Sorteo de la UEFA Champions League

Ana Paula González Muñoz
Dennis Daniel González Durán
C412

1 Descripción del problema

La UEFA Champions League es una de las competiciones deportivas más prestigiosas y seguidas a nivel mundial. La forma en que se asignan los equipos a los grupos y se diseñan los enfrentamientos juega un papel crucial en garantizar el equilibrio competitivo, la equidad y la rentabilidad del evento. Un sorteo desbalanceado puede resultar en grupos o eliminatorias poco competitivas, reduciendo el interés de los aficionados y afectando la equidad del torneo. En contraste, un sorteo equilibrado que considere factores como rivalidades históricas, diversidad geográfica y calidad competitiva puede maximizar la emoción de la temporada, promoviendo la equidad deportiva y optimizando la experiencia de los aficionados.

El presente informe tiene como objetivo modelar el problema del sorteo de la UEFA Champions League bajo un criterio de optimización dual: maximizar la ganancia económica mientras se garantiza que la desigualdad entre los grupos no supere un umbral dado. Además, se demostrará que esta formulación conduce a un problema NP-hard, lo que implica su alta complejidad computacional.

1.1 Formato de la UEFA Champions League

El formato de la UEFA Champions League consiste en una fase de grupos seguida de rondas eliminatorias. Inicialmente, 32 equipos se distribuyen en ocho grupos de cuatro, donde cada equipo juega contra los demás en un sistema de ida y vuelta. Los dos mejores de cada grupo avanzan a los octavos de final. A partir de octavos, la competición adopta un formato de eliminación directa a doble partido, con la final disputándose a partido único en una sede neutral

2 Modelado del Problema

El problema de optimización del sorteo de la UEFA Champions League se define como la asignación de un conjunto de equipos T en m grupos, garantizando que se cumplan diversas restricciones y optimizando un criterio dual.

Conjuntos y Variables

- $T = \{1, 2, \ldots, n\}$: Conjunto de equipos.
- $G = \{1, 2, \ldots, m\}$: Conjunto de grupos.
- $x_{i,g}$: Variable binaria que indica si el equipo i es asignado al grupo g. $x_{i,g} = 1$ si el equipo i está en el grupo g, y 0 en caso contrario.

Parámetros

- v_i: Valor competitivo del equipo i.
- R: Valor de referencia ideal para cada grupo, definido como:

$$R = \frac{1}{m} \sum_{i \in T} v_i.$$

- $\Delta \geq 0$: Umbral máximo permitido de desigualdad.
- M: Matriz de tamaño nxn donde la posición i, j indica la ganancia de emparejar los equipos i y j
- $E(g)$: Función de ganancia económica derivada de la composición del grupo g.

Restricciones

1. **Asignación única:** Cada equipo se asigna exactamente a un grupo.
$$\sum_{g \in G} x_{i,g} = 1, \quad \forall i \in T.$$
2. **Tamaño fijo de los grupos:** Cada grupo debe contener exactamente k equipos.
$$\sum_{i \in T} x_{i,g} = k, \quad \forall g \in G.$$
3. **Desigualdad limitada:** La suma total de las desigualdades de cada uno de los grupos no puede superar el umbral Δ.
4. **Compatibilidad y restricciones adicionales:** Se pueden imponer restricciones adicionales para evitar enfrentamientos entre equipos del mismo país.

Función Objetivo

El objetivo es balancear los grupos de la mejor manera posible según su valor competitivo y maximizar la ganancia económica total.

3 Demostración de NP-Hardness

Reducimos al problema PARTITION [1] que es el siguiente: Dado un conjunto de enteros positivos $\{a_1, a_2, \ldots, a_n\}$, ¿existe una partición de este conjunto en dos subconjuntos tales que la suma de los elementos en cada subconjunto sea igual?

3.1 Reducción

Dada una instancia del problema PARTITION, procedemos de la siguiente manera:

1. Para cada entero a_i de la instancia de PARTITION, definimos un equipo i y asignamos $v_i = a_i$.
2. Se fija $m = 2$ grupos, y se requiere que cada grupo contenga exactamente $\frac{n}{2}$ equipos (suponiendo que n es par, lo cual es habitual en PARTITION).
3. Se define $\Delta = 0$ para que el criterio de aceptación sea que la diferencia de suma de valores entre los dos grupos sea cero.
4. La función de ganancia para cualquier grupo se define como 0 para ignorar esta restricción.

Esta es precisamente la pregunta del problema PARTITION. Si se pudiera resolver en tiempo polinomial esta versión del problema de sorteo, entonces se podría resolver PARTITION en tiempo polinomial. Dado que PARTITION es un problema NP-completo, esto implica que la versión general del problema de optimización de sorteos es NP-hard.

La instancia construida del problema de sorteo es una instancia "especial" (o restringida) del problema general. Dado que incluso esta versión restringida es NP-hard (porque codifica PARTITION), el problema general, que además incluye restricciones adicionales y objetivos múltiples (balance competitivo y optimización económica), es también NP-hard.

Por lo tanto, si existiera un algoritmo polinomial para resolver el problema de optimización del sorteo de la UEFA Champions League, entonces se tendría un algoritmo polinomial para PARTITION, implicando que $P = NP$.

4 División en subproblemas

Se divide el problema general en varios subproblemas buscando facilidades al momento de su resolución.

1. **Subproblema de asignación de equipos en grupos**: se prioriza una asignación en grupos balanceados ignorando la maximización de la ganancia económica. Se asume que el valor competitivo de cada equipo es un número entero entre 1 y 5. Además equipos del mismo país pueden pertenecer al mismo grupo.
2. **Subproblema de cruces directos**: en esta fase se prioriza la ganancia económica seleccionando los cruces que maximicen la misma.

5 Subproblema de asignación de equipos en grupos

Sea un conjunto de $n = m \cdot k$ equipos, donde cada equipo i tiene un valor $v_i \in \{1, 2, 3, 4, 5\}$. El objetivo es particionar estos equipos en m grupos, cada uno de tamaño k, de manera que se minimice el *desbalance total*. Se define:

$$T = \sum_{i=1}^{n} v_i,$$

y se considera que el valor *ideal* de cada grupo es

$$R = \frac{T}{m}.$$

Sin embargo, en la formulación que se usa se escala el problema para trabajar con enteros; de este modo, definimos el *desbalance* de un grupo con suma S como:

$$\delta(S) = |S \cdot m - T|\,.$$

El subproblema a resolver es:

> **Encontrar una asignación de los equipos a los m grupos (cada uno de tamaño k) que minimice la suma de los desbalances de los grupos, es decir, minimizar**
>
> $$\sum_{g=1}^{m} \delta(S_g),$$
>
> *donde S_g es la suma de los valores de los equipos en el grupo g.*

5.1 Modelado del problema y estado de la DP

Para resolver el problema mediante programación dinámica (DP), se define el **estado** de la siguiente manera:

$$\text{state} = (c_1, c_2, c_3, c_4, c_5,\ j,\ s)$$

donde:

- f_v es la frecuencia del valor competitivo v.
- c_v es el número de equipos de valor v ya asignados (con $0 \le c_v \le f_v$).
- j es el número de equipos asignados en el grupo en formación (incompleto), con $0 \le j < k$.
- s es la suma de los valores de los equipos en el grupo incompleto.

El **estado inicial** es

$$(0, 0, 0, 0, 0,\ 0,\ 0),$$

y el **estado final** es aquel en el que se han asignado todos los equipos, es decir,

$$(c_1, c_2, c_3, c_4, c_5) = (f_1, f_2, f_3, f_4, f_5)$$

y además el grupo en formación está vacío ($j = 0$ y $s = 0$).

Recurrencia y Transiciones

Desde un estado dado, se intenta asignar un equipo adicional de cada valor $v \in \{1, \ldots, 5\}$, siempre que aún queden equipos disponibles (es decir, si $c_v < f_v$). Se distinguen dos casos:

Caso A: El grupo no se completa

Si al agregar un equipo de valor v se cumple que $j + 1 < k$, el nuevo estado se actualiza a

$$(c_1, \ldots, c_v + 1, \ldots, c_5,\ j + 1,\ s + v),$$

sin incurrir en costo adicional, ya que el grupo aún no se cierra.

Caso B: Se completa el grupo

Si $j + 1 = k$, al agregar un equipo de valor v se completa el grupo. Sea

$$S = s + v.$$

Se incurre en un costo local (desbalance) definido como:

$$\delta(S) = |S \cdot m - T|\,.$$

El nuevo estado es

$$(c_1, \ldots, c_v + 1, \ldots, c_5,\ 0,\ 0),$$

ya que se cierra el grupo y se inicia uno nuevo, y el costo acumulado se incrementa en $\delta(S)$.

Relación recursiva implícita:
Aunque el algoritmo se implemente de forma iterativa, la idea es la misma que en una solución recursiva con memoización: el costo mínimo para llegar a un estado depende de los costos mínimos de estados "anteriores" (con menos equipos asignados). Al procesar en niveles (según t) se garantiza que para cualquier estado nuevo el costo de sus "predecesores" ya está definido, lo que equivale a evaluar la relación recursiva:

Si no se cierra el grupo:

$$\mathrm{DP}(c_1, \ldots, c_v + 1, \ldots, c_5,\ j + 1,\ s + v) = \mathrm{DP}(c_1, \ldots, c_5,\ j,\ s).$$

Si se cierra el grupo:

$$\mathrm{DP}(c_1, \ldots, c_v + 1, \ldots, c_5,\ 0,\ 0) = \min\left(\mathrm{DP}(c_1, \ldots, c_5,\ k - 1,\ s) + |(s + v) \cdot m - T|\right), \forall s.$$

5.2 Demostración de correctitud por inducción

Para demostrar la correctitud del algoritmo, se utiliza inducción en t, donde t representa la cantidad de equipos asignados hasta el momento.

Caso base

Cuando no se ha asignado ningún equipo ($t = 0$), el único estado posible es:

$$\mathrm{DP}[0](0, 0, 0, 0, 0, 0, 0, 0) = 0.$$

Este estado representa que no hay equipos asignados y el desbalance total es 0. Dado que no hay grupos formados, la solución en este caso es trivialmente correcta.

Hipótesis de inducción

Suponemos que para algún $t \geq 0$, la función DP calcula correctamente el mínimo desbalance total hasta ese punto, cumpliendo con las ecuaciones de recurrencia:

- Si no se cierra el grupo:

$$\mathrm{DP}(c_1, \ldots, c_v + 1, \ldots, c_5, j + 1, s + v) = \mathrm{DP}(c_1, \ldots, c_5, j, s).$$

  Es decir, si se agrega un equipo pero el grupo no se completa, entonces el costo acumulado hasta el momento se mantiene igual porque aún no se ha calculado el desbalance de un grupo cerrado.

- Si se cierra el grupo:

$$\mathrm{DP}(c_1, \ldots, c_v + 1, \ldots, c_5, 0, 0) = \min\left(\mathrm{DP}(c_1, \ldots, c_5, k - 1, s) + |(s + v) \cdot m - T|\right), \forall s.$$

  Si al agregar un equipo el grupo alcanza su tamaño máximo k, se cierra y se incurre en un costo de desbalance $|(s + v) \cdot m - T|$, que se suma al costo óptimo encontrado hasta ese momento.

Paso inductivo

Queremos demostrar que la relación sigue siendo válida para $t + 1$, es decir, que la DP sigue eligiendo la mejor opción posible bajo las reglas de transición.

Caso 1: Si el grupo no se cierra : Si agregamos un equipo de valor v y el grupo no se completa, el estado transita de:

$$\mathrm{DP}(c_1, \ldots, c_v, \ldots, c_5, j, s)$$

a

$$\mathrm{DP}(c_1, \ldots, c_v + 1, \ldots, c_5, j + 1, s + v).$$

Dado que no cerramos el grupo, el desbalance no se evalúa aún, por lo que la información correcta de costos acumulados se preserva en la transición.

Como la hipótesis de inducción garantiza que para t equipos asignados, la DP ya es correcta, entonces también lo es para $t + 1$ porque solo estamos extendiendo una solución parcial sin modificar la validez del costo mínimo acumulado.

Caso 2: Si el grupo se cierra : Si agregamos un equipo de valor v y el grupo se completa ($j + 1 = k$), entonces el estado transita de:

$$\mathrm{DP}(c_1, \ldots, c_v, \ldots, c_5, k - 1, s)$$

a:

$$\mathrm{DP}(c_1, \ldots, c_v + 1, \ldots, c_5, 0, 0).$$

Dado que el grupo anterior se cierra, se incurre en un costo de desbalance $|(s + v) \cdot m - T|$, por lo que el nuevo estado almacena la mínima suma de desbalances posible hasta este punto.

Dado que por hipótesis de inducción asumimos que hasta t equipos el cálculo es correcto, y que la ecuación de recurrencia elige el mínimo costo entre todas las opciones posibles, entonces el algoritmo garantiza que el mejor camino hasta $t + 1$ también se conserva.

5.3 Análisis de complejidad

El estado de la programación dinámica se define como $(c_1, c_2, c_3, c_4, c_5, j, s)$. La cantidad de combinaciones posibles de cada variable está limitada de la siguiente manera: cada c_v puede tomar valores desde 0 hasta f_v (cantidad de equipos con ese valor). Como la suma total de los equipos es n, en el peor caso cada c_v puede tomar hasta $O(n)$ valores. Como hay 5 tipos de valores, el número de combinaciones posibles para los c_v es $O(n^5)$. Para j, como el grupo puede contener hasta k elementos ($0 \leq j < k$), el número de combinaciones posibles es $O(k)$. La suma s está acotada por el peor caso donde todos los elementos del grupo son del valor máximo (5), por lo que $s \leq 5k$, lo que implica $O(k)$ combinaciones posibles. En consecuencia, el número total de estados posibles es $O(n^5 \cdot k^2)$.

Cada estado tiene como posibles transiciones la asignación de un equipo de valor v ($1 \leq v \leq 5$), siempre que haya equipos disponibles. En cada estado, evaluamos hasta 5 transiciones (una por cada tipo de equipo disponible) y, en el peor caso, procesamos cada estado una sola vez. Por lo tanto, la cantidad total de transiciones a evaluar es $O(5 \cdot n^5 \cdot k^2) = O(n^5 \cdot k^2)$.

La complejidad total de la programación dinámica está dada por la cantidad de estados posibles multiplicada por el número de transiciones evaluadas en cada estado, resultando en $O(n^5 \cdot k^2)$.

6 Subproblema de cruces directos

Dado el conjunto de equipos que superó la fase de grupos y la matriz M en la que cada elemento i, j representa la ganancia de enfrentar al equipo i contra el equipo j. El objetivo es asignar enfrentamientos (pares de equipos) que maximicen la ganancia total. Este problema se puede modelar mediante un grafo general en el que cada vértice representa un equipo y cada arista, ponderada por la ganancia correspondiente, une dos equipos. Dado que el grafo no es bipartito, se utiliza el Algoritmo de Edmonds (Blossom Algorithm) para encontrar un emparejamiento máximo en tiempo polinomial [2].

6.1 Construcción del grafo

Representación de los equipos y de la matriz de ganancias

Sean:

- $V = \{1, 2, \ldots, n\}$ el conjunto de equipos (vértices).
- $M \in M^{n \times n}$ la matriz en que $M[i, j]$ es la ganancia al enfrentar al equipo i contra el equipo j.

Construcción del grafo

Para construir el grafo $H = (V, E)$ se sigue el siguiente proceso:

1. Cada equipo se representa como un vértice en V.
2. Para cada par de equipos (i, j), con $i < j$, se añade una arista $e = (i, j)$ con peso $w(e) = M[i, j]$, siempre que la combinación sea válida (en un grafo completo se tendrán $\frac{n(n-1)}{2}$ aristas).

Complejidad de la construcción

El proceso de construcción recorre la matriz de ganancias para evaluar cada par (i, j), por lo que la complejidad de esta fase es de $O(n^2)$.

6.2 Algoritmo de Edmonds para emparejamiento máximo ponderado

El **Algoritmo de Edmonds**, también conocido como el *algoritmo de Blossom*, es una técnica eficiente para encontrar un **emparejamiento máximo ponderado** en un grafo general. Su objetivo es encontrar un conjunto de aristas emparejadas de manera que la **suma de los pesos de las aristas** sea máxima, evitando que los vértices compartan más de una arista.

6.2.1 Conceptos Claves

Antes de describir el algoritmo, es importante definir algunos conceptos esenciales:

- **Blossom (Flor)**: Un ciclo impar en el grafo que necesita ser tratado especialmente para encontrar el mejor emparejamiento.

6.2.2 Funcionamiento del Algoritmo

El algoritmo sigue un proceso iterativo en el que se buscan caminos aumentantes y se manejan ciclos impares mediante la contracción de *blossoms*.

Paso 1: Asignación inicial de etiquetas a los vértices

Cada vértice v recibe una etiqueta inicial $y(v)$ basada en la máxima arista adyacente:

$$y(v) = \max\{w(u, v) \text{ para cada vecino } u\}$$

Esto permite definir los potenciales de cada nodo para encontrar la mejor asignación posible.

Paso 2: Construcción del Árbol de Búsqueda

El algoritmo parte de un emparejamiento inicial y busca caminos aumentantes. Si se encuentra un camino aumentante, se invierte el estado de las aristas dentro de este (las emparejadas se eliminan y las no emparejadas se incluyen en el emparejamiento).

Si durante esta búsqueda aparece un ciclo impar (*blossom*), se contrae en un solo nodo para simplificar el problema. Luego, cuando se encuentra un emparejamiento en el grafo reducido, se expande el *blossom* y se ajusta el emparejamiento dentro de él.

Paso 3: Cálculo del Costo Reducido de las Aristas

Para garantizar que el emparejamiento encontrado maximiza la ganancia, se usa un **costo reducido** definido como:

$$\tilde{w}(u, v) = w(u, v) - y(u) - y(v)$$

Este costo reducido ayuda a priorizar las aristas que pueden mejorar el emparejamiento.

Paso 4: Ajuste Dinámico de las Etiquetas

Si no se encuentra un camino aumentante, se ajustan las etiquetas de los vértices para desbloquear nuevas opciones de emparejamiento:

- Se reducen etiquetas en nodos no emparejados para permitir más conexiones.
- Se incrementan etiquetas en nodos ya emparejados para favorecer las mejores combinaciones.

El ajuste de etiquetas garantiza que las mejores aristas tengan un costo reducido cercano a cero y puedan ser seleccionadas en el emparejamiento.

Paso 5: Selección de Caminos Aumentantes con Mayor Ganancia

Una vez ajustadas las etiquetas, el algoritmo busca caminos aumentantes con las mejores aristas disponibles. Si se encuentra un camino, se invierte el estado de sus aristas dentro del emparejamiento y se repite el proceso hasta alcanzar la máxima ganancia posible.

6.3 Complejidad Temporal

El algoritmo de Edmonds tiene una complejidad de:

$$O(n^3)$$

Esto se debe a que:

- La búsqueda de caminos aumentantes toma $O(n^2)$.
- La contracción y expansión de *blossoms* puede ocurrir $O(n)$ veces.
- El ajuste de etiquetas para maximizar la ganancia contribuye al orden cúbico total.

6.4 Complejidad de múltiples aplicaciones del algoritmo

En el caso de la segunda fase del torneo como avanzan los 2 mejores equipos de cada grupo, se tienen en total $2 * m$ equipos para esta fase. Siendo su comlejidad temporal $O(m^3)$.

Para esta fase se necesita ejecutar el algoritmo varias veces hasta llegar al partido final. Cada vez se reduce a la mitad la cantidad de equipos que quedan en la competencia, quedando la complejidad de la segunda fase de la siguiente manera:

$$\mathrm{T(m)} = T(\frac{m}{2}) + O(m^3)$$

El cual por Teorema maestro se sabe que tiene una complejidad total de $O(m^3)$.

References

[1] Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, & J. D. Bohlinger (Eds.), *Complexity of computer computations* (pp. 85–103). Plenum. https://doi.org/10.1007/978-1-4684-2001-2_9.

[2] Edmonds, J. (1965). *Paths, Trees, and Flowers*. Canadian Journal of Mathematics, 17, 449–467. https://doi.org/10.4153/CJM-1965-045-4