], [4, 0, 1, 10, 3, 5]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 26
La mejor solución es: [[7, 0, 2, 3, 8, 1], [10, 0, 8, 5, 2, 4], [2, 1, 0, 11, 4, 10], [6, 2, 5, 7, 8, 9], [8, 11, 7, 5, 6, 0]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 27
La mejor solución es: [[8, 6, 3, 5, 7, 0], [6, 8, 7, 3, 4, 2], [0, 9, 11, 8, 1, 7], [10, 1, 6, 8, 3, 2], [3, 0, 8, 10, 2, 5]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 28
La mejor solución es: [[11, 8, 4, 6, 0, 7], [1, 8, 2, 10, 5, 3], [0, 9, 8, 1, 7, 6], [3, 2, 8, 4, 11, 7], [8, 4, 1, 10, 2, 5]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 29
La mejor solución es: [[11, 5, 4, 7, 1, 6], [6, 4, 1, 3, 5, 2], [9, 10, 5, 1, 11, 0], [6, 1, 4, 0, 8, 2], [4, 2, 3, 6, 5, 8]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 30
La mejor solución es: [[5, 2, 7, 8, 1, 10], [4, 0, 8, 6, 10, 5], [5, 11, 0, 8, 10, 4], [5, 6, 7, 1, 8, 3], [1, 11, 9, 7, 5, 0]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 31
La mejor solución es: [[0, 1, 7, 11, 6, 8], [1, 6, 4, 10, 8, 0], [1, 6, 2, 5, 3, 10], [0, 6, 9, 4, 1, 11], [7, 6, 8, 9, 10, 11]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 32
La mejor solución es: [[10, 0, 3, 8, 1, 6], [9, 8, 5, 4, 7, 6], [1, 10, 4, 2, 8, 5], [4, 6, 1, 10, 8, 5], [8, 3, 1, 11, 0, 4]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 33
La mejor solución es: [[10, 8, 1, 5, 4, 7], [0, 10, 6, 3, 2, 5], [7, 2, 5, 11, 1, 8], [10, 11, 6, 8, 9, 0], [6, 3, 8, 5, 2, 7]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 34
La mejor solución es: [[8, 0, 5, 1, 2, 10], [0, 8, 5, 4, 7, 6], [0, 9, 5, 7, 6, 8], [0, 10, 7, 6, 3, 11], [3, 8, 11, 5, 0, 10]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 35
La mejor solución es: [[5, 8, 7, 10, 0, 11], [0, 11, 1, 4, 7, 5], [1, 4, 0, 10, 6, 8], [6, 3, 0, 1, 5, 2], [9, 6, 4, 10, 2, 8]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 36
La mejor solución es: [[4, 5, 6, 1, 8, 0], [3, 2, 5, 7, 6, 9], [1, 8, 6, 4, 3, 10], [10, 8, 7, 11, 0, 5], [2, 6, 0, 1, 3, 8]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 37
La mejor solución es: [[4, 1, 0, 11, 9, 6], [6, 11, 10, 7, 1, 8], [10, 4, 5, 11, 6, 7], [4, 2, 5, 3, 0, 1], [0, 8, 5, 7, 10, 1]]
con  36  traslados de actores necesarios y cubre  12  tomas
```

```
Generacion # 38
La mejor solución es: [[8, 5, 6, 2, 10, 4], [0, 4, 5, 6, 1, 7], [0, 11, 1, 6, 8, 7], [10, 2, 11, 6, 3, 9], [8, 3, 2, 5, 0, 1]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 39
La mejor solución es: [[8, 4, 11, 6, 9, 0], [7, 1, 5, 2, 0, 8], [2, 1, 3, 6, 8, 5], [8, 6, 1, 0, 11, 5], [0, 6, 3, 5, 1, 10]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 40
La mejor solución es: [[10, 11, 3, 4, 0, 8], [6, 5, 2, 10, 4, 8], [5, 8, 9, 2, 6, 7], [5, 0, 9, 1, 6, 7], [3, 8, 6, 1, 2, 5]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 41
La mejor solución es: [[2, 6, 10, 5, 3, 4], [1, 5, 9, 8, 6, 7], [6, 5, 8, 11, 7, 4], [4, 8, 7, 0, 5, 6], [7, 5, 2, 0, 1, 10]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 42
La mejor solución es: [[3, 6, 2, 8, 4, 5], [8, 6, 1, 9, 10, 7], [8, 6, 0, 1, 3, 5], [10, 6, 5, 0, 4, 1], [0, 11, 6, 1, 2, 7]]
con  34  traslados de actores necesarios y cubre  12  tomas

Generacion # 43
La mejor solución es: [[7, 11, 6, 0, 1, 9], [0, 7, 6, 11, 4, 9], [11, 8, 4, 6, 5, 10], [5, 0, 7, 4, 1, 6], [6, 5, 2, 0, 3, 10]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 44
La mejor solución es: [[8, 2, 6, 7, 5, 1], [5, 0, 6, 1, 10, 7], [1, 5, 11, 10, 0, 9], [8, 2, 3, 7, 5, 6], [0, 8, 5, 4, 1, 6]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 45
La mejor solución es: [[0, 7, 8, 1, 2, 5], [11, 1, 5, 7, 10, 9], [10, 8, 7, 11, 6, 2], [7, 8, 1, 5, 0, 4], [8, 6, 0, 10, 1, 3]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 46
La mejor solución es: [[1, 9, 8, 6, 5, 11], [5, 1, 10, 7, 4, 11], [11, 6, 5, 1, 7, 0], [5, 3, 1, 8, 10, 2], [8, 11, 6, 5, 9, 1]]
con  34  traslados de actores necesarios y cubre  12  tomas

Generacion # 47
La mejor solución es: [[5, 8, 9, 1, 7, 11], [0, 2, 8, 1, 4, 5], [10, 4, 1, 7, 8, 3], [2, 4, 8, 3, 6, 5], [6, 11, 8, 4, 1, 10]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 48
La mejor solución es: [[1, 7, 10, 11, 8, 5], [5, 2, 7, 6, 0, 9], [4, 1, 5, 6, 9, 11], [5, 6, 1, 0, 3, 8], [0, 6, 1, 7, 9, 8]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 49
La mejor solución es: [[4, 8, 3, 1, 0, 10], [7, 1, 0, 6, 8, 5], [0, 9, 7, 4, 10, 6], [2, 7, 5, 11, 0, 6], [7, 11, 5, 2, 10, 6]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 50
La mejor solución es: [[5, 8, 1, 7, 10, 6], [4, 9, 3, 7, 11, 0], [11, 6, 1, 8, 7, 5], [7, 9, 6, 8, 11, 5], [8, 0, 2, 1, 11, 7]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 51
La mejor solución es: [[6, 7, 8, 5, 4, 0], [1, 6, 8, 7, 5, 0], [6, 10, 1, 9, 11, 4], [10, 6, 3, 1, 4, 0], [2, 8, 5, 6, 3, 4]]
con  34  traslados de actores necesarios y cubre  12  tomas

Generacion # 52
La mejor solución es: [[6, 7, 5, 3, 11, 4], [7, 0, 6, 5, 1, 2], [8, 6, 7, 5, 9, 2], [5, 3, 2, 7, 8, 4], [10, 0, 2, 5, 4, 6]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 53
La mejor solución es: [[7, 4, 0, 6, 8, 5], [8, 11, 6, 4, 5, 7], [2, 6, 10, 5, 1, 8], [0, 3, 6, 4, 2, 5], [0, 9, 7, 1, 5, 11]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 54
La mejor solución es: [[8, 9, 1, 6, 5, 0], [6, 10, 5, 0, 7, 4], [6, 10, 5, 8, 3, 1], [1, 8, 2, 0, 10, 5], [5, 8, 9, 0, 11, 1]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 55
La mejor solución es: [[2, 4, 8, 5, 6, 3], [0, 8, 11, 9, 7, 1], [7, 8, 3, 4, 6, 2], [0, 6, 10, 1, 3, 2], [2, 1, 0, 10, 6, 8]]
con  34  traslados de actores necesarios y cubre  12  tomas

Generacion # 56
La mejor solución es: [[5, 11, 9, 7, 8, 0], [8, 3, 6, 0, 1, 5], [5, 8, 4, 10, 0, 1], [5, 0, 1, 11, 2, 6], [7, 9, 5, 4, 11, 10]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 57
La mejor solución es: [[3, 1, 6, 0, 10, 8], [6, 5, 2, 7, 11, 1], [8, 2, 4, 1, 3, 6], [11, 8, 2, 9, 5, 1], [4, 10, 3, 7, 1, 8]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 58
La mejor solución es: [[6, 4, 3, 5, 8, 2], [1, 3, 0, 4, 8, 5], [1, 6, 7, 10, 2, 5], [9, 6, 7, 5, 11, 2], [7, 5, 3, 10, 6, 2]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 59
La mejor solución es: [[6, 8, 10, 3, 1, 11], [8, 6, 0, 4, 1, 10], [5, 0, 1, 4, 8, 7], [6, 0, 7, 8, 5, 9], [8, 2, 3, 5, 10, 0]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 60
La mejor solución es: [[8, 5, 2, 3, 1, 10], [10, 2, 5, 1, 3, 0], [11, 6, 5, 0, 8, 2], [11, 7, 4, 9, 3, 5], [1, 8, 3, 5, 10, 2]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 61
La mejor solución es: [[0, 7, 9, 1, 11, 6], [1, 11, 6, 5, 4, 2], [1, 8, 5, 4, 10, 9], [4, 6, 3, 0, 8, 10], [1, 4, 3, 8, 0, 6]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 62
La mejor solución es: [[1, 4, 10, 5, 8, 0], [11, 5, 9, 4, 0, 10], [1, 5, 8, 7, 10, 6], [8, 2, 3, 6, 10, 1], [8, 7, 5, 10, 4, 1]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 63
La mejor solución es: [[6, 8, 5, 11, 7, 1], [4, 0, 10, 8, 1, 6], [1, 3, 2, 4, 10, 5], [7, 3, 5, 9, 2, 11], [2, 10, 5, 6, 1, 0]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 64
La mejor solución es: [[2, 7, 11, 1, 5, 8], [9, 1, 11, 0, 4, 10], [0, 1, 3, 7, 8, 2], [8, 4, 1, 10, 2, 3], [6, 8, 5, 11, 7, 1]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 65
La mejor solución es: [[10, 11, 6, 1, 7, 0], [10, 4, 1, 3, 2, 5], [7, 11, 8, 4, 10, 1], [11, 8, 9, 5, 4, 6], [6, 7, 11, 8, 1, 0]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 66
La mejor solución es: [[0, 4, 5, 8, 6, 2], [9, 11, 0, 8, 1, 2], [6, 3, 0, 2, 10, 8], [0, 11, 4, 8, 7, 1], [2, 6, 1, 11, 5, 4]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 67
La mejor solución es: [[5, 6, 3, 4, 10, 2], [3, 6, 8, 0, 5, 1], [10, 4, 7, 5, 6, 8], [5, 2, 10, 1, 3, 4], [5, 11, 0, 1, 9, 7]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 68
La mejor solución es: [[0, 9, 10, 11, 4, 7], [11, 7, 0, 8, 6, 5], [11, 2, 0, 5, 8, 7], [0, 2, 4, 3, 8, 6], [0, 11, 7, 8, 4, 1]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 69
La mejor solución es: [[9, 6, 8, 4, 5, 7], [1, 8, 11, 0, 6, 5], [11, 6, 4, 7, 2, 10], [11, 1, 10, 7, 4, 6], [0, 6, 8, 5, 4, 3]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 70
La mejor solución es: [[0, 7, 6, 1, 8, 5], [2, 4, 8, 6, 5, 3], [3, 1, 6, 10, 5, 4], [8, 4, 1, 10, 9, 6], [8, 4, 9, 1, 11, 6]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 71
La mejor solución es: [[2, 10, 4, 5, 0, 8], [1, 2, 5, 0, 11, 3], [11, 3, 9, 4, 1, 7], [8, 6, 4, 7, 11, 10], [5, 7, 6, 2, 3, 4]]
con  38  traslados de actores necesarios y cubre  12  tomas

Generacion # 72
La mejor solución es: [[6, 1, 2, 11, 0, 7], [4, 3, 1, 8, 5, 6], [5, 4, 11, 8, 0, 10], [9, 8, 7, 6, 4, 10], [4, 6, 3, 1, 2, 10]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 73
La mejor solución es: [[9, 8, 2, 7, 6, 5], [5, 1, 7, 8, 0, 11], [3, 4, 2, 1, 0, 10], [2, 6, 1, 4, 8, 5], [3, 7, 1, 8, 5, 10]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 74
La mejor solución es: [[6, 0, 5, 3, 4, 8], [7, 11, 9, 0, 6, 2], [8, 3, 2, 1, 4, 10], [4, 1, 8, 2, 10, 6], [1, 11, 8, 6, 7, 2]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 75
La mejor solución es: [[8, 7, 11, 10, 2, 3], [8, 5, 0, 7, 6, 11], [5, 7, 6, 11, 1, 2], [0, 8, 5, 4, 6, 7], [7, 1, 2, 11, 9, 6]]
```

```
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 76
La mejor solución es: [[6, 9, 5, 7, 11, 0], [9, 11, 10, 5, 4, 8], [8, 1, 5, 0, 11, 4], [5, 6, 11, 8, 7, 2], [6, 3, 8, 11, 10, 5]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 77
La mejor solución es: [[8, 5, 9, 6, 1, 0], [6, 11, 7, 2, 1, 5], [7, 6, 8, 1, 0, 5], [5, 2, 1, 6, 4, 10], [3, 6, 11, 0, 7, 4]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 78
La mejor solución es: [[8, 0, 4, 3, 6, 2], [6, 11, 8, 5, 7, 2], [3, 1, 8, 11, 2, 6], [2, 10, 7, 5, 0, 9], [1, 8, 0, 11, 7, 6]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 79
La mejor solución es: [[1, 0, 5, 10, 8, 11], [4, 3, 10, 1, 5, 8], [7, 5, 0, 4, 8, 9], [5, 2, 4, 3, 7, 8], [0, 11, 6, 5, 7, 8]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 80
La mejor solución es: [[7, 0, 6, 10, 8, 1], [1, 8, 0, 2, 3, 5], [3, 6, 4, 9, 2, 7], [8, 2, 0, 1, 10, 7], [7, 6, 11, 1, 5, 8]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 81
La mejor solución es: [[6, 5, 8, 1, 7, 0], [4, 3, 10, 0, 1, 6], [1, 0, 5, 7, 6, 9], [2, 5, 3, 6, 1, 8], [2, 8, 7, 11, 5, 6]]
con  34  traslados de actores necesarios y cubre  12  tomas

Generacion # 82
La mejor solución es: [[7, 10, 11, 4, 5, 0], [7, 6, 1, 5, 8, 0], [9, 2, 1, 7, 6, 11], [2, 3, 5, 4, 8, 6], [3, 1, 8, 10, 7, 5]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 83
La mejor solución es: [[2, 11, 8, 1, 9, 7], [6, 3, 2, 10, 5, 0], [7, 11, 6, 5, 1, 4], [1, 0, 10, 6, 7, 5], [5, 4, 11, 3, 1, 8]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 84
La mejor solución es: [[8, 0, 4, 1, 10, 5], [10, 11, 3, 7, 2, 5], [10, 1, 0, 3, 2, 8], [6, 8, 1, 9, 4, 5], [4, 8, 1, 0, 10, 11]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 85
La mejor solución es: [[6, 5, 10, 11, 0, 1], [5, 0, 7, 8, 2, 11], [9, 7, 2, 4, 3, 5], [0, 4, 2, 10, 8, 3], [8, 4, 6, 7, 5, 0]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 86
La mejor solución es: [[1, 8, 7, 9, 0, 6], [8, 0, 10, 11, 7, 6], [4, 5, 8, 1, 11, 6], [5, 2, 3, 11, 1, 6], [10, 8, 11, 4, 5, 6]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 87
La mejor solución es: [[8, 2, 4, 7, 5, 3], [7, 1, 5, 8, 6, 9], [4, 8, 0, 6, 11, 7], [2, 0, 4, 10, 6, 3], [4, 0, 11, 7, 6, 5]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 88
La mejor solución es: [[10, 5, 6, 3, 8, 1], [6, 0, 4, 8, 5, 1], [8, 11, 10, 7, 0, 1], [8, 1, 6, 4, 10, 2], [11, 7, 2, 8, 9, 6]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 89
La mejor solución es: [[3, 4, 11, 7, 10, 8], [8, 5, 4, 1, 6, 10], [8, 11, 7, 9, 0, 1], [7, 8, 11, 3, 2, 6], [6, 2, 4, 3, 5, 8]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 90
La mejor solución es: [[6, 8, 7, 9, 5, 11], [11, 5, 8, 6, 7, 0], [11, 9, 1, 5, 8, 0], [5, 0, 2, 11, 7, 3], [6, 10, 3, 8, 4, 1]]
con  35  traslados de actores necesarios y cubre  12  tomas

Generacion # 91
La mejor solución es: [[2, 0, 8, 3, 7, 1], [4, 0, 2, 8, 10, 5], [11, 1, 6, 9, 7, 0], [6, 0, 7, 5, 11, 9], [6, 3, 1, 10, 11, 7]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 92
La mejor solución es: [[2, 10, 4, 3, 5, 8], [6, 1, 5, 0, 2, 8], [11, 6, 7, 8, 9, 1], [6, 8, 11, 0, 1, 5], [0, 5, 2, 1, 9, 8]]
con  34  traslados de actores necesarios y cubre  12  tomas

Generacion # 93
La mejor solución es: [[4, 6, 1, 9, 0, 3], [6, 7, 4, 0, 5, 8], [8, 10, 1, 6, 0, 5], [0, 11, 1, 3, 2, 4], [11, 0, 5, 4, 1, 8]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 94
La mejor solución es: [[2, 5, 4, 1, 10, 8], [10, 8, 6, 0, 9, 4], [4, 6, 5, 7, 8, 3], [11, 7, 0, 5, 10, 9], [1, 8, 6, 0, 7, 5]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 95
La mejor solución es: [[11, 3, 8, 7, 2, 5], [8, 0, 5, 7, 6, 11], [4, 1, 6, 10, 0, 5], [10, 9, 6, 3, 8, 2], [3, 0, 8, 11, 5, 2]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 96
La mejor solución es: [[7, 1, 0, 6, 11, 2], [4, 9, 0, 8, 10, 6], [1, 10, 0, 4, 7, 11], [5, 6, 8, 11, 0, 10], [5, 3, 2, 8, 1, 6]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 97
La mejor solución es: [[5, 1, 8, 6, 3, 0], [7, 5, 11, 10, 2, 9], [7, 5, 0, 1, 4, 3], [8, 10, 6, 0, 1, 4], [1, 8, 5, 4, 6, 0]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 98
La mejor solución es: [[10, 7, 0, 8, 6, 11], [6, 7, 10, 11, 0, 4], [1, 3, 8, 6, 0, 10], [9, 4, 5, 11, 10, 0], [8, 2, 10, 1, 0, 3]]
con  36  traslados de actores necesarios y cubre  12  tomas

Generacion # 99
La mejor solución es: [[7, 2, 0, 3, 5, 1], [10, 9, 5, 8, 7, 6], [2, 1, 0, 11, 5, 7], [8, 2, 4, 7, 5, 6], [6, 4, 11, 5, 1, 7]]
con  37  traslados de actores necesarios y cubre  12  tomas

Generacion # 100
La mejor solución es: [[8, 1, 3, 5, 10, 0], [8, 4, 10, 5, 3, 7], [10, 4, 6, 7, 5, 0], [9, 5, 6, 0, 7, 11], [2, 4, 0, 11, 1, 6]]
con  37  traslados de actores necesarios y cubre  12  tomas

La mejor solucion encontrada ha sido:  [[8, 18, 15, 17, 16, 1], [6, 20, 21, 28, 11, 9], [26, 29, 13, 7, 3, 14], [10, 24, 4, 12, 2, 19], [25, 0, 5, 27, 22, 23]]
La cual necesita  38  desplazamientos y cubre  30  tomas
```

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

(*)Calcula la complejidad del algoritmo

El mayor número de bucles anidados que nos encontramos dentro del algoritmo genético es de 2, por lo que su complejidad se queda en O(n^2)