Trabajo_Práctico_Algoritmos(V2)

March 1, 2024

1 Algoritmos de optimización - Trabajo Práctico

Nombre y Apellidos: Adrián Pérez Portero Url: https://github.com/…/03MAIR—Algoritmos-de-Optimizacion—/tree/master/TrabajoPractico Google Colab: https://colab.research.google.com/drive/xxxxxxxxxxxxxxx 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:

2 Modelo

- ¿Como represento el espacio de soluciones?

Se consideran todas las posibles combinaciones de tomas para cada día. Cada solución es una lista de listas, donde cada lista interna representa las tomas asignadas para un día específico. El objetivo es encontrar la mejor combinación de estas listas de tomas para minimizar el coste de grabación de todas las tomas, definido como el número total de actores activos en todas las tomas.

- ¿Cual es la función objetivo?

calcular_coste(solucion)

Consiste en minimizar el coste total, que se logra al maximizar la eficiencia en la asignación de tomas de manera que se involucre al menor número posible de actores.

- ¿Como implemento las restricciones?

1) Número máximo de tomas por día

Se crea la variable 'max_tomas_dia' para establecer la restricción de que no pueden grabarse más de 6 tomas al día.

2) Disponibilidad de actores

Se crea la matriz tomas que representa la disponibilidad de actores para cada toma. Las filas representan las tomas y las columnas los actores de doblaje marcados con 1 cuando son requeridos en la toma o 0 si no.

Para la selección aleatoria de la toma, se verifica que haya actores disponibles para esa toma.

3) Única asignación de tomas

La matriz tomas se va reduciendo a medida que van siendo seleccionadas:

tomas_aux.pop(pos)

[1]:
```
#Respuesta

import pulp
import numpy as np
import random

# Espacio de soluciones
max_tomas_dia = 6
solucion = []
mejor_solucion = []
tomas_dia = []
mejor_coste = 100000
max_iteraciones = 100000

tomas = [
    [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 5],
    [0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 3],
    [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, 3],
    [1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 4],
    [1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 4],
    [1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 3],
    [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 3],
    [1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 4],
    [1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 5],
    [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 5],
    [1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 3],
    [1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 3],
    [1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 3],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 2],
    [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2],
    [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 2],
    [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2],
    [1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 4],
    [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 2],
    [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 4],
    [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2],
    [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 2],
    [1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 4],
    [1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 4],
```

```
    [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 2],
    [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2],
    [1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 3],
    [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2]
]
```

[23]:
```python
def numero_filas_activas(matriz):
    activadas = np.zeros(len(matriz[0]))

    for fila in matriz:
        for j, valor in enumerate(fila):
            if valor == 1:
                activadas[j] = 1

    total_activadas = int(np.sum(activadas))

    return total_activadas

def pos_toma_aleatoria(tomas):
    if not tomas:
        return None  # Devuelve None si la matriz está vacía
    indice_fila_aleatoria = random.randint(0, len(tomas) - 1)
    return indice_fila_aleatoria

def calcular_coste(solucion):
    coste = 0

    for tomas_dia in solucion:
        coste += numero_filas_activas(tomas_dia)

    return coste
```

[24]:
```python
for it in range(max_iteraciones):
    tomas_aux = tomas.copy()
    while(len(tomas_aux) > 0):

        i = 0
        while(i < max_tomas_dia and len(tomas_aux) > 0):
            pos = pos_toma_aleatoria(tomas_aux)
            tomas_dia.append(tomas_aux[pos])
            tomas_aux.pop(pos)
            i += 1

        # Agregamos a solucion
        solucion.append(tomas_dia)
        tomas_dia = []
```

```
    coste = calcular_coste(solucion)

    if coste < mejor_coste:
        mejor_coste = coste
        mejor_solucion = solucion.copy()

    solucion = []

print('Mejor coste: ', str(mejor_coste))
```

```
Mejor coste:  32
```

3 Análisis

- ¿Que complejidad tiene el problema?. Orden de complejidad y Contabilizar el espacio de soluciones

3.0.1 Respuesta

4 Diseño

- ¿Que técnica utilizo? ¿Por qué?

4.0.1 Respuesta

Se ha utilizado la técnica de búsqueda local, ya que el espacio de búsqueda es grande y no se dispone de información que permita realizar una búsqueda exhaustiva en un tiempo razonable.

En este caso, el espacio de soluciones es el conjunto de todas las posibles asignaciones de tomas para cada día, lo que resulta en un espacio de búsqueda de alta dimensionalidad debido a la combinación de diferentes tomas para múltiples días. Además, el número de combinaciones posibles puede ser muy grande, lo que hace que una búsqueda exhaustiva sea computacionalmente costosa y poco práctica.

La técnica de búsqueda local, en contraste, es adecuada para problemas donde se tiene un espacio de búsqueda grande y no se dispone de una función de evaluación global. En lugar de intentar explorar todo el espacio de búsqueda, la búsqueda local busca iterativamente mejorar una solución inicial a través de movimientos locales, evaluando la vecindad de una solución en lugar de todo el espacio de búsqueda. Esto permite una exploración más eficiente del espacio de soluciones y puede conducir a soluciones aceptables en un tiempo razonable.

Dado que el objetivo en este problema es encontrar una asignación óptima de tomas para minimizar el número total de actores activos, la técnica de búsqueda local se adapta bien, ya que puede explorar de manera eficiente las diferentes combinaciones de tomas y evaluar su costo, buscando iterativamente mejorar la solución actual.