03MIAR_10_A Algoritmos de Optimización

Trabajo Práctico

Máster en Inteligencia Artificial

Curso 2023-2024. Edición Oct-2023

Profr.: **Raúl Reyero**

- raul.reyero@professor.universidadviu.com (raul.reyero@professor.universidadviu.com)

Alumno: **Victor David Betancourt Leal**

- victor.betancourt@alumnos.viu.es (victor.betancourt@alumnos.viu.es)
- vbleal@gmail.com (vbleal@gmail.com)

Repositorio

- 📒 Notebook Colab:
- https://drive.google.com/file/d/1cudXzRH38F5bPLDwdR3C5rZiRM0RMSUP/view?usp=sharing (https://drive.google.com/file/d/1cudXzRH38F5bPLDwdR3C5rZiRM0RMSUP/view?usp=sharing)
- 🚀 Repositorio GitHub:
- https://github.com/vbleal/03MIAR (https://github.com/vbleal/03MIAR)

Índice

Problema: La Liga

Organizar los horarios de partidos de La Liga

Desde la **La Liga** de fútbol profesional se pretende organizar los horarios de los partidos de liga de de cada jornada. Se conocen algunos datos que nos deben llevar a diseñar un algoritmo que realice la asignación de los partidos a los horarios que maximice la audiencia.

Los horarios disponibles se conocen a priori y son los siguientes:

Día	Hora

Viernes	20
Sábado	12, 16, 18, 20
Domingo	12, 16, 18, 20

En primer lugar se clasifican los equipos en tres categorías según el numero de seguidores que tienen relación directa con la audiencia. Hay

- **3 equipos en la categoría A**
- **11 equipos de categoría B** y
- **6 equipos de categoría C**

Se conoce estadísticamente la audiencia que genera cada partido según los equipos que se enfrentan y en horario de sábado a las 20 horas (el mejor en todos los casos):

-	Categoría A	Categoría B	Categoría C
Categoría A	2 M	1.3 M	1 M
Categoría B	...	0.9 M	0.75 M
Categoría C	...	...	0.47 M

Si el horario del partido no se realiza a las 20 horas del sábado se sabe que **se reduce según los coeficientes** de la siguiente tabla:

-	Viernes	Sábado	Domingo	Lunes
12 h	...	0.55	0.45	...
16 h	...	0.7	0.75	...
18 h	...	0.8	0.85	...
20 h	0.4	1	1	0.4

Debemos asignar obligatoriamente siempre un partido el viernes y un partido el lunes.

Es posible la coincidencia de horarios pero en este caso la audiencia se verá afectada y se estima que se reduce en porcentaje según la siguiente tabla, dependiendo del número de coincidencias:

Coincidencia	-%
0	0%
1	25%
2	45%
3	60%
4	70%

5	75%
6	78%
7	80%
8	80%

.

Modelo

- ¿Cómo represento el espacio de soluciones?
- ¿Cuál es la función objetivo?
- ¿Cómo implemento las restricciones?

Espacio de Soluciones

El espacio de soluciones se puede representar como un conjunto de asignaciones de partidos a horarios disponibles que cumplan con las restricciones impuestas (como tener al menos un partido el viernes y el lunes). Cada solución es un conjunto de tuplas, donde cada tupla representa un partido específico, compuesto por dos equipos (con sus respectivas categorías) y el horario asignado para ese partido.

Una forma de representarlo podría ser mediante una estructura de datos que contenga los siguientes campos para cada partido:

- Categoría del equipo local
- Categoría del equipo visitante
- Día de la semana
- Hora del día

Dado que hay 20 equipos divididos en categorías A, B, y C, y un total de 10 slots horarios a lo largo de la jornada (1 el viernes, 4 el sábado, 4 el domingo, 1 el lunes), el espacio de soluciones es el conjunto de todas las combinaciones posibles de partidos (equipos enfrentados según categoría) asignados a estos slots, respetando las restricciones de al menos un partido por viernes y lunes.

Función Objetivo

La **función objetivo** es *maximizar la audiencia total* de todos los partidos de la jornada. Esta se calcula sumando las audiencias esperadas para cada partido, ajustadas por los coeficientes de horario y las penalizaciones por coincidencia de partidos en el mismo horario.

La audiencia esperada de un partido se determina a partir de la tabla de audiencias base (por categoría de enfrentamiento y horario óptimo), multiplicada por el coeficiente de ajuste del horario seleccionado, y finalmente ajustada por la penalización de coincidencia, si la hay.

Restricciones

Las restricciones se pueden implementar de las siguientes maneras:

- **Restricción de Partidos en Viernes y Lunes**: Se asegura programáticamente que al menos un partido sea asignado a estos días.
- **Capacidad del Horario**: Permitir que más de un partido se asigne al mismo horario pero aplicando la penalización de audiencia según el número de coincidencias.
- **Distribución de Equipos por Categoría**: Asegurar que los enfrentamientos se asignen correctamente según la categoría de los equipos, lo que afecta la base de la audiencia esperada.

Para la implementación, se puede utilizar una estructura de programación que itere sobre todas las posibles asignaciones de partidos a horarios, calculando la audiencia esperada para cada configuración y aplicando las restricciones mencionadas. Podría ser útil emplear técnicas de programación entera, algoritmos genéticos, o búsqueda local para explorar eficientemente el espacio de soluciones y encontrar una configuración óptima.

.

.

Análisis

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

Complejidad Computacional

Este problema es un tipo de problema de optimización combinatoria, específicamente se puede relacionar con el problema de programación de horarios, que es conocido por ser **NP-difícil (NP-hard)**.

La complejidad NP-hard indica que no se conoce un algoritmo que lo resuelva en tiempo polinomial en el tamaño de la entrada para el caso general. La dificultad radica en que el número de posibles combinaciones de asignaciones de partidos a horarios crece exponencialmente con el número de partidos a programar.

Orden de Complejidad

Para contabilizar el espacio de soluciones, consideremos los siguientes puntos:

- Hay 10 slots horarios disponibles distribuidos en 4 días.
- 20 equipos en total, divididos en 3 categorías (3 equipos en la categoría A, 11 en la B, y 6 en la C), lo que da lugar a 10 partidos por jornada (cada equipo juega contra otro).

Primero, calculamos las combinaciones posibles de partidos sin tener en cuenta los horarios, solo considerando los emparejamientos de equipos. Luego, asignamos estos partidos a los slots horarios disponibles.

Combinaciones de Partidos

Para simplificar el cálculo del espacio de soluciones, supongamos que nos enfocamos en cómo se pueden formar los partidos basándonos en las categorías, sin especificar qué equipos concretos de cada categoría se enfrentan. La cantidad de partidos únicos posibles depende de las combinaciones de enfrentamientos entre categorías, que ya están predefinidas (A vs. A, A vs. B, etc.).

Asignación a Slots Horarios

Cada uno de estos partidos puede ser asignado a cualquiera de los 10 slots horarios. La primera aproximación para calcular el espacio de soluciones sería 10^{10}, considerando que cada uno de los 10 partidos se puede asignar independientemente a 10 slots horarios. Sin embargo, este cálculo no tiene en cuenta las restricciones específicas (como la necesidad de asignar partidos específicamente el viernes y el lunes) y la posibilidad de que varios partidos compartan el mismo horario con penalizaciones por coincidencia.

Contabilizar el Espacio de Soluciones

Consideraciones

El cálculo es mucho más complejo debido a:

- La necesidad de asignar al menos un partido el viernes y el lunes.
- La posibilidad de que varios partidos ocurran al mismo tiempo, lo cual introduce una dimensión adicional en la forma en que se calculan las combinaciones, ya que no solo importa qué partidos se juegan, sino también cuántos partidos se programan simultáneamente en cada slot.

.

.

Diseño

- ¿Qué técnica utilizo? ¿Por qué?

Selección

Se utilizará un **Algoritmo Voraz (Greedy)**.

Los algoritmos voraces son particularmente adecuados para problemas de optimización, ya que construyen la solución paso a paso, seleccionando en cada paso la opción que parece ser la más óptima en ese momento, con la esperanza de llegar a una solución global óptima.

Este enfoque es mucho más eficiente en términos de tiempo de ejecución en comparación con explorar exhaustivamente todas las posibles asignaciones de partidos a horarios, como sería el caso en un ***algoritmo por Fuerza Bruta***.

Justificación

El algoritmo voraz es adecuado para este problema por varias razones:

- **Simplicidad**: Es conceptualmente más simple y más fácil de implementar que otros algoritmos de optimización como la programación dinámica o los algoritmos genéticos para este tipo de problema.
- **Eficiencia**: Tiene una eficiencia temporal relativamente buena en comparación con el enfoque de fuerza bruta, lo cual es crucial dado el gran espacio de soluciones.
- **Estructura del Problema**: La decisión de maximizar la audiencia en cada paso puede conducir a una solución bastante buena, especialmente si las decisiones se toman considerando las penalizaciones por coincidencias y los coeficientes de horario.

Algoritmo

El algoritmo se diseñará para seguir estos pasos:

1. **Priorización de Partidos**: Ordenar los partidos potenciales por su audiencia máxima esperada, dando prioridad a los que involucran equipos de categoría A.

2. **Asignación de Horarios**: Asignar los partidos a los horarios disponibles comenzando por los que tienen mayor audiencia esperada, ajustando por el coeficiente de horario para maximizar la audiencia total.

3. **Manejo de Coincidencias**: Evitar coincidencias cuando sea posible, pero si son inevitables (especialmente en horarios de alta demanda), asignar los partidos de manera que las penalizaciones por coincidencia impacten lo menos posible en la audiencia total.

Código Python

In [ ]:
```
# inportar librerias
import pandas as pd
import numpy as np
```

In [ ]:
```python
# Supuestos: Tenemos listas de partidos con sus audiencias esperadas en el mej
# y una función que calcula la audiencia ajustada por horario y penalizaciones

# Clasificación de partidos por audiencia esperada
def ordenar_partidos(partidos):
    # Supongamos que cada partido tiene la forma (categoriaA, categoriaB, audi
    # y los ordenamos por audiencia esperada de manera descendente.
    partidos_ordenados = sorted(partidos, key=lambda x: x[2], reverse=True)
    return partidos_ordenados

# Asignación de partidos a horarios
def asignar_partidos_a_horarios(partidos_ordenados, horarios, penalizaciones_p
    asignaciones = []  # Lista para guardar las asignaciones de partidos a hor
    audiencias = []  # Lista para guardar la audiencia esperada de cada asigna
    for partido in partidos_ordenados:
        mejor_horario, mejor_audiencia = None, 0
        for horario in horarios:
            # Calcula la audiencia ajustada por horario y penalizaciones
            audiencia_ajustada = calcular_audiencia_ajustada(partido, horario,
            if audiencia_ajustada > mejor_audiencia:
                mejor_horario = horario
                mejor_audiencia = audiencia_ajustada
        asignaciones.append((partido, mejor_horario))
        audiencias.append(mejor_audiencia)
        # Actualizar horarios y penalizaciones según la asignación
        actualizar_horarios_y_penalizaciones(mejor_horario, penalizaciones_por
    return asignaciones, sum(audiencias)
```

In [ ]:
```python
# Calcular Audiencia Ajustada
def calcular_audiencia_ajustada(partido, horario, penalizaciones_por_coinciden
    # Audiencias base según la categoría del partido en el mejor horario (sába
    audiencias_base = {
        ('A', 'A'): 2.0,
        ('A', 'B'): 1.3,
        ('A', 'C'): 1.0,
        ('B', 'B'): 0.9,
        ('B', 'C'): 0.75,
        ('C', 'C'): 0.47
    }

    # Coeficientes de ajuste por horario
    coeficientes_horario = {
        ('Viernes', 20): 0.4,
        ('Sábado', 12): 0.55,
        ('Sábado', 16): 0.7,
        ('Sábado', 18): 0.8,
        ('Sábado', 20): 1.0,
        ('Domingo', 12): 0.45,
        ('Domingo', 16): 0.75,
        ('Domingo', 18): 0.85,
        ('Domingo', 20): 1.0,
        ('Lunes', 20): 0.4
    }

    categorias = (partido[0], partido[1])
    audiencia_base = audiencias_base.get(categorias, 0) # Obtiene la audiencia
    coeficiente = coeficientes_horario.get(horario, 0) # Obtiene el coeficient

    # Calcula penalizaciones por coincidencia
    coincidencias = horarios_usados.get(horario, 0) # Número de partidos ya as
    penalizacion = penalizaciones_por_coincidencia.get(coincidencias, 1) # Obt

    # Calcula la audiencia ajustada
    audiencia_ajustada = audiencia_base * coeficiente * (1 - penalizacion)

    return audiencia_ajustada
```

In [ ]:
```python
# Actualizar Horarios y Penalizaciones
def actualizar_horarios_y_penalizaciones(horario, horarios_usados):
    # Añade o incrementa el contador de partidos asignados a este horario
    if horario in horarios_usados:
        horarios_usados[horario] += 1
    else:
        horarios_usados[horario] = 1
```

In [ ]:
```
def asignar_partidos_a_horarios(partidos_ordenados, horarios, penalizaciones_p
    asignaciones = []  # Lista para guardar las asignaciones de partidos a hor
    audiencia_total = 0  # Variable para sumar la audiencia total
    horarios_usados = {}  # Diccionario para llevar el registro de los horario

    for partido in partidos_ordenados:
        mejor_horario = None
        mejor_audiencia = 0
        for horario in horarios:
            # Calcula la audiencia ajustada para el partido en el horario actu
            audiencia_ajustada = calcular_audiencia_ajustada(partido, horario,
            # Si la audiencia ajustada es mayor que la mejor audiencia hasta a
            if audiencia_ajustada > mejor_audiencia:
                mejor_horario = horario
                mejor_audiencia = audiencia_ajustada
        if mejor_horario:  # Si se encontró un horario óptimo
            asignaciones.append((partido, mejor_horario))
            audiencia_total += mejor_audiencia
            # Actualiza los horarios usados con el mejor horario encontrado
            actualizar_horarios_y_penalizaciones(mejor_horario, horarios_usado
        else:
            # Si no se encontró un horario, añadir el partido sin horario asig
            asignaciones.append((partido, None))

    return asignaciones, audiencia_total
```

.

Ejemplo

In [ ]:
```
# Ejemplo
partidos = [('A', 'A', 2), ('A', 'B', 1.3), ('B', 'C', 0.75)]  # Ejemplo de fo
horarios = [('Viernes', 20), ('Sábado', 12), ('Domingo', 16)]  # Ejemplo de ho
penalizaciones_por_coincidencia = {0: 0, 1: 0.25, 2: 0.45}  # Ejemplo de penal

partidos_ordenados = ordenar_partidos(partidos)
asignaciones, audiencia_total = asignar_partidos_a_horarios(partidos_ordenados

print("Asignaciones:", asignaciones)
print("Audiencia total esperada:", audiencia_total)
```

```
Asignaciones: [(('A', 'A', 2), ('Domingo', 16)), (('A', 'B', 1.3), ('Doming
o', 16)), (('B', 'C', 0.75), ('Sábado', 12))]
Audiencia total esperada: 2.6437500000000003
```

Resultado

Categoría Equipo Local	Categoría Equipo Visitante	Audiencia Esperada (M)	Horario Asignado
A	A	2	Domingo, 16
A	B	1.3	Domingo, 16

Categoría Equipo Local	Categoría Equipo Visitante	Audiencia Esperada (M)	Horario Asignado

.

Otro Ejemplo

In [ ]:
```
# Ejemplo alternativo
# Nuevo conjunto de partidos
partidos = [('A', 'C', 1.5), ('B', 'B', 0.8), ('C', 'C', 0.5), ('A', 'B', 1.4)
# Nuevos horarios
horarios = [('Viernes', 20), ('Sábado', 18), ('Domingo', 12), ('Domingo', 20)]
# Penalizaciones ajustadas (ejemplo)
penalizaciones_por_coincidencia = {0: 0, 1: 0.2, 2: 0.4, 3: 0.6}

partidos_ordenados = ordenar_partidos(partidos)
asignaciones, audiencia_total = asignar_partidos_a_horarios(partidos_ordenados

print("Asignaciones:", asignaciones)
print("Audiencia total esperada:", audiencia_total)
```

```
Asignaciones: [(('A', 'C', 1.5), ('Domingo', 20)), (('A', 'B', 1.4), ('Sábad
o', 18)), (('B', 'B', 0.8), ('Domingo', 20)), (('C', 'C', 0.5), ('Sábado', 1
8))]
Audiencia total esperada: 3.0608000000000004
```

Resultado

Categoría Equipo Local	Categoría Equipo Visitante	Audiencia Esperada (M)	Horario Asignado
A	C	1.5	Domingo, 20
A	B	1.4	Sábado, 18
B	B	0.8	Domingo, 20
C	C	0.5	Sábado, 18

.

.

Análisis Final de la Complejidad

Determinando la Complejidad en cada paso:

- **Ordenar Partidos por Audiencia Esperada**: El paso de ordenar los partidos por su audiencia esperada utiliza un algoritmo de ordenamiento, que en el caso de Python (usando `sorted` sobre listas) típicamente tiene una complejidad de ***O(n log n)***, donde ***n*** es el número de partidos.

- **Asignación de Partidos a Horarios**: La complejidad de este paso depende de dos factores principales:
- El número de partidos ***n***.
- El número de horarios ***m***.

Para cada uno de los ***n*** partidos, el algoritmo itera sobre cada uno de los ***m*** horarios disponibles para calcular la audiencia ajustada. Esto da como resultado una complejidad de ***O(n· m)*** para este paso.

- **Calcular Audiencia Ajustada**: La función calcular_audiencia_ajustada se llama ***n· m*** veces, pero su complejidad interna es mayormente constante ***O(1)***, ya que realiza operaciones de búsqueda en diccionarios y cálculos aritméticos simples. Sin embargo, la frecuencia de llamadas no altera el orden de complejidad global mencionado anteriormente.
- **Actualizar Horarios y Penalizaciones**: La operación de actualización dentro de esta función se ejecuta cada vez que se asigna un partido a un horario. Aunque la función se llama ***n*** veces (una por cada partido), las operaciones dentro de la función son de tiempo

Complejidad total:

- La ***complejidad total*** del algoritmo es:

O(n log n + n· m)

.

.