Algoritmos de optimización - Trabajo Práctico

Nombre y Apellidos: Juan Aroca Pérez
Url: https://github.com/JuanArocaMIAR/03MIAR---Algoritmos-de-Optimizacion---2023.git
Google Colab: https://colab.research.google.com/drive/1LGoQlTetXJ7OG3Qm3OBcyKunZxgf_QNU#scrollTo=Sbo9IqQtpaHZ

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.

Modelo

- ¿Como represento el espacio de soluciones?
- ¿Cual es la función objetivo?
- ¿Como implemento las restricciones?

¿Cómo represento el espacio de soluciones?

Las soluciones generadas representan la asignación de grabaciones a días de grabación. Cada solución se representa como un diccionario donde la clave es el día y el valor asociado es una lista que contiene las diferentes grabaciones que se graban ese día. Esta estructura de datos se imprime en pantalla para mostrar cómo se han asignado las grabaciones a los días.

¿Cuál es la función objetivo?

La función objetivo busca minimizar el coste total de los servicios de los actores de doblaje. Por lo que sumamos los costes de las tomas asignadas a cada día. en el código se encuentra en la variable ***coste_grabacion***

¿Cómo implemento las restricciones?

- **Número de tomas:**

  Esta restricción se asegura de que la suma de tomas asignadas a cada día no exceda 6. Se implementa agregando una restricción para cada día en la matriz binaria que representa las asignaciones de tomas a días. Esta restricción establece que la suma de las tomas asignadas a ese día debe ser menor o igual a 6.

- **Asignación única:**

  Esta restricción garantiza que cada toma se grabe en un día y no en el resto. Esto se logra asegurándose de que la suma de las asignaciones de las tomas a un día específico sea igual a 1.

- **Restricción de coincidencia de actores:**

  Así se asegura que los actores que comparten una toma estén asignados al mismo día. Se implementa asegurándose de que si dos tomas tienen actores en común, entonces deben asignarse al mismo día.

Análisis

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

Orden de complejidad:

- Ordenamiento de tomas: Esto tiene una complejidad de tiempo de O(n log n), donde n es el número de tomas.
- Asignación de tomas a días: En el peor de los casos, cada toma debe ser asignada a un nuevo día, lo que resultaría en O(n^2) operaciones.
- Cálculo del coste total: Esto tiene una complejidad de tiempo de O(n), donde n es el número total de tomas asignadas.

En total, la complejidad del algoritmo es dominada por el ordenamiento de las tomas (O(n log n)) y la asignación de tomas a días (O(n^2)), por lo que la complejidad de froam general puede definirse como O(n^2).

Espacio de soluciones:

Para el espacio de soluciones, se considera que hay 30 tomas en total. Cada toma puede asignarse a cualquiera de los días posibles, lo que significa que hay 6 posibilidades para la primera toma, 6 para la segunda, y así sucesivamente. Por lo tanto, el espacio de soluciones es 6^30.

Diseño

- ¿Que técnica utilizo? ¿Por qué?

El codigo se basa en la técnica voraz debido al equilibrio entre simplicidad y eficiencia, ya que proporciona soluciones aceptables en un tiempo razonable. La elección de esta técnica se debe a varias razones:

1. **Facilidad de implementación**: La estrategia de técnica voraz es bastante sencilla de entender y de implementar en comparación con métodos más complejos.
2. **Menor complejidad**: A diferencia de enfoques más elaborados, la técnica voraz no requiere manipulaciones complejas.
3. **Eficiencia computacional**: Aunque la técnica voraz no garantiza la solución óptima, suele proporcionar soluciones aceptables en un tiempo razonable, especialmente para problemas de tamaño moderado.
4. **Buena aproximación**: En muchos casos, la estrategia de técnica voraz puede dar soluciones bastante cercanas a la óptima.

1º: Carga de los datos del problema

```
# Librerias necesarias

import numpy as np
import pandas as pd
from google.colab import files
import random
from copy import deepcopy
import math



# Subir el archivo CSV desde tu sistema local
archivo = files.upload()

# Seleccionar el archivo
datos = list(archivo.keys())[0]

# Cargar el dataframe desde el archivo CSV
df = pd.read_csv(datos, index_col=0)

# # Modificar y limpiar datos
df = df.iloc[1:30, 0:10]
```

```
Elegir archivos Datos prob… Hoja 1.csv
• Datos problema doblaje(30 tomas, 10 actores) - Hoja 1.csv(text/csv) - 900 bytes, last modified: 18/3/2024 - 100% done
Saving Datos problema doblaje(30 tomas, 10 actores) - Hoja 1.csv to Datos problema doblaje(30 tomas, 10 actores) - Hoja 1 (3).cs
```

```
tabla_actores = df.values.tolist()
```

2º: Técnica Voraz

```
def planificar_doblaje(tabla_actores_np):
    num_tomas = len(tabla_actores_np)
    num_actores = len(tabla_actores_np[0])

    # Inicializa la asignación de tomas a días
    planificacion = {}

    # Ordena las tomas por la cantidad de actores presentes
    tomas_ordenadas = sorted(range(num_tomas), key=lambda x: -np.sum(tabla_actores_np[x]))

    # Asigna las tomas a días en orden descendente de actores presentes
    for toma in tomas_ordenadas:
        asignada = False
        for dia, tomas in planificacion.items():
            if len(tomas) + 1 <= 6 and not any(tabla_actores_np[toma][actor] and actor + 1 in tomas for actor in range(num_actores)):
                tomas.append(toma)
                asignada = True
                break
        if not asignada:
            planificacion[len(planificacion) + 1] = [toma]

    # Calcula el coste total de los actores que van al estudio en todas las sesiones
    coste_grabacion = sum(len(set([actor + 1 for toma in dia for actor in range(num_actores) \
                                   if tabla_actores_np[toma][actor] == 1])) for dia in planificacion.values())
    print("~ ORGANIZACIÓN DE TOMAS POR DIAS ~")
    print("---------------------------------")
    for dia, tomas in planificacion.items():
        print(f"Día {dia} --------------------------")
        print(f"Tomas: {', '.join(map(lambda x: str(x + 1), tomas))}")
        actores_presentes = set()
        for toma in tomas:
            actores_toma = [j + 1 for j in range(len(tabla_actores_np[toma])) if tabla_actores_np[toma][j] == 1]
            actores_presentes.update(actores_toma)
        print(f"Actores necesarios para grabar: {', '.join(map(str, actores_presentes))}")
        print()

    print(f"El coste total de la grabación es: {coste_grabacion}")
    return planificacion, coste_grabacion

tabla_actores_np = np.array(tabla_actores)
planificacion, coste_grabacion = planificar_doblaje(tabla_actores_np)
```

```
~ ORGANIZACIÓN DE TOMAS POR DIAS ~
---------------------------------
Día 1 --------------------------
Tomas: 1, 11, 12, 4, 6, 10
Actores necesarios para grabar: 1, 2, 3, 4, 5, 6, 7, 8, 9

Día 2 --------------------------
Tomas: 7, 20, 22, 25, 26, 2
Actores necesarios para grabar: 1, 2, 3, 4, 5, 9, 10

Día 3 --------------------------
Tomas: 3, 13, 14, 29, 16, 17
Actores necesarios para grabar: 1, 2, 3, 4, 5, 6, 7, 10

Día 4 --------------------------
Tomas: 5, 8, 18, 19, 21, 23
Actores necesarios para grabar: 1, 2, 3, 4, 6, 8

Día 5 --------------------------
Tomas: 9, 15, 24, 27, 28
Actores necesarios para grabar: 1, 2, 3, 4, 5, 6, 7

El coste total de la grabación es: 37
```

3º: Técnica voraz con componentes heurísticos:

A continuación, se ha intentado incorporar distintos componentes heurísticos al código para mejorar la eficiencia y efectividad del algoritmo:

- Ordenación de las tomas por suma de duración y cantidad de actores necesarios: Esto puede ayudar a priorizar las tomas que son más complejas y requieren más recursos, lo que podría conducir a una asignación más eficiente.
- Asignación basada en la capacidad restante y la complejidad de las tomas: Esto implica calcular el coste de agregar una toma a un día en función de la cantidad de actores necesarios y los actores que ya están programados para ese día.

```
def planificar_doblaje(tabla_actores_np):
    num_tomas = len(tabla_actores_np)
    num_actores = len(tabla_actores_np[0])

    # Inicializa la asignación de tomas a días
    planificacion = {}

    # Ordena las tomas por la suma de la duración de las tomas y la cantidad de actores necesarios
    tomas_ordenadas = sorted(range(num_tomas), key=lambda x: (np.sum(tabla_actores_np[x]), -np.sum(tabla_actores_np[x])))

    # Asigna las tomas a días en función de la capacidad restante y la complejidad de las tomas
    for toma in tomas_ordenadas:
        asignada = False
        min_coste = float('inf')
        min_dia = None
        for dia in planificacion:
            if len(planificacion[dia]) + 1 <= 6:
                # Calcula el coste de agregar la toma a este día
                coste = sum(tabla_actores_np[toma][actor] for actor in range(num_actores)) + \
                len(set([actor + 1 for actor in range(num_actores) \
                         if tabla_actores_np[toma][actor] == 1]).difference(set([actor + 1 for toma in planificacion[dia] \
                                                 for actor in range(num_actores) if tabla_actores_np[toma][actor] == 1])))
                if coste < min_coste:
                    min_coste = coste
                    min_dia = dia
        if min_dia is not None:
            if min_dia not in planificacion:
                planificacion[min_dia] = []
            planificacion[min_dia].append(toma)
        else:
            planificacion[len(planificacion) + 1] = [toma]

    # Calcula el coste total de los actores que van al estudio en todas las sesiones
    coste_grabacion = sum(len(set([actor + 1 for toma in planificacion[dia] for actor in range(num_actores) \
                                   if tabla_actores_np[toma][actor] == 1])) for dia in planificacion)
    print("~ ORGANIZACIÓN DE TOMAS POR DIAS ~")
    print("---------------------------------")
    for dia in planificacion:
        print(f"Día {dia} --------------------------")
        print(f"Tomas: {', '.join(str(toma + 1) for toma in planificacion[dia])}")
        actores_presentes = set()
        for toma in planificacion[dia]:
            actores_toma = [j + 1 for j in range(len(tabla_actores_np[toma])) if tabla_actores_np[toma][j] == 1]
            actores_presentes.update(actores_toma)
        print(f"Actores necesarios para grabar: {', '.join(map(str, actores_presentes))}")
        print()

    print(f"El coste total de la grabación es: {coste_grabacion}")
    return planificacion, coste_grabacion


# Ejemplo de uso:
tabla_actores_np = np.array(tabla_actores)
planificacion, coste_grabacion = planificar_doblaje(tabla_actores_np)
```

```
~ ORGANIZACIÓN DE TOMAS POR DIAS ~
---------------------------------
Día 1 --------------------------
Tomas: 16, 17, 18, 19, 21, 23
Actores necesarios para grabar: 1, 3, 4, 6, 8, 10

Día 2 --------------------------
Tomas: 24, 27, 28, 2, 3, 5
Actores necesarios para grabar: 1, 2, 3, 4, 5, 6, 7, 8

Día 3 --------------------------
Tomas: 8, 9, 13, 14, 15, 29
Actores necesarios para grabar: 1, 2, 3, 4, 5, 6, 7

Día 4 --------------------------
Tomas: 4, 6, 7, 10, 20, 22
Actores necesarios para grabar: 1, 2, 3, 4, 5, 6, 7, 8, 9

Día 5 --------------------------
Tomas: 25, 26, 1, 11, 12
Actores necesarios para grabar: 1, 2, 3, 4, 5, 6, 8, 9, 10

El coste total de la grabación es: 39
```

Como se puede ver la introducción de estos componentes heurísticos no conlleva conseguir una mejor solución. Esto puede ser por la ordenación de las tomas, la exhaustividad de la búsqueda o la sensibilidad a los datos de entrada.

4º: Recocido Simulado

Como última opción, se va a comprobar si se pueden obetener mejores soluciones mediante el uso de ***recocido simulado***. A continuación se puede ver el código:

```
def asignar_grabaciones_aleatorias():
    grabaciones = list(range(num_grabaciones))
    random.shuffle(grabaciones)

    dias = []
    index = 0

    while index < num_grabaciones:
        dia = grabaciones[index:index+6]
        dias.append(deepcopy(dia))
        index += 6
    return dias


def probabilidad_aceptacion(gasto_actual, nuevo_gasto, temperatura):
    if nuevo_gasto < gasto_actual:
        # Si la nueva solución es mejor se acepta
        return 1.0
    # Calcular la probabilidad de aceptar una solución peor utilizando la fórmula del recocido simulado
    return math.exp((gasto_actual - nuevo_gasto) / temperatura)
```

```
def calcular_coste_total(asignacion, tabla_actores):
    coste_total = 0

    for dia in asignacion:
        actores_presentes = obtener_actores_presentes(dia, tabla_actores)
        cantidad_actores = len(actores_presentes)
        coste_total += cantidad_actores

    return coste_total


def obtener_actores_presentes(dia, tabla_actores):
    actores_presentes = set()
    for grabacion in dia:
        for actor in range(1, num_actores + 1):
            if tabla_actores[grabacion][actor - 1] == 1:
                actores_presentes.add(actor)
    return actores_presentes


def generar_vecino(asignacion):
    vecino = deepcopy(asignacion)

    # Elige dos días aleatorios para el intercambio
    dia1 = random.randint(0, len(asignacion) - 1)
    dia2 = random.randint(0, len(asignacion) - 1)

    if dia1 != dia2:
        # Elige una grabación aleatoria en cada día para el intercambio
        grabacion1 = random.choice(vecino[dia1])
        grabacion2 = random.choice(vecino[dia2])

        # Realiza el intercambio de las grabaciones entre los días
        vecino[dia1].remove(grabacion1)
        vecino[dia2].remove(grabacion2)
        vecino[dia1].append(grabacion2)
        vecino[dia2].append(grabacion1)
    return vecino
```

```
def recocido_simulado(temperatura_inicial, temperatura_final, factor_enfriamiento, iteraciones_por_temperatura, tabla_actores):
    # Generar una asignación inicial aleatoria
    asignacion_actual = asignar_grabaciones_aleatorias()

    coste_actual = calcular_coste_total(asignacion_actual, tabla_actores)

    mejor_asignacion = asignacion_actual
    mejor_coste = coste_actual

    temperatura = temperatura_inicial

    while temperatura > temperatura_final:
        for _ in range(iteraciones_por_temperatura):
            nueva_asignacion = generar_vecino(asignacion_actual)
            nuevo_coste = calcular_coste_total(nueva_asignacion, tabla_actores)

            # Verificar si se acepta la asignación vecina utilizando la probabilidad de aceptación
            if probabilidad_aceptacion(coste_actual, nuevo_coste, temperatura) > random.random():

                asignacion_actual = nueva_asignacion
                coste_actual = nuevo_coste

            # Verificar si la nueva asignación tiene un mejor coste que la mejor asignación encontrada hasta el momento
            if nuevo_coste < mejor_coste:
                mejor_asignacion = nueva_asignacion
                mejor_coste = nuevo_coste

        # Enfriar la temperatura multiplicándola por el factor de enfriamiento
        temperatura *= factor_enfriamiento

    return mejor_asignacion


# Parámetros iniciales del algoritmo
temperatura_inicial = 10000
temperatura_final = 0.0000000000001
factor_enfriamiento = 0.3
iteraciones_por_temperatura = 10000

num_grabaciones = len(tabla_actores_np)
num_actores = len(tabla_actores_np[0])
```

```
dias_grabacion = recocido_simulado(temperatura_inicial, temperatura_final, factor_enfriamiento, iteraciones_por_temperatura, tabla_actores)

coste_total = calcular_coste_total(dias_grabacion, tabla_actores)

print("~ ORGANIZACIÓN DE TOMAS POR DIAS ~")
print("---------------------------------")

for i, dia in enumerate(dias_grabacion):
    actores_presentes = obtener_actores_presentes(dia, tabla_actores)

    if dia and actores_presentes:
        print(f"Día {i + 1} -------------------------\nGrabaciones: {', '.join(map(str, [grabacion + 1 for grabacion in dia]))}")
        print(f"Actores necesarios para grabar: {', '.join(map(str, actores_presentes))}")

print(f"\nEl coste total de la grabación es: {coste_total}")
```

```
 ~ ORGANIZACIÓN DE TOMAS POR DIAS ~
 ---------------------------------
 Día 1 -------------------------
 Grabaciones: 6, 22, 2, 20, 1, 12
 Actores necesarios para grabar: 1, 2, 3, 4, 5, 6
 Día 2 -------------------------
 Grabaciones: 4, 15, 3, 5, 28, 27
 Actores necesarios para grabar: 1, 2, 4, 5, 7, 8
 Día 3 -------------------------
 Grabaciones: 14, 24, 18, 17, 23, 19
 Actores necesarios para grabar: 1, 3, 6
 Día 4 -------------------------
 Grabaciones: 26, 10, 8, 11, 21, 29
 Actores necesarios para grabar: 1, 2, 3, 5, 6, 8, 9
 Día 5 -------------------------
 Grabaciones: 25, 16, 7, 13, 9
 Actores necesarios para grabar: 1, 2, 4, 5, 10

 El coste total de la grabación es: 27
```

Como se puede observar, mediante recocido simulado si que se obtiene una mejor solución al problema. Esto puede ser por diversos factores como:

- Adaptabilidad a diferentes problemas: El recocido simulado es una técnica general que se puede aplicar a una amplia variedad de problemas de optimización combinatoria.
- Flexibilidad del algoritmo de recocido simulado: Su flexibilidad permite explorar un amplio espacio de soluciones, lo que puede conducir a la identificación de soluciones de menor coste que otros métodos más deterministas.
- Capacidad para escapar de mínimos locales: El recocido simulado tiene la capacidad de escapar de estos mínimos locales mediante la aceptación de soluciones peores con cierta probabilidad. Esto permite explorar de manera más efectiva el espacio de búsqueda en busca de soluciones de menor coste.