Seminario_Algoritmos

July 30, 2023

1 Algoritmos de optimización - Seminario

Nombre y Apellidos: Luis Jama Tello Url: https://github.com/ljham/03MIAR_ALG_OPTZ/tree/main/SEMINA
Problema: 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.

- 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

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

Dado que hay N actores y M tomas, el número de combinaciones actor-toma es N x M. Por lo tanto, el número total de posibilidades sin restricciones es:

Posibilidades sin restricciones = 2^(N x M)

Por ejemplo, si hay 10 actores y 30 tomas, el número de posibilidades sin restricciones sería:

Posibilidades sin restricciones = 2^(10 x 30) = 2^300

¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.

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)

Respuesta:

Para el problema planteado se han utilizado dos estructuras de datos :

1.- Pandas : Para leer los datos del problema desde una hoja de cálculo en Google Sheets, en la cual, después de haber descargado la información se ha realizado una limpieza de datos y una preparación adecuada para utilizar en el algoritmo que resuelve el problema de planificación del doblaje. El uso de está librería para cargar y manipular los datos facilita la organización y limpieza de los mismos, permitiendo la implementación del algoritmo en sí.

2.- Numpy : Para trabajar con matrices y realizar operaciones matriciales eficientes. La estructura de datos clave utilizada en el algoritmo es una matriz bidimensional, que representa la participación de los actores en cada toma. Cada fila de la matriz corresponde a una toma, y cada columna corresponde a un actor. Los valores en la matriz son 1 o 0, donde 1 indica que el actor participa en la toma y 0 indica que no participa.

Según el modelo para el espacio de soluciones ___(*) ¿Es un problema de maximización o minimización?___

___(*) ¿Cual es la función objetivo?___

Respuesta:

El problema planteado es de ***minimización***. El objetivo del problema es planificar las sesiones por día de manera que el gasto por los servicios de los actores de doblaje sea el menor posible.

Lo que se busca es encontrar una asignación de tomas a los días de grabación que minimice el costo total de los servicios de los actores. El costo total se calcula sumando los costos de los actores que participan en cada toma en cada día acorde a las restricciones planteadas para la solución del mismo.

Por lo tanto, el objetivo del algoritmo es encontrar una solución que minimice el costo total, lo que implica minimizar la función objetivo que representa el costo de los actores.

Funcion Objetivo :

Sea N el número total de actores. Sea M el número total de tomas. Sea x_{ij} una variable binaria que indica si el actor i participa en la toma j, entonces $x_{ij} = 1$; de lo contrario, $x_{ij} = 0$. Sea c_{ij} el costo asociado al actor i en la toma j.

La función objetivo F que buscamos minimizar es :

$$f(x) = \sum_{i=1}^{N} \sum_{j=1}^{M} C_{ij} \cdot x_{ij}$$

Sujeto a las siguientes restricciones :

Cada toma j debe asignarse a exactamente un día d. Por lo tanto, para cada toma j, tenemos:

$$\sum_{d=1}^{D} x_{dj} = 1$$

donde D es el número total de días de grabación. No se pueden grabar más de 6 tomas por día. Por lo tanto, para cada día d, tenemos:

$$\sum_{j=1}^{M} x_{dj} \leq 6$$

Las variables x_{ij} deben ser binarias:

$$x_{ij} \epsilon \{0, 1\}$$

El objetivo del problema es encontrar los valores de las variables x_{ij} que minimicen la función objetivo F, cumpliendo con las restricciones mencionadas anteriormente.

Diseña un algoritmo para resolver el problema por fuerza bruta :

Respuesta:

[ ]:
```
import pandas as pd
import numpy as np
import random
```

[ ]:
```
# Obtiene matriz de datos para dar solucion al problema planteado
path = 'https://docs.google.com/spreadsheets/d/
  ↪1Ipn6IrbQP4ax8zOnivdBIw2lN0JISkJG4fXndYd27U0/export?gid=0&format=csv'
data = pd.read_csv(path, sep=',', header=1)

# Elimina columna inncesaria del archivo descargado
data = data.drop(['Unnamed: 11'], axis =1)

# Elimina filas con datos basura
data = data.drop([30, 31])

# Convierte tipos de datos
data = data[["Toma","1","2","3","4","5","6","7","8","9","10"]].astype(int)

display(data)

# Obtiene lista de actores
actores = data.columns[1:].astype(int).to_numpy()

# Obtiene un detalle del nuúmero de tomas
tomas = data[data.columns[0:1]].to_numpy()

# Obtiene la matriz de tomas asignadas a cada actor
tomas_actores = data[data.columns[1:]].to_numpy()
```

	Toma	1	2	3	4	5	6	7	8	9	10
0	1	1	1	1	1	1	0	0	0	0	0
1	2	0	0	1	1	1	0	0	0	0	0
2	3	0	1	0	0	1	0	1	0	0	0

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

[ ]:
```
def obtiene_solucion_tomas(tomas, tomas_actores):

    dia = 1
    solucion = np.zeros((len(tomas),3)).astype(int)
    matriz_tomas = np.zeros(10, dtype=int).reshape(1,10)

    for idx, toma in enumerate(tomas):

        # Obtiene la toma actual de la matriz de tomas
        toma_actual = tomas_actores[toma-1]

        # Calcula la diferencia de matrices entre la matriz actual y la matriz␣
↪de tomas
        diferencia = np.subtract(toma_actual, np.add.reduce(matriz_tomas,␣
↪axis=0))
        diferencia[diferencia < 0] = 0

        # Agrega la toma actual a la matriz de tomas
        matriz_tomas = np.append(matriz_tomas, toma_actual, axis=0)
```

```
        #Almacenamos el día, número de toma y costo adicional de la toma
        solucion[idx] = [dia, toma[0], diferencia.sum()]
        # print(f'dia [{dia}], toma{toma}, costo[{diferencia.sum()}] ')

        # Verifica el número de días máximo para las tomas
        if(toma%6==0):
            # Inicia desde cero la matriz de toma
            matriz_tomas = np.zeros(10, dtype=int).reshape(1,10)
            dia+=1
    return solucion
```

[ ]:
```
solucion = obtiene_solucion_tomas(tomas, tomas_actores)
print (f'Costo minímo = {np.sum(solucion[:,2], axis = 0)}')
print("Mejor asignación de actores a tomas:")
print(solucion)
```

```
Costo minímo = 38
Mejor asignación de actores a tomas:
[[ 1  1  5]
 [ 1  2  0]
 [ 1  3  1]
 [ 1  4  1]
 [ 1  5  0]
 [ 1  6  0]
 [ 2  7  4]
 [ 2  8  1]
 [ 2  9  0]
 [ 2 10  1]
 [ 2 11  2]
 [ 2 12  0]
 [ 3 13  3]
 [ 3 14  2]
 [ 3 15  2]
 [ 3 16  1]
 [ 3 17  0]
 [ 3 18  0]
 [ 4 19  2]
 [ 4 20  2]
 [ 4 21  2]
 [ 4 22  1]
 [ 4 23  0]
 [ 4 24  0]
 [ 5 25  4]
 [ 5 26  3]
 [ 5 27  0]
 [ 5 28  0]
```

```
 [ 5 29  1]
 [ 5 30  0]]
```

Calcula la complejidad del algoritmo por fuerza bruta

Respuesta:

El algoritmo recorre todas las tomas una vez, por lo que la complejidad de esta parte es O(N).

Dentro del bucle que recorre las tomas, se realiza una operación de suma **(np.add.reduce(matriz_tomas, axis=0))** y una operación de resta **(np.subtract(toma_actual, np.add.reduce(matriz_tomas, axis=0))).** Cada una de estas operaciones de suma y resta tiene una complejidad de O(M), ya que involucra sumar o restar M elementos. Por lo tanto, la complejidad de estas operaciones dentro del bucle es O(M).

Dentro del bucle, se realiza una operación de concatenación de matrices **(np.append(matriz_tomas, toma_actual, axis=0)).** La operación de concatenación de matrices tiene una complejidad de O(NM), ya que debe copiar todos los elementos de ambas matrices en una nueva matriz. Por lo tanto, la complejidad de esta operación dentro del bucle es O(NM).

El bucle también realiza algunas operaciones como asignaciones y verificaciones condicionales, que tienen una complejidad constante y no afectan significativamente la complejidad total del algoritmo.

En resumen, la complejidad del algoritmo es : **O(N * (M + N * M)) = O(N^2 * M)**, ya que la operación de concatenación de matrices dentro del bucle es la que domina la complejidad.

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

Respuesta:

El algoritmo que se utilizó para mejorar la complejidad fue el algoritmo greddy, el cual obtiene una solución inicial aproximada al problema de planificación de sesiones de doblaje en donde se seleccionan tomas de forma voraz considerando en cada paso la toma que maximice la cantidad de actores que no han participado previamente en ninguna toma asignada.

[ ]:
```python
def obtiene_matriz_tomas_aleatoria(tomas):
    return np.random.choice(tomas, len(tomas), replace=False)

# Obtiene la distancia entre tomas, es decir los actores que yan realizado una␣
 ↪toma e identifica los que tienen una toma nueva
def distancia_entre_tomas(toma_actual, toma_actores):
    diferencia = np.subtract(toma_actores, toma_actual)
    diferencia[diferencia < 0] = 0

    return np.sum(diferencia, axis=1)
```

[ ]:
```python
def obtiene_tomas_greddy(tomas, m_tomas_actores):
    # Para iniciar se excluye de la lista de tomas el primer elemento
    tomas_disponibles = list(tomas[1:])
```

```
    # Inicializa la lista de tomas seleccionando el primero elemento de la␣
 ↪lista de tomas
    tomas_greddy = list(tomas[0:1])

    # Obtiene la primera toma
    actores_toma_actual = m_tomas_actores[0]

    # Se obtiene la distancia entre la primera toma y la matriz de tomas
    # para identificar los actores que no han participado en tomas anteriores
    distancia_tomas = distancia_entre_tomas(actores_toma_actual,␣
 ↪m_tomas_actores)

    # Excluye de la matriz de distancia el primero elemento
    distancia_tomas = distancia_tomas[1:]

    tamanio_tomas_disponibles = len(tomas_disponibles)
    while(tamanio_tomas_disponibles > 0):
        # Obtiene la distancia minima de la toma actual y la matriz de tomas
        distancia_minima = np.argmin(distancia_tomas)

        toma_seleccionada = tomas[distancia_minima]
        if(toma_seleccionada in tomas_disponibles):
            tomas_greddy.append(toma_seleccionada)
            tomas_disponibles.remove(toma_seleccionada)
            tamanio_tomas_disponibles= tamanio_tomas_disponibles - 1

            # Identifica los actores que han participado en la toma actual y la␣
 ↪matriz de tomas con distancia minima
            actores_toma_actual = np.add(actores_toma_actual,␣
 ↪m_tomas_actores[distancia_minima])
            actores_toma_actual[actores_toma_actual > 1] = 1
            distancia_tomas = distancia_entre_tomas(actores_toma_actual,␣
 ↪m_tomas_actores)
        else:
            distancia_tomas[distancia_minima] = 9999
    return np.array(tomas_greddy).reshape(len(tomas),1)
```

[ ]:
```
m_tomas = tomas.reshape(len(tomas))

# Obtiene la lista de tomas de acuerdo a las tomas en donde los actores han␣
 ↪realizado al menos una intervencion
tomas_greddy = obtiene_tomas_greddy(m_tomas, tomas_actores)

solucion_greddy = obtiene_solucion_tomas(tomas_greddy, tomas_actores)
print (f'Costo minímo = {np.sum(solucion_greddy[:,2], axis = 0)}')
print("Mejor asignación de actores a tomas (algoritmo greddy) :")
```

```
print(solucion_greddy)
```

```
Costo minímo = 34
Mejor asignación de actores a tomas (algoritmo greddy) :
[[ 1  1  5]
 [ 1  5  1]
 [ 1  2  0]
 [ 1  6  0]
 [ 2  7  4]
 [ 2  9  0]
 [ 2 11  2]
 [ 2 13  0]
 [ 2 17  0]
 [ 2 19  0]
 [ 2 20  0]
 [ 2 22  0]
 [ 2 23  0]
 [ 2 27  0]
 [ 2 28  0]
 [ 2 30  0]
 [ 3  3  3]
 [ 3  4  2]
 [ 3 15  0]
 [ 3  8  1]
 [ 3 12  2]
 [ 4 14  3]
 [ 4 18  0]
 [ 5 21  2]
 [ 5 24  1]
 [ 6 29  3]
 [ 6 10  2]
 [ 6 26  1]
 [ 6 16  2]
 [ 6 25  0]]
```

El costo minimo calculado por el algoritmo greddy es de 34 un valor menor al calculado por el algoritmo de fuerza bruta

(*)Calcula la complejidad del algoritmo

Respuesta:

La complejidad del algoritmo planteado es : O(n^2) en el peor caso.

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

Respuesta:

[ ]:
```
# Se define una matriz 80 tomas y 10 actores
# Para eso creamos una matriz con 1 y 0 de manera aleatoria
```

```
n_tomas = 80
m_tomas_actores = np.random.randint(2, size=(n_tomas, 10))
t_tomas = np.arange(1, n_tomas + 1,1)
```

Aplica el algoritmo al juego de datos generado

Respuesta:

[ ]:
```
# Obtiene la lista de tomas de acuerdo a las tomas en donde los actores han realizado al menos una intervencion
tomas_greddy = obtiene_tomas_greddy(t_tomas, m_tomas_actores)

solucion_greddy = obtiene_solucion_tomas(tomas_greddy, m_tomas_actores)
print (f'Costo minímo = {np.sum(solucion_greddy[:,2], axis = 0)}')
print("Mejor asignación de actores a tomas (algoritmo greddy) :")
print(solucion_greddy)
```

```
Costo minímo = 114
Mejor asignación de actores a tomas (algoritmo greddy) :
[[ 1  1  3]
 [ 1 49  3]
 [ 1 36  0]
 [ 2 50  0]
 [ 2 58  4]
 [ 2 66  1]
 [ 3  2  6]
 [ 3 34  1]
 [ 3 57  0]
 [ 3  3  1]
 [ 3 29  0]
 [ 3 30  0]
 [ 4 31  6]
 [ 4 37  1]
 [ 4 51  1]
 [ 4 52  0]
 [ 4 53  0]
 [ 4 54  0]
 [ 5 59  3]
 [ 5 61  2]
 [ 5 64  1]
 [ 5 72  0]
 [ 6 80  4]
 [ 6  5  5]
 [ 6 11  0]
 [ 6 19  0]
 [ 6 23  0]
 [ 6 25  0]
```

```
[ 6 27  0]
[ 6 32  0]
[ 6 42  0]
[ 7 43  4]
[ 7 47  3]
[ 7 60  0]
[ 8 65  6]
[ 8 67  1]
[ 8 69  1]
[ 8 70  0]
[ 8 73  1]
[ 8  4  1]
[ 8  6  0]
[ 9  7  3]
[ 9  8  3]
[ 9  9  0]
[ 9 10  0]
[ 9 12  3]
[10 13  4]
[10 14  3]
[10 15  2]
[10 16  0]
[10 17  1]
[10 18  0]
[11 20  6]
[11 21  3]
[11 22  0]
[11 24  1]
[12 26  6]
[12 28  2]
[12 33  1]
[12 35  1]
[12 38  0]
[12 39  0]
[12 40  0]
[12 41  0]
[12 44  0]
[12 45  0]
[12 46  0]
[12 48  0]
[13 55  6]
[13 56  3]
[13 62  1]
[13 63  0]
[13 68  0]
[13 71  0]
[13 74  0]
[13 75  0]
```

```
 [13 76  0]
 [13 77  0]
 [13 78  0]
 [14 79  6]]
```

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

Respuesta:

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

Respuesta: