Algoritmos de optimización - Trabajo Práctico

Nombre y Apellidos: José Jesús La Casa Nieto Url: https://github.com/JoseJesusLaCasaNieto/03MIAR---Algoritmos-de-Optimizacion---2024/tree/main/Trabajo_Practico Google Colab: https://colab.research.google.com/drive/1U9wUobh8NYpL9eRDLN_GsSY71oOCQUVC?usp=sharing

Problemas:

1. Sesiones de doblaje
2. Organizar los horarios de partidos de La Liga
3. Configuración de Tribunales

Descripción del problema:

Problema 2. 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 cada jornada. Se conocen algunos datos que nos deben llevar a diseñar un algoritmo que realice la asignación de los partidos a los horarios de forma que maximice la audiencia.

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

Viernes - 20:00 / Sábado - 12:00, 16:00, 18:00, 20:00 / Domingo - 12:00, 16:00, 18:00, 20:00 / Lunes - 20:00

```
# Diccionario con los horarios de la jornada
horarios = {
    "Viernes": [20],
    "Sábado": [12, 16, 18, 20],
    "Domingo": [12, 16, 18, 20],
    "Lunes": [20]
}
```

En primer lugar se clasifican los equipos en tres categorías según el numero de seguidores(que tiene 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 20h (el mejor en todos los casos).

```
# Audiencia que se generan por encuentro en el mejor horario
disponible
audiencia_por_partido = {
    ("A", "A"): 2,
    ("A", "B"): 1.3,
    ("A", "C"): 1,
```

```
    ("B", "A"): 1.3,
    ("B", "B"): 0.9,
    ("B", "C"): 0.75,
    ("C", "A"): 1,
    ("C", "B"): 0.75,
    ("C", "C"): 0.47
}
```

Si el horario del partido no se realiza a las 20:00 horas del sábado se sabe que se reduce según los coeficientes de una tabla.

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

```
# Reducción de los espectadores por horario
reduccion_por_horario = {
    "V20": 0.4,
    "S12": 0.55,
    "S16": 0.7,
    "S18": 0.8,
    "S20": 1,
    "D12": 0.45,
    "D16": 0.75,
    "D18": 0.85,
    "D20": 1,
    "L20": 0.4
}
```

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

Los cálculos asociados a una jornada de ejemplo se realizan según se muestra en la siguiente tabla:

```
# Penalizaciones por coincidencias de horarios
ponderacion_por_coincidencias = {
    1: 1,
    2: 0.75,
    3: 0.55,
    4: 0.4,
    5: 0.3,
    6: 0.25,
    7: 0.22,
    8: 0.2,
    9: 0.2
}
```

Modelo

- ¿Cómo represento el espacio de soluciones?

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

```
#Respuesta
```

La representación del espacio de soluciones que he realizado es a través de un diccionario de listas de listas. En un principio no era esa mi idea, pretendía una alternativa algo más simple. Sin embargo, a lo largo del desarrollo del código he visto necesario utilizar un diccionario con estas características porque se pueden manejar bien a la hora de realizar operaciones y, además, te están proporcionando constantemente información sobre los partidos y horarios. De esta manera, es fácil releer el problema y cambiar cualquier cosa que se quiera en el futuro con facilidad.

Cada una de esas listas, es una lista de listas, que incluye el partido correspondiente en una tupla y el número de espectadores que tendrá el partido una vez se hayan aplicado todas las variables del problema. Con variables del problema me refiero a los espectadores que verán el partido por la categoría de los equipos, las penalizaciones por partidos duplicados y la ponderación por el horario.

- ¿Cuál es la función objetivo?

```
# Respuesta
```

La función objetivo es aquella que nos devuelva un valor que representa el evento identificado por diversas variables de entrada. En este problema en particular, la función objetivo depende de una serie de variables de entrada como son los espectadores según las categorías de los equipos que se enfrentan, la variable horario y la variable coincidencia de partidos. Lo que buscamos con esa función objetivo es maximizarla para los datos de entrada. De esa manera, obtendremos el mayor número de espectadores posibles.

La función objetivo es:

$$f = \sum_{i=1}^{n} x_i \cdot y_i \cdot z_i$$

La variable 'x' de la función es la base de espectadores para el partido según la categoría de los equipos y el mejor horario posible (es sobre las 20:00 del sábado o domingo), siendo la variable horario máxima. La variable 'y' se corresponde con la reducción por horario y la variable 'z' es la ponderación por coincidencias. La variable 'z' (representada en el diccionario 'ponderacion_por_coincidencias') la he ajustado invirtiendo los porcentajes, para que, al multiplicarla por 'x' e 'y', obtener como resultado los espectadores totales para ese partido con todas las variables aplicadas.

```
# Diccionarios de las variables de la función objetivo
audiencia_por_partido = {
    ("A", "A"): 2,
    ("A", "B"): 1.3,
    ("A", "C"): 1,
    ("B", "A"): 1.3,
```

```
    ("B", "B"): 0.9,
    ("B", "C"): 0.75,
    ("C", "A"): 1,
    ("C", "B"): 0.75,
    ("C", "C"): 0.47
}

reduccion_por_horario = {
    "V20": 0.4,
    "S12": 0.55,
    "S16": 0.7,
    "S18": 0.8,
    "S20": 1,
    "D12": 0.45,
    "D16": 0.75,
    "D18": 0.85,
    "D20": 1,
    "L20": 0.4
}

ponderacion_por_coincidencias = {
    1: 1,
    2: 0.75,
    3: 0.55,
    4: 0.4,
    5: 0.3,
    6: 0.25,
    7: 0.22,
    8: 0.2,
    9: 0.2
}
```

- ¿Cómo implemento las restricciones?

```
# Respuesta
```

Las restricciones de este problema son que la asignación de partidos tiene que tener, sí o sí, partidos los lunes y viernes. Para entender la implementación de las restricciones, hay que saber las posibilidades sobre las que se van a aplicar las restricciones. La forma en la que yo he planteado el problema es que disponemos de 10 horarios. Esos horarios se pueden repetir, no es necesario utilizar todos y sí importa el orden. Por tanto, las posibilidades del problema se rigen bajo una variación con repetición. Además, tenemos 10 partidos (al tratarse de 20 equipos en total) que se deben ajustar a los horarios disponibles, aunque no sea necesario utilizar todos los horarios.

De esta manera, todas las posibilidades que tenemos en el problema son:

$$V R_{10,10} = 10^{10} = 10.000.000.000$$

Si aplicamos las restricciones de jugar partidos los viernes y los sábados, se reducen el número de posibildades sobre unas 4 mil millones. Esto es porque sí o sí se tienen que utilizar dos horarios, siendo los otros 8 sometidos a una variación con repetición, puesto que no es necesario tener que utilizar todos los valores posibles, es importante el orden y se pueden repetir.

$$10^{10} - 2*9^{10} + 8^{10} = 4.100.173.022$$

En mi código he aplicado una condición para seleccionar, dentro de las posibilidades proporcionadas por la variación con repetición, aquellas opciones en las que se encuentren el horario de viernes y el del lunes. A continuación, un extracto del código en el que aplica la restricción.

```
# Implementación de la restricción en código
variaciones_con_repeticion_2 = []
for item in variaciones_con_repeticion:
    if 'V20' in item and 'L20' in item:
        variaciones_con_repeticion_2.append(item)
```

Análisis

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

```
#Respuesta
```

El análisis de la complejidad del problema la he hecho mediante un algoritmo de fuerza bruta. Así, podemos analizar la peor situación posible en términos de complejidad. De esta manera, los algoritmos que se intenten implementar posteriormente, serán mejores computacionalmente hablando.

A continuación, vamos a analizar el orden de complejidad del problema poco a poco, viendo cada función por separado.

```
# Función generar jornada
def generar_jornada(equipos_por_categoria, numero_partidos):
    todos_los_equipos = []
    for equipos in equipos_por_categoria.values():
      todos_los_equipos.extend(equipos)
    jornada = []
    while len(jornada) < numero_partidos:
      equipo_1 = random.choice(todos_los_equipos)
      todos_los_equipos.remove(equipo_1)
      equipo_2 = random.choice(todos_los_equipos)
      todos_los_equipos.remove(equipo_2)
      jornada.append([equipo_1, equipo_2])
    return jornada
```

La función se encarga de formar los partidos haciendo una selección de 2 en 2 de entre todos los equipos introducidos. Consta de dos procesos:

1. **Recopilación de todos los equipos**: En este paso, se recorren todos los valores del diccionario 'equipos_por_categoria' y se concatenan todos los equipos en una lista. Este paso tiene un coste computacional de O(n), siendo 'n' la suma de la cantidad de equipos introducidos en la función.

2. **Selección de equipos por cada partido**: Se seleccionan dos equipos aleatorios de la lista anterior, hasta que se hayan formado todos los partidos posibles (n/2). Al formar cada partido por dos equipos y luego eliminar los equipos de la lista, la complejidad de la operación es de O(k), siendo 'k' el número de partidos a generar.

Por lo tanto, el orden de complejidad de la función quedaría:

$$O(n+k) = O(n+n/2)$$

```
# Función generar categorías
def generar_categorias(jornada, dct):
    def categoria(equipo):
      for categoria, equipos in dct.items():
        if equipo in equipos:
          return categoria
      return None

    jornada_por_categoria = []
    for partido in jornada:
      partido_categoria = [categoria(equipo) for equipo in partido]
      jornada_por_categoria.append(partido_categoria)
    return jornada_por_categoria
```

Esta función se encarga de aplicar la categoría de los equipos a los partidos formados en la función anterior. Este paso se realiza principalmente para poder aplicarle la base de espectadores en el algoritmo que queramos implementar.

1. **Asignación de categoría a cada equipo por partido**: En esta parte de la función, para cada equipo en cada partido, iteramos sobre todas las categorías del diccionario 'dct' para encontrar la categoría a la que pertenece el equipo. Esto implica una iteración sobre todas las categorías para cada equipo en cada partido, lo que resulta en una complejidad de O(k*n) donde 'k' es el número de partidos y 'n' el número de categorías.

2. **Construcción de la lista 'jornada_por_categoria'**: Después de determinar la categoría para cada equipo en cada partido, construimos una lista de listas 'jornada_por_categoria' que contiene estas categorías. Esto simplemente implica un recorrido lineal de la lista 'jornada', que tiene un coste de O(k), donde 'k' es el número de partidos.

Por lo tanto, el orden de complejidad total de la función es O(k*n), siendo 'k' el número de partidos y 'n' el número de categorías del diccionario 'dct'.

```
# Función algoritmo de fuerza bruta
def algoritmo_fuerza_bruta(jornada, audiencia_por_partido,
reduccion_por_horario, ponderacion_por_coincidencias):
    jornada_espectadores = {}
    for partido, partido_categorias in jornada.items():
       partido_categorias = tuple(partido_categorias)
       if partido_categorias in audiencia_por_partido:
          jornada_espectadores[partido] =
audiencia_por_partido[partido_categorias]

    horarios = list(reduccion_por_horario.keys())

    longitud_variacion = 8
    variaciones_con_repeticion = list(product(horarios,
repeat=longitud_variacion))

    variaciones_con_repeticion_2 = []
    for item in variaciones_con_repeticion:
       if 'V20' in item and 'L20' in item:
          variaciones_con_repeticion_2.append(item)

    resultado = {}
    mejor_resultado = 0
    for jornada in variaciones_con_repeticion_2:
       resultado_jornada  = {}
       for horario, (partido, espectadores) in zip(jornada,
jornada_espectadores.items()):
          lista_partido_espectadores = [partido, espectadores *
reduccion_por_horario[horario]]
          if horario in resultado_jornada:

resultado_jornada[horario].append(lista_partido_espectadores)
          else:
             resultado_jornada[horario] = [lista_partido_espectadores]

       for lista_listas in resultado_jornada.values():
          if len(lista_listas) in ponderacion_por_coincidencias and
len(lista_listas) > 1:
             for lista in lista_listas:
                lista[1] *=
ponderacion_por_coincidencias[len(lista_listas)]

       suma_total = 0
       for lista_listas in resultado_jornada.values():
          for lista in lista_listas:
             suma_total += lista[1]

       if suma_total > mejor_resultado:
          mejor_resultado = suma_total
          resultado = resultado_jornada
```

```
    return resultado, mejor_resultado
```

Esta función utiliza como datos de entrada los diccionarios que hemos analizado en los puntos anteriores: 'audiencia_por_partido', 'reduccion_por_horario' y 'ponderacion_por_coincidencias'. Además, se introduce la jornada de partidos y categorías que se han obtenido con las otras funciones.

1. **Recopilación de información sobre la audiencia por partido**: Recorre todos los partidos en la jornada y verifica si la combinación de categorías del partido está presente en el diccionario 'audiencia_por_partido'. Esta operación tiene un coste de O(k), donde 'k' es el número de partidos en la jornada.

2. **Generación de todas las posibles variaciones con repetición de horarios**: Genera todas las posibles variaciones con repetición de longitud 'n' (en el código 'n' vale 8 por una cuestión de límite computacional para generar las opciones. Realmente, deberían ser 10 porque son los horarios totales posibles) utilizando la función 'product()' de 'itertools'. Dado que hay 'm' partidos y 'n' horarios, el número total de variaciones con repetición es:

$$O\left(n^m\right)$$

1. **Filtrado de variaciones de horarios**: Filtra las variaciones generadas para incluir solo aquellas que contienen tanto 'V20' como 'L20'. Este paso tiene un costo constante.

2. **Cálculo del resultado para cada variación de horarios**: Para cada variación de horarios filtrada, calcula el resultado total sumando la audiencia ponderada de los partidos en cada horario. El costo total de este paso depende del número de variaciones de horarios consideradas.

En resumen, el algoritmo de fuerza bruta tiene un orden de complejidad aproximado de:

$$O\left(k * n^m\right)$$

Esto significa que el tiempo de ejecución del algoritmo aumenta exponencialmente con respecto al número de horarios disponibles.

La contabilización del espacio de soluciones se ha analizado en apartados anteriores. Por resumir, se aplica una variación con repetición de 10 partidos y 10 horarios, a los que hay que seleccionar las opciones que incluyan 'V20' y 'L20'. El total de espacio de soluciones es:

$$10^{10} - 2 * 9^{10} + 8^{10} = 4.100.173.022$$

Diseño

- ¿Que técnica utilizo? ¿Por qué?

```
# Respuesta
```

Algoritmo de Fuerza Bruta

La primera técnica que utilizo es un algoritmo de fuerza bruta. Principalmente, la he utilizado porque quería comprobar el límite computacional en la ejecución del código para el peor caso posible. Un algoritmo de fuerza bruta es una técnica de resolución de problemas que implica probar sistemáticamente todas las posibles soluciones para encontrar la que cumpla con ciertos criterios o restricciones. Es una estrategia simple pero poderosa, aunque son impracticables para problemas con grandes conjuntos de datos debido a su alto tiempo de ejecución, como sucede en este problema.

Como era de esperar, para un problema con 10 mil millones de posibilidades, el ordenador no ha sido capaz de ejecutar todos los posibles resultados. Es por ello, que he tenido que reducir el número de horarios hasta poder tener una solución por parte del algoritmo. He priorizado dejar los 10 partidos de la jornada. Sin embargo, el límite lo he ido poniendo en los horarios. Para 9 y 10 horarios (variable 'longitud_variacion' en la función 'algoritmo_fuerza_bruta'), el ordenador no es capaz de proporcionarme una solución. Sin embargo, para una cantidad de 8 horarios sí que he podido proporcionar resultados del número de espectadores. Suele tardar sobre un minuto y medio, aunque en Google Colab no es capaz de ejecutarse.

A continuación, voy a proporcionar un resumen sobre el algoritmo, porque ya se ha explicado en el apartado anterior de coste computacional. En primer lugar, genero los 10 partidos de forma aleatoria con la función 'generar_jornada'. Después, se aplica la función 'generar_categorias' a los partidos obtenidos, y así saber qué base de espectadores tendrá cada partido. Los partidos y sus categorías se incluyen en un diccionario para proporcionárselos al algoritmo de fuerza bruta. El algoritmo genera todas las posibilidades, trabajando con los horarios y las restricciones, y muestra como salida la jornada que más espectadores totales ha encontrado.

```
# Algoritmo de Fuerza Bruta
from itertools import product
import random
import time

def algoritmo_fuerza_bruta(jornada, audiencia_por_partido,
reduccion_por_horario, ponderacion_por_coincidencias):
    jornada_espectadores = {}
    for partido, partido_categorias in jornada.items():
       partido_categorias = tuple(partido_categorias)
       if partido_categorias in audiencia_por_partido:
          jornada_espectadores[partido] =
audiencia_por_partido[partido_categorias]

    horarios = list(reduccion_por_horario.keys())

    longitud_variacion = 8
    variaciones_con_repeticion = list(product(horarios,
repeat=longitud_variacion))

    variaciones_con_repeticion_2 = []
    for item in variaciones_con_repeticion:
       if 'V20' in item and 'L20' in item:
```

```
          variaciones_con_repeticion_2.append(item)

    resultado = {}
    mejor_resultado = 0
    for jornada in variaciones_con_repeticion_2:
       resultado_jornada  = {}
       for horario, (partido, espectadores) in zip(jornada,
jornada_espectadores.items()):
          lista_partido_espectadores = [partido, espectadores *
reduccion_por_horario[horario]]
          if horario in resultado_jornada:

resultado_jornada[horario].append(lista_partido_espectadores)
          else:
             resultado_jornada[horario] = [lista_partido_espectadores]

       for lista_listas in resultado_jornada.values():
          if len(lista_listas) in ponderacion_por_coincidencias and
len(lista_listas) > 1:
             for lista in lista_listas:
                lista[1] *=
ponderacion_por_coincidencias[len(lista_listas)]

       suma_total = 0
       for lista_listas in resultado_jornada.values():
          for lista in lista_listas:
             suma_total += lista[1]

       if suma_total > mejor_resultado:
          mejor_resultado = suma_total
          resultado = resultado_jornada

    return resultado, mejor_resultado

def generar_jornada(equipos_por_categoria, numero_partidos):
    todos_los_equipos = []
    for equipos in equipos_por_categoria.values():
      todos_los_equipos.extend(equipos)
    jornada = []
    while len(jornada) < numero_partidos:
      equipo_1 = random.choice(todos_los_equipos)
      todos_los_equipos.remove(equipo_1)
      equipo_2 = random.choice(todos_los_equipos)
      todos_los_equipos.remove(equipo_2)
      jornada.append([equipo_1, equipo_2])
    return jornada

def generar_categorias(jornada, dct):
    def categoria(equipo):
      for categoria, equipos in dct.items():
```

```
        if equipo in equipos:
          return categoria
      return None

    jornada_por_categoria = []
    for partido in jornada:
      partido_categoria = [categoria(equipo) for equipo in partido]
      jornada_por_categoria.append(partido_categoria)
    return jornada_por_categoria

inicio = time.time()

equipos_por_categoria = {
    "A": ["Real Madrid", "Barcelona", "Atlético de Madrid"],
    "B": ["Real Sociedad", "Celta", "Valencia", "Athletic Club",
"Villarreal", "Alavés", "Levante", "Espanyol", "Sevilla", "Betis",
"Getafe"],
    "C": ["Mallorca", "Eibar", "Leganés", "Osasuna", "Granada",
"Valladolid"]
}

audiencia_por_partido = {
    ("A", "A"): 2,
    ("A", "B"): 1.3,
    ("A", "C"): 1,
    ("B", "A"): 1.3,
    ("B", "B"): 0.9,
    ("B", "C"): 0.75,
    ("C", "A"): 1,
    ("C", "B"): 0.75,
    ("C", "C"): 0.47
}

reduccion_por_horario = {
    "V20": 0.4,
    "S12": 0.55,
    "S16": 0.7,
    "S18": 0.8,
    "S20": 1,
    "D12": 0.45,
    "D16": 0.75,
    "D18": 0.85,
    "D20": 1,
    "L20": 0.4
}

ponderacion_por_coincidencias = {
    1: 1,
    2: 0.75,
    3: 0.55,
```

```
    4: 0.4,
    5: 0.3,
    6: 0.25,
    7: 0.22,
    8: 0.2,
    9: 0.2
}

jornada = generar_jornada(equipos_por_categoria, 10)
print("Jornada generada aleatoriamente:\n{}".format(jornada))

jornada_por_categorias = generar_categorias(jornada,
equipos_por_categoria)
# print("\nJornada anterior por categorías de equipos:\
n{}".format(jornada_por_categorias))

jornada_diccionario = {tuple(equipo): categoria for equipo, categoria
in zip(jornada, jornada_por_categorias)}
# print("\nJornada diccionario:\n{}".format(jornada_diccionario))

resultado, espectadores = algoritmo_fuerza_bruta(jornada_diccionario,
audiencia_por_partido, reduccion_por_horario,
ponderacion_por_coincidencias)
print(f"\nResultado:\n{resultado}")
print(f"\nEspectadores: {espectadores:.2f} millones")

fin = time.time()

tiempo_transcurrido = fin - inicio
print("\nTiempo transcurrido:", round(tiempo_transcurrido, 2),
"segundos")
```

```
Jornada generada aleatoriamente:
[['Leganés', 'Sevilla'], ['Osasuna', 'Valladolid'], ['Villarreal',
'Athletic Club'], ['Betis', 'Alavés'], ['Espanyol', 'Barcelona'],
['Real Madrid', 'Valencia'], ['Mallorca', 'Getafe'], ['Celta',
'Eibar'], ['Granada', 'Atlético de Madrid'], ['Levante', 'Real
Sociedad']]

Resultado:
{'V20': [[('Leganés', 'Sevilla'), 0.30000000000000004]], 'L20':
[[('Osasuna', 'Valladolid'), 0.188]], 'S18': [[('Villarreal',
'Athletic Club'), 0.7200000000000001]], 'D18': [[('Betis', 'Alavés'),
0.765]], 'S20': [[('Espanyol', 'Barcelona'), 1.3]], 'D20': [[('Real
Madrid', 'Valencia'), 1.3]], 'S16': [[('Mallorca', 'Getafe'),
0.5249999999999999]], 'D16': [[('Celta', 'Eibar'), 0.5625]]}

Espectadores: 5.66 millones

Tiempo transcurrido: 80.95 segundos
```

Algoritmo de Búsqueda aleatoria

Un algoritmo de búsqueda aleatoria es una técnica de resolución de problemas que utiliza un enfoque de búsqueda basado en la generación aleatoria de soluciones. A diferencia de los algoritmos de fuerza bruta, que consideran sistemáticamente todas las posibles soluciones, un algoritmo de búsqueda aleatoria selecciona soluciones al azar y las evalúa para ver si cumplen con los criterios de optimización del problema.

Los algoritmos de búsqueda aleatoria son útiles cuando el espacio de búsqueda es grande y no es factible o práctico explorar todas las posibles soluciones, como sucede en nuestro problema. Sin embargo, su eficacia depende en gran medida de la calidad de la función de evaluación y de la capacidad de generar soluciones aleatorias que sean prometedoras en términos de los criterios de optimización del problema.

Este algoritmo lo utilizo porque tiene una implementación sencilla y, aunque creo que no es del todo bueno, puede proporcionar soluciones factibles sobre todo el espectro de soluciones. Aquí, por ejemplo, sí que se pueden utilizar todos los horarios y todos los partidos posibles, a diferencia del algoritmo de fuerza bruta que no era capaz de proporcionar ninguna solución.

```
# Algoritmo de Búsqueda Aleatoria
import random
import time

def busqueda_aleatoria(n, equipos_por_categoria,
audiencia_por_partido, reduccion_por_horario,
ponderacion_por_coincidencias):
    jornada = generar_jornada(equipos_por_categoria, 10)
    jornada_por_categorias = generar_categorias(jornada,
equipos_por_categoria)
    jornada_diccionario = {tuple(equipo): categoria for equipo,
categoria in zip(jornada, jornada_por_categorias)}

    jornada_espectadores = {}
    for partido, partido_categorias in jornada_diccionario.items():
       partido_categorias = tuple(partido_categorias)
       if partido_categorias in audiencia_por_partido:
          jornada_espectadores[partido] =
audiencia_por_partido[partido_categorias]

    lista_horarios = list(reduccion_por_horario.keys())

    resultado = {}
    mejor_resultado = 0
    i = 0
    while i < n:
      horarios = generar_solucion(lista_horarios, 10)
      resultado_solucion = {}
      for horario, (partido, espectadores) in zip(horarios,
jornada_espectadores.items()):
        lista_partido_espectadores = [partido, espectadores *
```

```
reduccion_por_horario[horario]]
        if horario in resultado_solucion:

resultado_solucion[horario].append(lista_partido_espectadores)
        else:
            resultado_solucion[horario] = [lista_partido_espectadores]

      for lista_listas in resultado_solucion.values():
            if len(lista_listas) in ponderacion_por_coincidencias and
len(lista_listas) > 1:
              for lista in lista_listas:
                  lista[1] *=
ponderacion_por_coincidencias[len(lista_listas)]

      suma_total = 0
      for lista_listas in resultado_solucion.values():
        for lista in lista_listas:
          suma_total += lista[1]

      if suma_total > mejor_resultado:
        mejor_resultado = suma_total
        resultado = resultado_solucion

      i += 1

    return resultado, mejor_resultado, jornada

def generar_jornada(equipos_por_categoria, numero_partidos):
    todos_los_equipos = []
    for equipos in equipos_por_categoria.values():
      todos_los_equipos.extend(equipos)
    jornada = []
    while len(jornada) < numero_partidos:
      equipo_1 = random.choice(todos_los_equipos)
      todos_los_equipos.remove(equipo_1)
      equipo_2 = random.choice(todos_los_equipos)
      todos_los_equipos.remove(equipo_2)
      jornada.append([equipo_1, equipo_2])
    return jornada

def generar_categorias(jornada, dct):
    def categoria(equipo):
      for categoria, equipos in dct.items():
        if equipo in equipos:
          return categoria
      return None

    jornada_por_categoria = []
    for partido in jornada:
      partido_categoria = [categoria(equipo) for equipo in partido]
```

```
      jornada_por_categoria.append(partido_categoria)
    return jornada_por_categoria

def generar_solucion(conjunto, longitud_variacion):
   while True:
     variacion = list(random.choice(conjunto) for _ in
range(longitud_variacion))
     if 'V20' in variacion and 'L20' in variacion:
       return variacion

inicio = time.time()

equipos_por_categoria = {
    "A": ["Real Madrid", "Barcelona", "Atlético de Madrid"],
    "B": ["Real Sociedad", "Celta", "Valencia", "Athletic Club",
"Villarreal", "Alavés", "Levante", "Espanyol", "Sevilla", "Betis",
"Getafe"],
    "C": ["Mallorca", "Eibar", "Leganés", "Osasuna", "Granada",
"Valladolid"]
}

audiencia_por_partido = {
    ("A", "A"): 2,
    ("A", "B"): 1.3,
    ("A", "C"): 1,
    ("B", "A"): 1.3,
    ("B", "B"): 0.9,
    ("B", "C"): 0.75,
    ("C", "A"): 1,
    ("C", "B"): 0.75,
    ("C", "C"): 0.47
}

reduccion_por_horario = {
    "V20": 0.4,
    "S12": 0.55,
    "S16": 0.7,
    "S18": 0.8,
    "S20": 1,
    "D12": 0.45,
    "D16": 0.75,
    "D18": 0.85,
    "D20": 1,
    "L20": 0.4
}

ponderacion_por_coincidencias = {
    1: 1,
    2: 0.75,
    3: 0.55,
```

```
    4: 0.4,
    5: 0.3,
    6: 0.25,
    7: 0.22,
    8: 0.2,
    9: 0.2
}

resultado, espectadores, jornada = busqueda_aleatoria(10000000,
equipos_por_categoria, audiencia_por_partido, reduccion_por_horario,
ponderacion_por_coincidencias)

print("Jornada generada aleatoriamente:\n{}".format(jornada))
print(f"\nResultado:\n{resultado}")
print(f"\nEspectadores: {espectadores:.2f} millones")

fin = time.time()

tiempo_transcurrido = fin - inicio
print("\nTiempo transcurrido:", round(tiempo_transcurrido, 2),
"segundos")
```

```
Jornada generada aleatoriamente:
[['Sevilla', 'Athletic Club'], ['Celta', 'Valencia'], ['Villarreal',
'Mallorca'], ['Leganés', 'Barcelona'], ['Real Sociedad', 'Espanyol'],
['Granada', 'Valladolid'], ['Alavés', 'Levante'], ['Osasuna',
'Eibar'], ['Betis', 'Real Madrid'], ['Getafe', 'Atlético de Madrid']]

Resultado:
{'S18': [[('Sevilla', 'Athletic Club'), 0.7200000000000001]], 'D18':
[[('Celta', 'Valencia'), 0.765]], 'D12': [[('Villarreal', 'Mallorca'),
0.3375]], 'D16': [[('Leganés', 'Barcelona'), 0.75]], 'S12': [[('Real
Sociedad', 'Espanyol'), 0.49500000000000005]], 'V20': [[('Granada',
'Valladolid'), 0.188]], 'S16': [[('Alavés', 'Levante'), 0.63]], 'L20':
[[('Osasuna', 'Eibar'), 0.188]], 'S20': [[('Betis', 'Real Madrid'),
1.3]], 'D20': [[('Getafe', 'Atlético de Madrid'), 1.3]]}

Espectadores: 6.67 millones

Tiempo transcurrido: 89.19 segundos
```

La implementación que he realizado para el algoritmo de búsqueda aleatoria, consiste en generar una jornada de forma aleatoria y añadirle los espectadores que tendría cada partido en hora punta, en un diccionario. Posteriormente, comienzo un bucle que se repetirá tantas iteraciones como se quiera (en el código he utilizado 10.000.000 de iteraciones). Consiste en generar aleatoriamente un único horario para los partidos, aplicarle las penalizaciones por coincidencias y horarios, y luego sumar el número de espectadores de esa jornada. Este proceso se repetirá tantas veces como iteraciones le hayamos señalado a la función, escogiendo el mayor número de espectadores de todas las jornadas generadas.

De esta manera, se ha conseguido implementar un algoritmo de búsqueda aleatoria para resolver el problema de organización de horarios de partidos de La Liga. Este algoritmo proporciona buenas resoluciones y en tiempos de ejecución relativamente cortos. Además, tiene la ventaja con respecto del algoritmo de fuerza bruta, de generar horarios para todos los partidos, sin dejar ninguno fuera por tema de coste computacional.