Informe proyecto de curso

Planificación de Calendarios Deportivos

Análisis de Algoritmos II

Escuela de Ingeniería de Sistemas y Computación

Carlos Caceres Campos 2126639
Jose Gabriel Jiménez Vidales 1941557
Andres Felipe Velasco Coronado 1941375
Santiago Gutierrez Hincapie 1843859

Universidad del Valle
13 de Junio del 2023

Abstract

Este informe se centra en la difícil tarea de generar calendarios apropiados para torneos deportivos, abordando condiciones complejas como la alternancia de partidos en casa y fuera, la programación de partidos de ida y vuelta y la minimización de desplazamientos. Para resolver este desafío, se aplican técnicas de optimización junto con el uso del lenguaje MiniZinc para construir modelos y encontrar soluciones eficientes y viables. El objetivo principal es encontrar un calendario deportivo que satisfaga las condiciones previamente planteadas y que además, minimice los desplazamientos que tengan que hacer los equipos para enfrentarse a sus rivales.

Introducción

El presente informe tiene como objetivo abordar el desafiante problema de la creación de calendarios adecuados para torneos deportivos. La generación de calendarios para torneos deportivos implica satisfacer una serie de condiciones y restricciones complejas, como el número par de equipos, la alternancia de partidos locales y visitantes, la programación de partidos de ida y vuelta, minimizar los desplazamientos de los equipos visitantes, entre otras.

Para resolver este problema, se recurre a técnicas de optimización, como la programación lineal, la programación mixta y la programación entera. Estas técnicas permiten formular y resolver problemas de optimización con múltiples variables y restricciones. La construcción de modelos de optimización adecuados, que incluyan variables y restricciones precisas, es fundamental para encontrar soluciones óptimas o cercanas a la óptima.

En este informe, se explorará la construcción de modelos de optimización específicos para la creación de calendarios en torneos deportivos. Se utilizará MiniZinc, un lenguaje eficiente y flexible para representar modelos, definir variables y restricciones, y encontrar soluciones viables. El objetivo es proporcionar una solución eficiente y viable a este desafiante problema, garantizando la satisfacción de los equipos.

Problema

La creación de calendarios adecuados para torneos deportivos es una tarea compleja que implica cumplir con una serie de condiciones y restricciones. Entre ellas se encuentran la alternancia de partidos locales y visitantes, la programación de partidos de ida y vuelta y la minimización de desplazamientos. Además, cada equipo debe jugar exactamente una vez en cada fecha del calendario.

Resolver este desafío requiere considerar el número de equipos participantes, que debe ser par con el fin de formar parejas de enfrentamientos en cada jornada. En los torneos de una ronda, el calendario debe asegurar que cada equipo se enfrente una sola vez a cada otro equipo. En los torneos de dos rondas, se programan fechas en las cuales cada equipo se enfrenta exactamente dos veces a cada otro equipo, alternando la localía.

Otras restricciones importantes incluyen la alternancia de partidos de local y visitante para cada equipo en cada jornada, evitar programar partidos de vuelta antes de los de ida, minimizar los desplazamientos de los equipos visitantes.

¿Entendimos el problema?

D:

0	745	665	929
745	0	80	337
665	80	0	380
929	337	380	0

Cal1 :

3	4	-1	-2
2	-1	4	-3
-3	-4	1	2
4	-3	2	-1
-2	1	-4	3
-4	3	-2	1

Costos :
Se calcula el costo total de las giras del Cal1:

Gira equipo 1 : d13 + d31 + d12 + d24 + d41 = 665 + 665 + 745 + 377 + 929 = 3381
Gira equipo 2 : d21 + d14 + d43 + d32 = 745 + 929 + 380 + 80 = 2134
Gira equipo 3 : d31 + d13 + d34 + d42 + d23 = 665 + 665 + 380 + 377 + 80 = 2167
Gira equipo 4 : d42 + d23 + d34 + d41 + d14 = 377 + 80 + 380 + 929 + 929 = 2695

Costo total : 3381 + 2134 + 2167 + 2695 = 10377

Esta solución es una solución factible, pues cumple con todas las condiciones, sin embargo no es la solución óptima al problemas pues como se puede evidenciar más adelante (Cal3) existe una solución que sigue cumpliendo con todas las restricciones y tiene un costo menor de distancias.Es prueba que permutando los valores se puede llegar a diferentes soluciones que satisfagan el problema pero que no necesariamente son las mejores, esto cobra mas sentido en el desarrollo del informe dado que en ocasiones el encontrar la solución óptima toma mucho tiempo y se toma en cuenta una solución factible.

Cal 2:

3	4	-1	-2
-2	1	-4	3
2	-1	4	-3
-3	-4	1	2
4	-3	2	-1
-4	3	-2	1

- En este caso, podemos identificar que Cal2 no cumple con la restricción de no repetir un mismo partido en dos fechas consecutivas. Podemos ver en el calendario que los partidos entre los equipos 1 y 2 se repiten en las fechas 2 y 3, esto sucede en varias ocasiones aparte de la ejemplificada.

- Ahora se tiene la consideración de cambiar el valor Max de 3 a 2, en este caso ninguna de las soluciones Cal, Cal1, Cal2 serían factibles. Esto se debe a que ninguna de las soluciones cumple con la restricción de que el tamaño de toda gira y toda permanencia sea menor o igual al valor declarado en Max.

  Entrada: n = 4, min = 1, max = 2.

D:

0	745	665	929
745	0	80	337
665	80	0	380
929	337	380	0

Una solución a este problema cumpliendo todas las restricciones sería:

Cal3:

-3	-4	1	2
-2	1	4	-3
4	3	-2	-1
2	-1	-4	3
-4	-3	2	1
3	-1	-1	-2

El costo de la gira en este caso es de: 10287

El modelo:

Para este proyecto se planteó un modelo matemático para las restricciones que debe de tener este problema, buscando restringir la región factible para poder encontrar resultados.

Parámetros de entrada

- n: Número de equipos.
- $D_{i,j}$: Matriz que representa la distancia que hay entre el equipo i y j. $(\forall i\ ,\ \forall j \in \{1..\ n\})$
- Min : Representa el mínimo número de giras y permanencias que un equipo puede tener
- Max: Representa el máximo número de giras y permanencias que un equipo puede tener.

Donde

$n \in\ 2N$

$Min,\ Max\ \in\ N$

Variables

- $Cal_{i,j}$: Matriz que representa contra cual equipo se enfrenta el equipo j en la fecha i, ya sea de local o visitante y no existe el 0.
  $(-\ n \leq Cal_{i,j} \leq n,\ Cal_{i,j} \neq 0,\ \ \forall i\ \in \{1..\ 2(n\ -\ 1)\}, \forall j\ \in \{1..\ n\})$

  $Cal_{i,j} \in\ N$s
- $ContadorGiras_{i,j}$: Matriz que representa el acumulado de giras consecutivas del equipo j en la fecha i. donde $(0\ \leq ContadorGiras_{i,j}\ \leq\ Max)$
- $ContadorPermanencias_{i,j}$: Matriz que representa el acumulado de permanencias consecutivas del equipo j en la fecha i. donde $(0\ \leq ContadorPermanencias_{i,j}\ \leq\ Max)$

Restricciones

Para toda fecha i:
$\forall i \in \{1, 2, ..., 2(n-1)\}$:

- La suma de cada fecha debe de ser 0
  $(\sum_{j=1}^{n} Cal_{i,j}) = 0$

- Cada equipo puede jugar una sola vez por fecha
  $\forall k, j \in \{1, 2, ..., n\},\ k < j : Cal_{i,k} \neq Cal_{i,j}$

- Si un equipo juega de local en una fecha, el equipo contrincante jugará de visitante
  $\forall k, j \in \{1, 2, ..., n\}:\ Cal_{i,j} = k \Leftrightarrow Cal_{i,k} = -j$

- No puede repetirse un partido en dos fechas consecutivas
  $\forall j \in \{1..n\}\ |Cal_{i,j} \neq -Cal_{i+1,j}$

Para cada equipo j:
$\forall j \in \{1, 2, ..., n\}$:

- No puedo jugar 2 veces contra el mismo equipo en la misma condición (Visitante o local)
  $\forall k, i \in (1..2(n-1)),\ k < i : Cal_{i,j} \neq Cal_{k,j}$

Para todo el calendario:
$\forall i \in \{1, 2, ..., 2(n-1)\}, \forall j \in \{1, 2, ..., n\}$:

- La misma cantidad de números positivos y negativos en la matriz, de tal forma que asegure que en todos los partidos, uno de los equipos juega de local y el otro de visitante
  $\frac{n}{2} = |\{Cal_{i,j} > 0 : 1 \leq j \leq n\}| = |\{Cal_{i,j} < 0 : 1 \leq j \leq n\}|$, para todo $1 \leq i \leq 2(n-1)$

- Para que los equipos tengan partidos de ida y vuelta. Uno de local y otro de visitante, en diferentes fechas.
  $\forall j \in [1..n],\ \forall k \neq j,\ \exists i1,\ i2 \in [1., 2(n-1)] :\ Cal_{i1,j} = k \wedge Cal_{i2,j} = -k.$

- Los equipos no pueden jugar el partido de vuelta hasta que todos hayan jugado el partido de ida
  $\forall j \in [1..n],\ \forall k, i \in [1..(n-1)],\ k < i :\ |Cal_{i,j}| != |Cal_{k,j}|$

- Los equipos deberán tener un número de giras mayor o igual al Min o menor igual al Max.

  MAX

  La restricción de que los equipos no puedan superar el número de giras y de permanencias se da en el momento en que se define que el número máximo que pueden tomar las matrices $ContadorPermanencias_{i,j}$ y $ContadorGiras_{i,j}$ es de Max, estas matrices suman un 1 a la posición anterior en caso de que esté de gira o permanencia, si no lo está, se llena con un 0.

  MIN: El final de cambio de gira o permanencia debe de ser mayor a Min

  $\forall i \in \{2, 3, \ldots, 2(n-1)\}, \forall j \in \{1, 2, \ldots, n\}$:
  Si $(i = 2(n-1) | Si\ (Cal_{i-1,j} < 0 : Contador_{i-1,j} \geq Min))$
  Sino:
  Si$(Cal_{i,j} > 0 \wedge Cal_{i-1,j} < 0 : Contador_{i-1,j} \geq Min)$

Función Objetivo:

Minimizar el costo de la suma de las giras, buscando que los equipos no se desplacen demasiado.

$\text{Min}(\sum_{j=1}^{n} (\sum_{i=1}^{2(n-1)} ($
si(i==1 |
si($Cal_{i,j}$ <0 | D[|$Cal_{i,j}$ |, j]
sino(0))
sino(
si($Cal_{i,j}$ >0 |
si ($Cal_{i-1,j}$ < 0 | D[|$Cal_{i-1,j}$|,j]
sino(0))
sino(
si(i = 2*(n-1)|
si($Cal_{i-1,j} < 0$| D[|$Cal_{i-1,j}$|, |$Cal_{i,j}$|]+D[|$Cal_{i,j}$|,j]
sino(D[|$Cal_{i,j}$|,j]+D[|$Cal_{i,j}$|,j])
)
sino(si($Cal_{i-1,j} < 0$| D[|$Cal_{i-1,j}$|, |$Cal_{i,j}$|]

sino(D[|$Cal_{i,j}$|,j]
))
))
))
))
)))

Programación entera:

En este modelo todas las restricciones hacen parte de la programación entera pues están acotadas por los valores que puede tomar $Cal_{k,j}$, como este solo puede tomar valores enteros sus restricciones también.

Programación Mixta:

Por otra parte la función objetivo hace parte de la programación lineal, pues depende de minimizar una matriz de distancias D, la cual puede tomar valores Reales positivos, además teniendo en cuenta que nuestro objetivo es minimizar las distancias recorridas, puede ser mas optimo el encontrar una solución en los reales, sin embargo al utilizar condicionales, estaríamos utilizando a su vez programación binaria pues internamente los condicionales se representan con 0 y 1.

Análisis de árboles generados

A Partir de la implementación del modelo a la instancia donde se manejaron los siguientes datos de entrada:

número de equipos = 4,
mínimo = 1,
máximo = 3.

Tabla de distancias :

0	745	665	929
745	0	80	337
665	80	0	380
929	337	380	0

llegamos al siguiente árbol de soluciones :

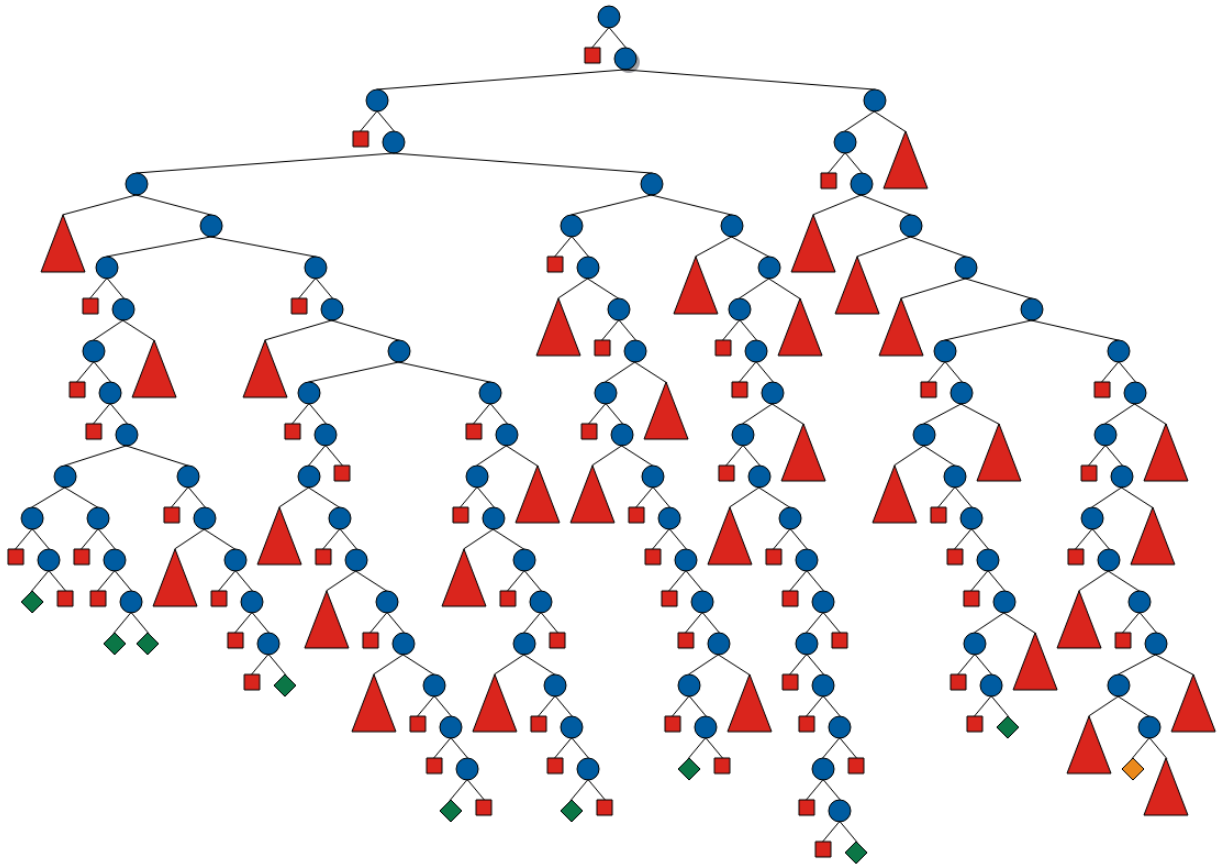

Figura 1. *Árbol generado a partir del algoritmo Branch and Bound aplicado a los datos de entrada :* número de equipos = 4, mínimo = 1 ,máximo = 3 y la tabla de distancias del punto de análisis de árboles generados.

En el caso del primer nodo, los valores de la matriz "Cal" son generados aleatoriamente en el rango de [-4..-1,1..4] para cada posición. La variable "objective" tiene un rango de [0..25116].

Dado que cada posición de la matriz puede contener valores en el rango [-4..-1,1..4], es posible que algunas combinaciones de valores violen las restricciones. Estas violaciones son un motivo para podar una rama del árbol de búsqueda, ya que podrían conducir a soluciones no válidas.

Como lo viene siendo el caso que se presenta en el segundo nivel del árbol :

Nodo Izquierdo:

{-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4}, {-3..-1,1..4},

{-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4}, {-3..-1,1..4},
{-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4}, {-3..-1,1..4},
{-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4}, {-3..-1,1..4},
{-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4}, {-3..-1,1..3},
{-3..-1,1..4}, {-3..-1,1..4}, {-3..-1,1..4}, -4

Nodo Derecho :

{-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4},
{-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4},
{-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4},
{-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4},
{-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4},
{-4..-1,1..4}, {-4..-1,1..4}, {-4..-1,1..4}, {-3..-1,1..4}

En el nodo izquierdo se puede evidenciar que en la última fecha el equipo cuatro tendría que jugar de visitante contra sí mismo, lo que haría que el calendario no tuviera sentido. Por lo tanto, se decide podar esa rama.

Se evidencia cómo se realiza el mecanismo de branch and bound porque el árbol expande nodos a medida que exista un estado de la solución que se pueda mejorar aún más. Como se muestra en el ejemplo anterior, se podó el nodo izquierdo. Sin embargo, el nodo derecho, aunque no viola ninguna restricción y es una solución válida, aún se pueden mejorar los valores dentro de las posiciones de la matriz para obtener un menor costo. De esta manera, se repite el mismo proceso con diferentes ramas del árbol. También se evidencia que hay casos en donde el árbol simplifica subárboles porque esa rama no fue capaz de llegar a una solución que satisfaga todas las restricciones del problema o porque la solución hallada en esa rama no fue capaz de superar a la que ya fue encontrada en otra rama. Además, el árbol cuenta con rombos verdes que nos indican que se llegó a la mejor solución de esa rama, ya que si se intentara seguir mejorando el costo en ese caso, lo estaríamos empeorando, por lo que se detiene la búsqueda de la solución en esa rama. Por último, el rombo amarillo nos permite reconocer cuál fue la mejor solución hallada en el árbol, la cual sería aquella que tenga el menor costo y satisfaga todas las restricciones del problema.

Pruebas:

Se utilizaron diferentes casos de prueba para evaluar el comportamiento del algoritmo y asegurarse de que produce resultados correctos y factibles. Las pruebas se llevaron a cabo utilizando una variedad de entradas con diferentes valores de n, Min y Max, así como distintas matrices de distancias D. Se verificó que la implementación generará una matriz de calendario Cal que cumpliera con todas las restricciones establecidas en el problema, incluyendo la alternancia de localidad en los partidos y el tamaño máximo de giras y permanencias.

1. **Caso 1:**

Entrada:
n=10;
Min=1;
Max=6;
D=
[|0, 594, 296, 426, 443, 366, 233, 114, 192, 310
|594, 0, 580, 574, 603, 228, 409, 622, 402, 284
|296, 580, 0, 130, 147, 352, 529, 182, 178, 296
|426, 574, 130, 0, 29, 346, 659, 312, 238, 414
|443, 603, 147, 29, 0, 375, 676, 329, 255, 431
|366, 228, 352, 346, 375, 0, 467, 394, 174, 222
|233, 409, 529, 659, 676, 467, 0, 347, 421, 245
|114, 622, 182, 312, 329, 394, 347, 0, 220, 338
|192, 402, 178, 238, 255, 174, 421, 220, 0, 176
|310, 284, 296, 414, 431, 222, 245, 338, 176, 0|];

Salida:
SATISFIED
Costo de viajes: 42020
Calendario:
[7, -6, 9, 8, 10, 2, -1, -4, -3, -5]
[-6, 7, -10, -9, 8, 1, -2, -5, 4, 3]
[5, 3, -2, 7, -1, -10, -4, 9, -8, 6]
[9, -10, 4, -3, 7, -8, -5, 6, -1, 2]
[3, 8, -1, -10, 6, -5, -9, -2, 7, 4]
[10, 4, -5, -2, 3, -9, 8, -7, 6, -1]
[-8, -9, -6, -5, 4, 3, -10, 1, 2, 7]
[2, -1, -7, 6, -9, -4, 3, 10, 5, -8]
[-4, -5, -8, 1, 2, 7, -6, 3, 10, -9]
[6, -8, -4, 3, 9, -1, 10, 2, -5, -7]
[-9, -3, 2, 5, -4, -7, 6, -10, 1, 8]
[-5, -4, 8, 2, 1, 10, 9, -3, -7, -6]
[-2, 1, 7, 10, -6, 5, -3, -9, 8, -4]
[-10, 5, -9, -7, -2, 8, 4, -6, 3, 1]
[-7, 9, 6, -8, -10, -3, 1, 4, -2, 5]
[-3, -7, 1, -6, -8, 4, 2, 5, -10, 9]
[4, 10, 5, -1, -3, 9, -8, 7, -6, -2]
[8, 6, 10, 9, -7, -2, 5, -1, -4, -3]

Análisis:
Los datos que se usaron para esta prueba son de un tamaño considerable para la función, esto quiere decir que le tomará un tiempo considerable encontrar la solución óptima, en este caso al no encontrar la solución óptima en los primeros 7 minutos (tiempo estándar para este proyecto) arroja como salida la última solución que encontró cumplido ese tiempo, esto no quiere decir

que no encuentre la solución óptima, sino que el tiempo para encontrarla depende del tamaño de n, es decir, la cantidad de equipos según el caso. Tiempo de ejecución +7 min

2. **Caso 2:**

Entrada:
n=4;
Min=1;
Max=2;
D=[|0, 315, 318, 293
|315, 0, 633, 608
|318, 633, 0, 121
|293, 608, 121, 0|];

Salida:
OPTIMAL_SOLUTION
Costo de viajes: 6810
Calendario:
[-2, 1, -4, 3]
[4, -3, 2, -1]
[3, 4, -1, -2]
[-4, 3, -2, 1]
[-3, -4, 1, 2]
[2, -1, 4, -3]

Análisis:
Para este caso se utilizaron datos pequeños para el problema objetivo, se declararon menor cantidad de equipos para que el problema mostrará su capacidad de encontrar la solución óptima. Tiempo de ejecución ~2 seg

3. **Caso 3:**

Entrada:
n=6;
Min=1;
Max=2;
D=[|0, 260, 485, 264, 445, 396
|260, 0, 389, 64, 705, 656
|485, 389, 0, 325, 436, 387
|264, 64, 325, 0, 709, 660
|445, 705, 436, 709, 0, 89
|396, 656, 387, 660, 89, 0|];

Salida:
SATISFIED
Costo de viajes: 17332
Calendario:

[-2, 1, 4, -3, -6, 5]
[-4, -5, 6, 1, 2, -3]
[3, 6, -1, -5, 4, -2]
[5, 3, -2, -6, -1, 4]
[-6, -4, 5, 2, -3, 1]
[-5, -3, 2, 6, 1, -4]
[6, 4, -5, -2, 3, -1]
[4, 5, -6, -1, -2, 3]
[-3, -6, 1, 5, -4, 2]
[2, -1, -4, 3, 6, -5]

Análisis:
Para este caso se utilizaron datos promedios, ni muy altos, ni muy bajos, sigue cumpliento todas las restricciones y da un resultado que satisface el problema, incluso mejor que el propuesto por el profesor. Tiempo de ejecución +7 min

4. **Caso 4:**

Entrada:
n=8;
Min=2;
Max=3;
D=[|0, 455, 449, 606, 228, 219, 439, 618
|455, 0, 794, 151, 397, 464, 564, 417
|449, 794, 0, 675, 397, 330, 230, 377
|606, 151, 675, 0, 378, 387, 445, 298
|228, 397, 397, 378, 0, 67, 211, 390
|219, 464, 330, 387, 67, 0, 220, 399
|439, 564, 230, 445, 211, 220, 0, 179
|618, 417, 377, 298, 390, 399, 179, 0|];

Salida:
SATISFIED
Costo de viajes: 27744
Calendario:
[-7, 4, -8, -2, 6, -5, 1, 3]
[-8, 3, -2, -5, 4, -7, 6, 1]
[6, -5, -4, 3, 2, -1, -8, 7]
[5, -6, 7, 8, -1, 2, -3, -4]
[-3, 8, 1, -6, -7, 4, 5, -2]
[-2, 1, -6, -7, -8, 3, 4, 5]
[-4, -7, -5, 1, 3, -8, 2, 6]
[2, -1, 6, 7, 8, -3, -4, -5]
[4, 7, 5, -1, -3, 8, -2, -6]
[-6, 5, 4, -3, -2, 1, 8, -7]
[-5, 6, -7, -8, 1, -2, 3, 4]

[3, -8, -1, 6, 7, -4, -5, 2]
[8, -3, 2, 5, -4, 7, -6, -1]
[7, -4, 8, 2, -6, 5, -1, -3]

Análisis:
Para este caso se probó con un número min > 1, esto quiere decir que a la hora de arrojar un resultado, la función evidencia una restricción que no es muy utilizada por el resto de pruebas que sería tomar en cuenta que el tiempo de permanencia mínimo sea mayor a 1. Tiempo de ejecución +7 min Primera solución : 7 seg

5. **Caso 5:**

Entrada:
n=4;
Min=1;
Max=1;
D=[|0, 194, 216, 400
|194, 0, 340, 206
|216, 340, 0, 546
|400, 206, 546, 0|];

Salida:
UNSATISFIABLE
No se encontró solución

Análisis:
En este caso podemos observar que la función no encuentra una solución que cumpla las restricciones, en pocas palabras es insatisfacible. En este caso en concreto no se satisface debido a que no hay forma de que un equipo pueda permanecer ya sea de local o visitante un máx de 1 vez sin que se juegue primero los partidos de ida antes que cualquier partido de vuelta que es una de las restricciones. Tiempo de ejecución ~2 seg.

Análisis:

En general, las pruebas realizadas demostraron una buena implementación del algoritmo y produjeron soluciones óptimas o factibles que cumplen con todas las restricciones del problema. Tiene la limitación del tiempo para acotar la búsqueda de resultados para entradas muy grandes, en caso de que se declare una gran cantidad de equipos el programa no podría encontrar ninguna solución, teniendo en cuenta estas excepciones los resultados con las pruebas obtenidos fueron consistentes y coincidieron con las expectativas esperadas.

Para hacer un análisis más detallado de las pruebas es necesario poder entender la complejidad de este tipo de programación y como el solver encuentra las soluciones. Como ya se mencionó

anteriormente los solvers llevan a cabo un proceso de ramificación para encontrar soluciones y que tan óptimas son, este proceso puede crecer exponencialmente y por ende cualquier variación en los datos de entrada puede hacer crecer el tiempo estimado para encontrar una solución óptima dependiendo de la complejidad del programa.

En este caso el programa tiene una complejidad bastante alta. por ende los tiempos registrados en las pruebas son los siguientes:

Caso	n (Número de equipos)	Tiempo(min)	Tiempo (seg) Primera Solución	Solución
1	10	+7	69	factible
2	4	0.2	1	optima
3	6	+7	2	factible
4	8	+7	8	factible
5	4	0.2	1	insatisfacible

Figura 2. *Tabla de datos sobre las diferentes pruebas que se realizaron, en donde se indica el número de equipos, tiempo estimado* para encontrar solución, el tiempo estimado en encontrar la primera solución, y si encuentra una solución óptima, factible o insatisfacible

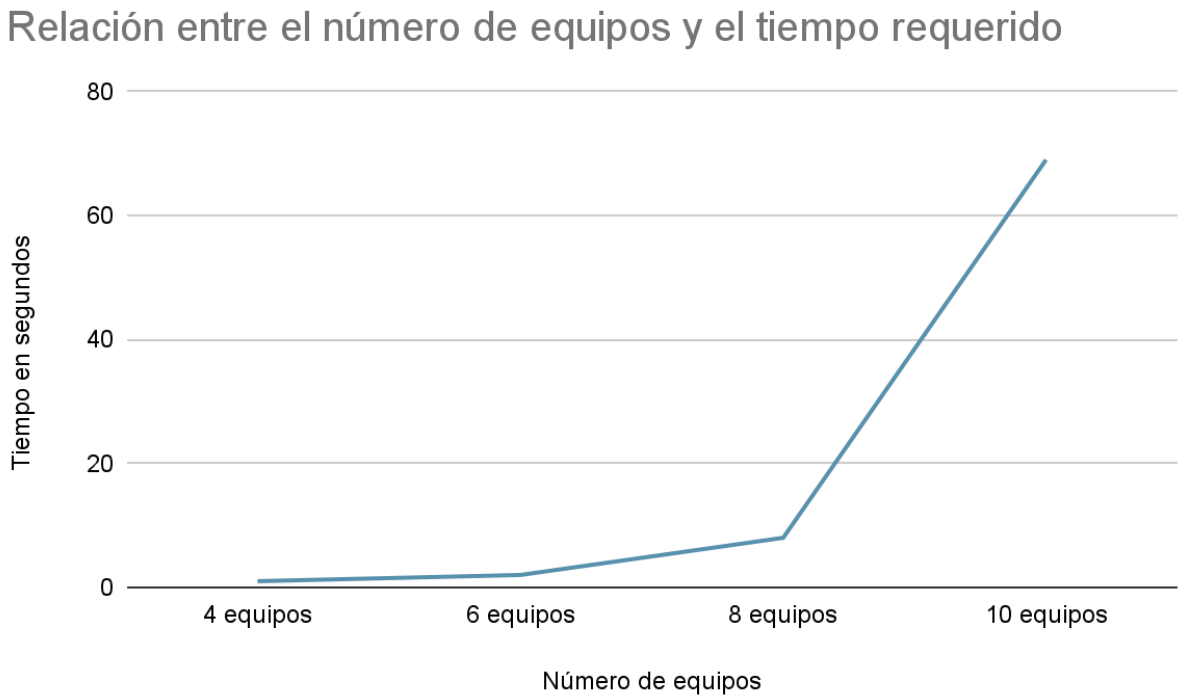

Figura 3. *Gráfico que relaciona el número de equipos con el tiempo en que se demora encontrar la primera solución*

Como se puede apreciar en la ***Figura 2***, los tiempos que toma el programa para correr instancias con 6 o más equipos toma un tiempo mayor a nuestro límite , mientras que el 100% de los casos donde se utilizaron un número de equipos menor a 6 este logro encontrar la solución óptima en segundos, o en su defecto decir que el problema es insatisfacible. Por otro lado, en la ***Figura 3*** se evidencia un comportamiento en donde si aumentamos el número de equipos, aumenta de manera exponencial el tiempo en el que el solver se demora en encontrar una solución, ya sea la solución óptima, una solución satisfactoria o insatisfacible.

Detalles importantes de la implementación

Para la implementación del problema, primero se llevó a cabo el modelo escrito y definido previamente en el lenguaje de modelado de restricciones llamado MiniZinc. Una vez escrito el modelo de manera formal, fue relativamente sencillo la implementación en MiniZinc debido a que este nos ofrece lenguaje de alto nivel para modelar restricciones en pocas líneas de código. Posteriormente, se creó una interfaz sencilla en Python usando Streamlit donde le permite cargar los archivos con los parámetros de entrada que desee el usuario. Internamente, para

encontrar la solución una vez leídos estos archivos, se utilizó la librería de MiniZinc Python, la cual nos permite utilizar fácilmente los modelos construidos en MiniZinc directamente en Python. Cabe resaltar que se le dio un tiempo máximo al modelo a la hora de calcular una solución, ya que hay casos que por la magnitud del problema, tiene un costo en tiempo muy elevado. Un problema que se presentó con la librería MiniZinc Python fue la incompatibilidad de algunos solvers, en macOS funcionó el solver gecode, pero en Windows no, por lo que al final se optó por usar el solver chuffed, que funciona en ambos sistemas operativos.

Explicación de interfaz

En el siguiente enlace se encuentra la explicación en video: Link

Conclusión

La búsqueda de un calendario que cumpla con todas las restricciones establecidas y, al mismo tiempo, sea capaz de minimizar la distancia recorrida entre los equipos al jugar un partido, es una tarea compleja. Sin embargo, al desarrollar el proyecto mediante la modelación del problema y luego el uso de los solvers, se logró trazar un camino claro para el desarrollo de la solución.

No obstante, se pudo observar que en todos los casos se encontró una solución al problema, siempre y cuando este tuviera una solución factible considerando las restricciones. Aunque no siempre se logró encontrar la solución óptima, se podrían explorar otras estrategias o modificar algunas restricciones para mejorar el comportamiento del algoritmo. Además, a partir del análisis de la tabla de pruebas, se evidenció que a medida que aumenta el tamaño de las entradas, el programa tarda más en encontrar una solución factible, lo que indica que el algoritmo tiene una alta complejidad para entradas grandes, en este caso, exponencial.