Un nuevo modelo para la planificación de los partidos de la fase regular de la NBA

Begoña Álvarez Tena

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Director del trabajo: Pedro M. Mateo Collazos
28 de junio de 2021

Abstract

The objective of this Report is to present the methodology for fixture list for the NBA regular season. This topic has been chosen due to my fondness of basketball, in particular the NBA; and the interest that *Operation Research* subject aroused in me during the degree, in special, the possibility of applying Mathematics to real problems.

Without a doubt the NBA is the most important basketball league in the world and as a result it creates the most interest and spectacle. Nowadays, the league is made up of 30 teams, which compete every season, to win the champion's ring. In the regular season teams play 82 games. Each team play against the others 2, 3 or 4 times. This depends on different factors. The competition system does have some evident disadvantages:

- It's considered unfair that teams do not play against some teams the same number of times.
- The increase in the number of games has also increased the threat of injury, and a lot of the regular season games are played with a lower level of intensity because of the amount of games.
- The distances the teams travel are high too and this generates a lot of pollution.

In the present project, a scheduling design with two games (home and away games) between the teams is presented (this is common for the rest of competitions) but with a particular differentiating element. This element exists because of the huge distances between two cities. This is called the *trip* and it means the one team might play several away games consecutively before going back home. The sequence of games was called *trip* and it's important in the model that is going to be created[1]. The mathematics model that exists for this type of double round tournament sport with trips is known as: *Travelling Tournament Problem* (TTP) and it was applied in others sports [5, 6].

The TTP used for NBA scheduling has some modifications. It's difficult on a computational level. The mathematical field that plays an important role is Operation Research and in particular, Lineal Programming (PL) as well as Integer Lineal Programming (PLE). Lineal Programming uses a mathematical model to minimise or maximise the objective function whilst considering a set of constraints which are modelled by lineal expressions. A problem with PLE is a PPL with part of the decision variables are integers. The problems that are created in this project are binary PLE problems, where the decision variable only has two values: $\{0, 1\}$. To resolve these problems branch and bround techniques are used and the informatic program: CPLEX.

The approach to this unique TTP model, which resolution provides competition scheduling, is only possible on a computational way by reducing the number of teams. The major problem solved only have 12 teams [3, 4]. To approach this situation with more than 12 teams (it's our case) we need to conduct the methodology in two phases like the suggestion in [4]. The first phase decides the order of in which the games will be played and in the second phase the days of the games are assigned based on existing availability. The first phase creates the sequences and trips with the objective to minimise the total number of kilometres travelled, while in the second phase the objective is minimising the necessary games days and the rest between the games so players have enough time to relax.

One of the newness on this project is the way of making trips. In this reference [3] the trips are suggested by each team and federation, while in this project the trips are going to be made by the

[1] If we remove the existence of trips, the double round system competition is not complicated to mold and solve.

model. For that cluster analysis and genetics algorithms have been used. Specifically the algorithm K-Means has been studied in order to allows us to build patterned groupings according to distance, and one modification of this clustering method is K-Means Constrained that allows us to control the size of the clusters, in other words the size of the trips. Secondly, the basic notions of genetic algorithms have been studied in order to adapt the algorithms suggested in [7] from the 8-queens to the problem of obtaining the set of trips the teams will make with the aim to minimise the kilometers travelled.

The resolution to this problem is divided into two phases. The first phase creates the order that the teams will play their fixtures. In this phase we talk about *rounds* instead of days to resolve the problem, but later they don't match with the exact days, and in the second phase it decides the exact day when each match will take place following the order from the first phase.

As mentioned previously the objective of the first phase is to minimise the total kilometers that all teams travelled, and the objective in the second phase is reduce the number os byes and the number of days on which matches are planned. It's important to notice that the second objective function is weighted.

The methodology will be shown with only 12 teams, where it is not necessary to split the original problems. However, later it will be used for all 30 teams in the League. The second phase is unsolvable in a reasonable amount of time, for this reason it does a *Rolling Horizant* Plan to solve the two problems during the process of generating the fixtures. This strategy consists of creating a sequence with different periods of time, solving and joining the results to obtain a solution. Through these techniques the scheduling produced will have some unwanted characteristics, however it is a feasible solution to the problem of making sure teams do not travel unnecessarily long distances.

Índice general

Abstract III

1. Introducción y objetivos 1
1.1. La NBA en la actualidad . . . 1
1.2. Objetivos . . . 2
1.3. Tournament Travelling Problem. TTP . . . 2
1.4. Estructura del documento . . . 3

2. Programación lineal entera 5
2.1. Conceptos previos . . . 5
2.2. PLE. Método general de resolución de problemas de PLE . . . 5

3. Técnicas cluster y algoritmos genéticos 9
3.1. Técnicas cluster . . . 9
3.1.1. Algoritmo K-Means . . . 9
3.1.2. Algoritmo K-Means Constrained . . . 10
3.2. Algoritmos genéticos . . . 11

4. Desarrollo, modelización e implementación del problema 15
4.1. 1ª Fase: Determinación de trips y secuencia de partidos . . . 15
4.1.1. Notación . . . 15
4.1.2. Restricciones . . . 16
4.1.3. Función objetivo . . . 16
4.1.4. Planteamiento . . . 17
4.2. 2ª Fase: Creación del calendario de partidos . . . 17
4.2.1. Notación . . . 17
4.2.2. Restricciones . . . 18
4.2.3. Función objetivo . . . 19
4.2.4. Planteamiento . . . 19
4.3. Modelización . . . 20
4.3.1. Planteamiento subproblemas . . . 21
4.4. Implementación y resultados . . . 22
4.4.1. Estudio 12 equipos . . . 22
4.4.2. Estudio NBA . . . 23

5. Conclusiones 27

Bibliografía 29

Anexos 31

A. NBA 33

B. Ejemplos **39**
B.1. Resolución de un problema de PLE aplicando técnicas de ramificación y acotación . . 39
B.2. Aplicación del método *K-Means* en un ejemplo con 6 datos 43

C. Códigos **45**
C.1. Kmvar 45
C.2. Generación de trips mediante clusters 46
C.3. Generación de trips mediante algoritmos genéticos 50
C.3.1. Evaluación de trips 50
C.3.2. Recombinación 50
C.3.3. Mutación 50
C.3.4. Selección de supervivientes 51
C.3.5. Programa principal - BuildTrips 51
C.4. Programa principal - CPLEX 53
C.4.1. Problema 1 53
C.4.2. Subproblema1-1 56
C.4.3. Subproblema1-2 59
C.4.4. Subproblema1-4 63
C.4.5. Problema 2 66

D. Calendario optimizado para 12 equipos **73**

Capítulo 1

Introducción y objetivos

La optimización es una de las ramas de las matemáticas que más interés ha despertado en mí a lo largo del grado. Por ello ampliar mis conocimientos en este campo tiene un especial atractivo.

El presente trabajo surge debido a mi gran interés en dicho campo y a mi gran dedicación en el mundo del baloncesto. Así pues, la finalidad es unir mis conocimientos adquiridos con una de mis mayores aficiones, la *National Basketball Association* (NBA).

1.1. La NBA en la actualidad

La NBA está considerada la mejor liga de baloncesto del mundo. Esta la disputan 29 equipos de América del Norte y 1 equipo de Canadá, dando lugar así al total de 30 equipos que conforman la liga. Se puede consultar el nombre de los equipos así como sus correspondientes ubicaciones en el Anexo A.

El funcionamiento de la NBA en la actualidad es el siguiente:

- Está formada por dos grupos de 15 equipos cada uno, que reciben el nombre de Conferencia Este y Conferencia Oeste. Dichos grupos están formados por la proximidad existente entre los diferentes equipos. Además, dentro de cada conferencia existen 3 divisiones, compuesta cada una por 5 equipos.
- Se disputan un total de 82 partidos (jugando el mismo número de partidos de local[1] como de visitante) distribuidos de la siguiente manera:
  - Contra los equipos de la conferencia opuesta se juegan 2 partidos.
  - Contra los equipos de la misma conferencia se juegan 3 ó 4 partidos (esta decisión no sigue ningún tipo de regla, se decide si son 3 ó 4 partidos para que el calendario de partidos quede cuadrado).
  - Contra los equipos de la misma división se juegan 4 partidos.

Estos 82 partidos dan como resultado la fase regular de la NBA, estableciendo así una clasificación del 1 al 15 en cada conferencia, para posteriormente jugar la fase final y conseguir el anillo[2]. Se puede ver el sistema de competición de la fase final en el Anexo A. Esta información y otra relativa a la NBA se puede encontrar en [1].

Cabe notar que el formato comentado es el que se ha llevado a cabo hasta la temporada 2018-2019. La temporada 2019-2020 tuvo que finalizarse con un formato distinto como consecuencia del COVID19, mientras que en la temporada 2020-2021 se ha reducido el número de partidos que juega cada equipo a 72 debido a que la temporada 2019-2020 finalizó meses más tarde de lo previsto, provocando el comienzo tardío de la temporada 2020-2021. El formato que se ha propuesto, pero aún no confirmado para la próxima temporada se puede ver en [2].

[1] Se entiende que un equipo juega como local cuando juega en su propio campo y como visitante cuando no; es decir, si se va a disputar el partido Suns - Bulls en Phoenix, se dice que Suns juegan como locales y Bulls juegan como visitantes.

[2] Se llama anillo al premio que recibe el equipo ganador de la NBA.

1.2. Objetivos

Como se ha podido observar, actualmente se juegan una gran cantidad de partidos en la fase regular. Esto supone recorrer un elevado número de kilómetros, y en la mayoría de los casos, existen considerables diferencias entre los propios equipos. Por ejemplo, en la temporada 2017-2018 el equipo que más kilómetros recorrió fue TrailBlazers con unos 89000 kilómetros, mientras que el equipo que menos kilómetros tuvo que desplazarse fue Pacers con aproximadamente 60550 kilómetros [12].

Como esta gran diferencia de kilómetros se debe a la localización de los equipos, y realizar un equilibrado resulta complejo, nos planteamos minimizar el número de kilómetros que recorren todos los equipos. Este será el principal objetivo del trabajo. Se proporcionará como resultado un calendario lo más optimizado de partidos, indicando el día en el que se jugará cada partido y el número de kilómetros totales que recorrerá cada equipo.

En el Anexo A, se muestra la tabla de distancias obtenidas a partir de la latitud y longitud de las ciudades a la que pertenece cada equipo.

Actualmente, en la NBA cuando un equipo se desplaza a jugar un partido, no juega por lo general contra un único equipo, juega contra varios de ellos. Este concepto recibe el nombre de *trip*. Un *trip* es una agrupación de equipos contra los que un equipo juega de manera consecutiva sin volver a casa. Por ejemplo, podría existir el trip Lakers, Clippers y Suns para Celtics, esto significaría que si Celtics eligiera este trip debería jugar contra los 3 equipos mencionados de manera consecutiva y se diría que va a jugar un trip.

Esta última idea se va a seguir utilizando para resolver el problema planteado. Se deberán construir distintos tipos de trips utilizando técnicas cluster y algoritmos genéticos que se explicarán en el Capítulo 3. Cabe destacar que se construirán una gran cantidad de posibles trips, para que posteriormente sea el programa el que, al resolver el problema, decida cuáles son los más adecuados entre los propuestos. Todo ello con el objetivo de minimizar los kilómetros totales.

Para la resolución del problema anterior, se han planteado dos modelos de programación lineal entera. Posteriormente, han sido resueltos con CPLEX [9], un programa informático de optimización cuyas técnicas de resolución que emplea se explicarán en el Capítulo 2.

Por lo tanto, en el trabajo se plantea un sistema de competición comúnmente conocido como ida-vuelta, al que se le denomina: *Tournament Travelling Problem* (TTP) [5, 6]. Esto conlleva que el número de partidos en la fase regular se reduzca de 82 a 58 por equipo. De esta forma, el sistema de competición será más justo, ya que todos los equipos jugarán contra todos en las mismas condiciones, de manera que se clasificarán para disputar los *playoff* los 16 mejores equipos, independientemente de la conferencia a la que pertenezcan.

Por último, podemos mencionar que con la propuesta comentada, se contribuye a reducir el número de posibles lesiones que sufren los jugadores durante la temporada a consecuencia de los partidos disputados y del poco tiempo de descanso que tienen. Además, se contribuye a reducir las emisiones de CO2 y en consecuencia la huella de carbono producida por dicha competición, ya que los equipos realizan todos sus desplazamientos en avión.

1.3. Tournament Travelling Problem. TTP

Sea n el número de equipos que disputan un torneo y sea conocida la distancia existente entre cada uno de ellos, el objetivo del TTP es minimizar la distancia recorrida entre todos los equipos durante un torneo deportivo. El TTP está planteado para que se jueguen $2(n-1)$ jornadas, de manera que el número de partidos consecutivos que dispute un equipo, tanto de local como de visitante, esté entre $[L,U]$ siendo L y U enteros no negativos. Además, se tiene en cuenta que si el equipo i juega contra el equipo j en la jornada k, estos dos equipos no pueden volver a jugar entre ellos hasta haber disputado al menos un partido contra todos los equipos. Por último, el TTP exige que durante un trip ningún equipo regrese a casa.

Este tipo de problema ha sido aplicado en diferentes tipos de competiciones deportivas en los últimos 30 años, siendo la primera aplicación real la que se dio en la liga de "volleybal" argentina. En [3, 4] se pone de manifiesto que dicho problema se puede resolver con relativa facilidad cuando se trata de 4 equipos, un poco más de complejidad tiene al aumentar a 6 equipos y con 8 equipos se llegan a obtener soluciones (en algunos casos) pero con un tiempo de ejecución extremadamente largo. Además, para un número mayor de 12 equipos no se ha logrado resolver este tipo de problema. Es por ello que, en este trabajo donde el número máximo de equipos es 30, se ha planteado subdividir el problema en distintos subproblemas, generando de esta manera soluciones sub-óptimas.

Cabe notar, que el problema planteado en este trabajo difiere del TTP original. Se mantiene que: se minimizan los kilómetros recorridos por los equipos; el formato ida-vuelta; y se exige que durante un trip los equipos no vuelvan a casa. Sin embargo, no se impone que los equipos deban jugar una ronda completa antes de comenzar la segunda ronda. Debido a la dureza y complejidad del problema y de las limitaciones propias de un *Trabajo fin de grado*, se ha conseguido obtener soluciones completas para un modelo simplificado con 12 equipos y soluciones parciales para el caso completo (30 equipos). El desarrollo del planteamiento del problema se podrá ver en el Capítulo 4.

1.4. Estructura del documento

En el Capítulo 2 se va a proceder, por un lado a realizar una breve introducción sobre la programación lineal entera, y por otro lado a explicar el funcionamiento de uno de sus principales métodos de resolución: el algoritmo de ramificación y acotación.

En el Capítulo 3, se expondrán dos métodos utilizados para la construcción de trips: uno basado en técnicas cluster y el otro diseñando un algoritmo genético. Se explicará el funcionamiento de cada uno de ellos en términos generales, así como la forma en la que se han utilizado en el caso que nos concierne. Es posible notar, que sin la creación de los trips, a través de estas técnicas u otras existentes, no se podría avanzar en la resolución del problema.

Posteriormente, en el Capítulo 4 se desarrollarán dos modelos de programación lineal entera binaria, explicando las restricciones empleadas y la necesidad de hacer uso de las mismas. Uno de ellos determina el orden en el que deben jugarse los partidos y el otro, determina los días exactos de juego. Se proporcionará el planteamiento general para resolver un problema con estas características y una explicación sobre su modelización e implementación. Como ya se ha comentado, para resolver el problema completo será necesario dividirlo en subproblemas. Esto implicará que en general se proporcionen subóptimos. Además, como se ha mencionado previamente, el problema será planteado para los 30 equipos que conforman la NBA, obteniendo solución solo para la primera fase (ordenación de partidos), y para uno de dimensión más pequeña, 12 equipos obteniendo una solución completa para la que se indicará el número de kilómetros que tendrá que recorrer cada equipo, así como un calendario optimizado.

Por último, en el Capítulo 5 se analizarán los resultados obtenidos y se propondrán futuras líneas de trabajo para mejorar el funcionamiento de la modelización propuesta.

Capítulo 2

Programación lineal entera

La programación lineal entera (PLE) fue introducida en la asignatura *Investigación Operativa* de tercer curso. Esta se centró principalmente en el planteamiento de dichos problemas y no en su resolución. Por ello, en este capítulo se presentará brevemente la PLE así como una de sus técnicas más habituales de resolución: técnicas de ramificación y acotación. Además, se mostrará por medio de un ejemplo de dimensión pequeña como se aplica dicha técnica. Cabe destacar que la programación lineal entera tiene gran importancia, ya que sin ella hubiera sido imposible plantear el problema real que se presenta en el trabajo que nos concierne.

2.1. Conceptos previos

Antes de definir el concepto de programación lineal entera, es conveniente saber qué es un problema de programación lineal (PPL).

La programación lineal utiliza un modelo matemático para maximizar o minimizar una función objetivo teniendo en cuenta un conjunto de restricciones. En estos modelos, tanto la función objetivo como las restricciones se representan mediante expresiones lineales.

La formulación de un PPL puede plantearse sin pérdida de generalidad de la siguiente forma:

$$\begin{cases} \max & Z(\mathbf{x}) = \mathbf{cx} \\ \text{s.a} & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{cases}, \tag{2.1}$$

donde $\mathbf{x}' = (x_1, \ldots, x_n)$ representa las variables de decisión (que se asumen no negativas), $\mathbf{b}' = (b_1, \ldots, b_m)$ es el vector de recursos, $\mathbf{c} = (c_1, \ldots, c_n)$ es el vector de costes, $Z(x_1, \ldots, x_n)$ representa la función objetivo, $\mathbf{A}$ es una matriz $m \times n$ y la expresión $\mathbf{Ax} \leq \mathbf{b}$ representa el conjunto de restricciones[1].

Cabe mencionar que un PPL puede no tener solución, tener una única solución o bien infinitas soluciones. Para resolver este tipo de problemas, un algoritmo ampliamente utilizado es el algoritmo del *Simplex* [8]. Este lo tienen implementado todos los softwares de optimización y actualmente permite resolver PPL de grandes dimensiones.

2.2. PLE. Método general de resolución de problemas de PLE

2.2.1 Definición. *Un problema de PLE, es un PPL en el que parte de sus variables de decisión han de tomar valores enteros. Podemos distinguir 3 tipos de problemas de PLE:*

- ***Problemas de PLE puros**: donde todas las variables de decisión $x_j \in \mathbb{Z}^+ \ \forall j$, siendo $\mathbb{Z}^+ = \{0, 1, \cdots\}$.*

[1] Estas restricciones pueden ser también de tipo $\geq$, $>$, $<$ ó $=$, y pueden no ser todas ellas del mismo tipo.

- ***Problemas de PLE binarios (PEB)****: donde las variables de decisión pueden tomar únicamente dos valores. Normalmente estos valores son:* 0 *y* 1*, y están asociados a tomar o no una cierta decisión. Las variables de decisión* x_j *reciben el nombre de variables de decisión binarias.*

- ***Problemas de PLE mixtos****: donde solo algunas de las variables de decisión deben tomar valores enteros. Denotaremos:*

$$\begin{cases} x_j \geq 0 \text{ y enteras} & j \in J \\ x_j \geq 0 & j \notin J \end{cases}.$$

Cabe notar, que los problemas que se plantearán en el Capítulo 4 serán del tipo PEB.

Una vez vistos los tipos de problemas de PLE, es momento de ver cuáles son los métodos de resolución que se utilizan. Existen 2 técnicas principales para resolver dichos problemas: los métodos de planos de corte[2] y las técnicas de ramificación y acotación [8]. A continuación se va a proceder a explicar con detalle estas últimas, debido a que son las que más peso tienen en los paquetes de optimización, como es en el caso de CPLEX.

La idea que subyace detrás de las técnicas de ramificación y acotación es la siguiente: en lugar de intentar resolver el problema original directamente, se trata de resolver un problema más sencillo asociado al problema inicial. Si este no proporciona la solución óptima, se dividirá en dos subproblemas a resolver, de manera que unidos formen el problema original. En el caso de que alguno de estos dos subproblemas proporcione una solución factible y se pueda garantizar que es óptima, el proceso se detendrá. En caso contrario se volverá a dividir el subproblema en dos nuevos subproblemas. El proceso continua hasta que se encuentre una solución óptima o se llegue a un problema no factible.

2.2.2 Definición. *Se llama problema relajado (PR) de un problema de PLE al PPL obtenido al eliminar todas las condiciones de integridad que existen sobre las variables de decisión.*

Denotaremos por las siglas VFO, al valor que toma la función objetivo. El VFO del problema relajado es siempre igual o mejor que el VFO del problema de PLE del que provenga.

Cuando se resuelve un problema de PLE de máximo utilizando técnicas de ramificación y acotación se debe proceder de la siguiente manera:

- En primer lugar, se establece una cota inferior[3] del VFO de la mejor solución del problema entero hasta el momento, $Z^* = -\infty$. Esta se irá actualizando conforme se vayan encontrando soluciones factibles que la mejoren. Se resuelve el problema relajado $(PR1)$ del problema original (P), cuya solución es $\mathbf{X^1}$ y VFO $\mathbf{Z^1}$. Si la solución obtenida es factible, es decir $x_j^1 \in \mathbb{Z}\ \forall j \in J$, entonces dicha solución es la solución óptima y el problema queda resuelto. Si el problema relajado $(PR1)$ del problema original (P) es infactible el proceso finaliza. En caso contrario, comienza el proceso iterativo del algoritmo de ramificación y acotación, que consta de los dos elementos que le dan nombre:

  Ramificación - Acotación.

- **Ramificación**: Se selecciona un problema relajado ya resuelto y que no haya sido cerrado inicialmente (PR_k) con solución $\mathbf{X^k}$. Se elige una de las $j \in J$ con $x_j^k \notin \mathbb{Z}$ pero que debería pertenecer a $\mathbb{Z}$, de manera que si el valor de $x_j^k = c$, se plantean dos subproblemas relajados (PR_{H+1}) y (PR_{H+2}), y en cada uno de ellos hay que añadir una nueva restricción a las ya existentes:

$$PR_{H+1} = \begin{cases} PR_H \\ + \\ x_j^k \leq \lfloor c \rfloor \end{cases}. \qquad PR_{H+2} = \begin{cases} PR_H \\ + \\ x_j^k \geq \lfloor c \rfloor + 1 \end{cases}.$$

[2] Los planos de corte se suelen utilizar como una herramienta adicional dentro de la ramificación y acotación.

[3] Si el problema fuera de mínimo se establecería una cota superior del VFO, $Z^* = +\infty$.

A continuación se resuelven (PR_{H+1}) y (PR_{H+2}).

Cabe notar que a la hora de elegir una de las $j \in J$ con $x_j^k \notin \mathbb{Z}$ se pueden seguir, entre otros, los siguientes criterios:

- Arbitrariamente, se elige cualquiera de las variables de decisión x_j con $j \notin J$ que deberían pertenecer a J.
- Se escoge la variable de decisión que tome un mayor valor no entero hasta el momento.
- Cada variable de decisión que debe tomar valor entero, tiene asignada una prioridad, luego se elige la variable de decisión en función de dicha prioridad.
- Se elige la variable de decisión en la que se alcanza el máximo de esta expresión:

$$\max_{x_j}\{min\,(b_j - \lfloor b_j \rfloor, \lfloor b_j \rfloor + 1 - b_j)\}.$$

- **Acotación**: Se pueden dar distintas situaciones al resolver los subproblemas (PR_{H+1}) y (PR_{H+2}).
  - Si alguno de los anteriores subproblemas relajados a resolver resulta ser no factible, se cierra esa ramificación como consecuencia de la no factibilidad. Esto se debe a que al añadir más restricciones a un problema que no es factible, este va a seguir siendo no factible.
  - Si alguna de las soluciones que se obtienen son enteras, es decir $x_j^{k+1} \in \mathbb{Z}\ \forall j \in J$ ó $x_j^{k+2} \in \mathbb{Z}$ $\forall j \in J$ y el VFO es mejor que el que se tenía anteriormente, se procede a actualizar la cota Z^*, siendo $Z^* = \mathbf{Z^{k+1}}$ ó $\mathbf{Z^{k+2}}$ y el problema correspondiente se cierra (ya no se ramifica más) porque cualquier subproblema tendrá una solución igual o peor que la ya encontrada.
  - Si existe solución para alguno de los subproblemas, pero esta es no factible para el problema de PLE y además la solución obtenida es igual o peor a Z^*, el problema se cierra por acotación[4].

Una vez que se ha realizado el proceso mencionado con los subproblemas construidos, se escoge alguno de los que no han sido cerrados (han podido ser cerrados por infactibilidad, por haber encontrado una solución factible para el problema de PLE o por acotación) y se procede a ramificar nuevamente. El proceso finaliza cuando no quedan subproblemas no cerrados.

Igual que existen criterios para la elección de la variable de decisión a ramificar, también existen criterios para la elección del subproblema a ramificar, algunos de ellos son:

- Se elige cualquier subproblema no cerrado de manera arbitraria.
- Se escoge el subproblema con mayor VFO, de esta manera se espera que tengamos una mayor posibilidad de cerrar problemas por acotación, o bien tengamos la posibilidad de encontrar soluciones con mejor VFO.
- Se toma el subproblema no cerrado más nuevo.
- Se elige el subproblema no cerrado más antiguo.

Se puede observar que todos los subproblemas relajados que se resuelven al aplicar las técnicas de ramificación y acotación se resuelven, generalmente, aplicando el algoritmo del *Simplex* ya que se tratan de problemas de PLE.

En el Anexo B se presenta un problema de dimensión reducida en el cual se puede ver la implementación del algoritmo de ramificación y acotación.

[4] Asumimos que estamos buscando una de las soluciones óptimas. Si quisiéramos buscar todas las soluciones óptimas, el problema solo se cerraría cuando la solución fuera peor a Z^*.

Capítulo 3

Técnicas cluster y algoritmos genéticos

El objetivo de este capítulo es presentar dos técnicas diferentes para la construcción de trips. Para ello, en primer lugar se va a proceder a introducir las técnicas de análisis cluster. Se presentará la técnica clásica de *K-Means* y una de sus modificaciones (*K-Means Constrained*) que será la que posteriormente permita crear los trips necesarios. En segundo lugar, se introducirán brevemente los algoritmos genéticos que serán utilizados para construir trips de una manera, a priori, más realista.

3.1. Técnicas cluster

Un cluster es una agrupación de datos que presentan ciertas características similares. De manera más formal, se puede decir que el análisis cluster es una técnica estadística multivariante que busca agrupar un conjunto de datos con características similares, tratando así de crear grupos lo más homogéneos posibles.

Existen dos grandes tipos de análisis de clusters:

- **No jerárquicos**: se debe definir previamente el número de clusters que se quiere para posteriormente, asignar cada dato a un cluster diferente.
- **Jerárquicos**: su objetivo es agrupar clusters para formar uno nuevo o bien para separar alguno ya existente, y dar origen a otros dos, de manera que si sucesivamente se va efectuando este proceso se minimizan distancias o se maximiza alguna medida de similitud.

Comenzamos explicando unos de los métodos más habituales debido a su sencillez y efectividad: el método de *K-Means*. Este pertenece a las técnicas no jerárquicas. Posteriormente se explicará una variante de *K-Means* que recibe el nombre de *K-Means Constrained*. Este último se utilizará para construir los trips, ya que nos permitirá controlar el tamaño de los cluster, es decir el tamaño de nuestros trips.

3.1.1. Algoritmo K-Means

Como se ha mencionado anteriormente, el algoritmo *K-Means* pertenece al grupo de métodos no jerárquicos, y por ello en primer lugar se debe fijar el número de clusters, N, que se quieren construir.

Sea $\mathbf{x}^i$ el conjunto de items de los que disponemos, cada uno de ellos con m características medidas, $\mathbf{x}^i = \left(x_1^i, \cdots, x_m^i\right)$ con $i = 1, \cdots, k$. Se quieren crear N clusters, en función de sus características. Para ello, se define una medida de similaridad o de distancia entre ellos, de manera que se pueda calcular $d\left(\mathbf{x}^i, \mathbf{x}^j\right)$. El método construye N elementos $\mathbf{C}^l = \left(C_1^l, \cdots, C_m^l\right)$ con $l = 1, \cdots, N$ denominados *centroides*, de manera que cada punto $\mathbf{x}^i$ se asocia al centroide $\mathbf{C}^l$ más cercano de acuerdo a la distancia $d\left(\mathbf{x}^i, \mathbf{C}^l\right)$. Cabe notar que $N \leq k$, ya que en caso contrario se querrían construir más clusters que el número de items de los que se disponen.

Formalmente el algoritmo *K-Means* queda de la siguiente manera:

- **Paso 0**: Construir un conjunto de centroides iniciales, $\mathbf{C}^l$ con $l = 1, \cdots, N$. Estos pueden elegirse de manera aleatoria o bien hacerlos coincidir con algunos de los datos que se tengan.

- **Paso 1**: Asociar cada dato $\mathbf{x}^i$ al centroide $\mathbf{C}^l$ más cercano.

  $\forall l = 1, \cdots N$, siendo r_i el índice del centroide asociado a $\mathbf{x}^i$ con $i = 1, \cdots, k$.

  $$r_i = \underset{l=1,\cdots,N}{argmin} \left\{ d\left(\mathbf{x}^i, \mathbf{C}^l\right) \right\}.$$

  Si la asignación del cluster no ha cambiado, o bien se ha alcanzado el número máximo de iteraciones: STOP.

- **Paso 2**: Actualizar los centroides.

  $\forall l = 1, \cdots, N$

  $$\mathbf{C}^l = \frac{1}{\left|\mathbf{CK}^l\right|} \sum_{\mathbf{x}^i \in \mathbf{CK}^l} \mathbf{x}^i,$$

  donde $\mathbf{CK}^l = \{\mathbf{x}^i | r_i = l, \forall i = 1, \cdots, k\}$. Tras actualizar los centroides volver al paso 1.

Cabe notar que el algoritmo finaliza cuando los centroides no cambian entre dos iteraciones o se alcanza un número máximo de iteraciones.

Se puede observar un ejemplo de la aplicación del algoritmo en el Anexo B.

A lo largo de la explicación del método *K-Means* se ha podido observar que el tamaño de cada cluster no se controla, y en consecuencia no tienen porque tener el mismo tamaño. Como nuestros trips están sujetos a tener unos tamaños preestablecidos no podemos utilizar el algoritmo *K-Means*. Por ello consideraremos una modificación suya que se presenta en la siguiente sección.

3.1.2. Algoritmo K-Means Constrained

Para solucionar el problema que se acaba de comentar, el algoritmo que se presenta en [10] denominado *K-Means Constrained*, permite establecer el tamaño de los clusters. Es por ello que utilizaremos este algoritmo para construir los trips en el modelado de nuestro problema. En particular, proponemos crear trips de tamaño 2 y 3. Se puede ver la función implementada en R (*Kmvar*, obtenida de [11]), en el Anexo C.1.

El funcionamiento de este algoritmo es muy similar al ya explicado para el método *K-Means*. En este caso, el tamaño de los clusters se fija previamente, lo que conlleva que cuando un cluster se completa ya no se puede asignar ningún otro dato a él. Debido a esta situación, es importante establecer el orden en el que se van a asignar los datos. En el caso de que el cluster resulte completo, se procede a asignar el dato al segundo mejor cluster, y así sucesivamente.

Formalmente, el algoritmo *K-Means Constrained* queda de la siguiente manera:

- **Paso 0**: Construir un conjunto de centroides iniciales, $\mathbf{C}^l$ con $l = 1, \cdots, N$. Estos pueden elegirse de manera aleatoria o bien hacerlos coincidir con algunos de los datos que se tengan.

- **Paso 1**: Elegir el orden en el que se va a asignar cada dato $\mathbf{x}^i$ a un cluster. Los autores del código proponen las siguientes 4 opciones:

  - De manera aleatoria.
  - Distancia al centroide más cercano menos distancia al centroide más lejano.
  - Distancia al centroide más cercano.
  - Distancia al centroide más lejano.

- **Paso 2**: Asociar cada dato $\mathbf{x}^i$ al centroide $\mathbf{C}^l$ más cercano todavía disponible, es decir, a un cluster que no este completo. Si esta completo, se asocia al siguiente más cercano.

- **Paso 3**: Actualizar los centroides, de manera análoga al método *K-Means*.

En nuestro caso, tras realizar distintas pruebas y observar los resultados obtenidos, el método utilizado para la ordenación de los equipos es el tercero de los propuestos, debido a que la asignación de los equipos a los clusters parece más sensata.

Una vez hallados los clusters hay que construir los trips, para ello se toma uno a uno cada equipo y cada uno de los cluster (asumiendo que el tamaño del cluster es ≥ 3).

Si el equipo pertenece al cluster se quita, y sobre los elementos restantes del cluster se determina el orden en el que deben recorrerse teniendo en cuenta que hay que viajar desde la ciudad del equipo a la ciudad del primer equipo del trip, jugar todos los partidos del trip y volver a la ciudad del equipo, realizando el mínimo número de kilómetros.

Se puede ver el código programado, que utiliza el método anterior, en R para la construcción de trips en el Anexo C.2.

3.2. Algoritmos genéticos

Un algoritmo genético es un algoritmo de optimización inspirado en la evolución biológica. Estos algoritmos se basan en la evolución de las poblaciones, donde los individuos que tienen mayor probabilidad de sobrevivir son aquellos que están más adaptados al medio. Además, los individuos que presentan una mejor adaptación, tienen unas cualidades que los hacen superiores, las cuáles presumiblemente podrán transmitir a la descendencia.

En el proceso reproductivo la información genética de los padres se recombina para generar la información genética de los hijos, información que además puede sufrir ciertos cambios espontáneos que modifican los genes de la carga genética recibida. Estos pequeños cambios se denominan *mutaciones*.

Para medir la adaptación al medio se define una función denominada *fitness*. A mayor fitness mejor adaptación del individuo al medio y mayor probabilidad de ser seleccionado para reproducirse y sobrevivir. En consecuencia, el individuo tiene una mayor probabilidad de sobrevivir en el medio durante sucesivas generaciones.

Este proceso conlleva, en general, que se produzcan cambios favorables en algunos individuos, haciendo que estos presenten una mejor adaptación al medio, mientras que en otros individuos los pequeños cambios serán desfavorables.

Se puede ver un esquema del funcionamiento de un algoritmo genético[1] en la figura 3.1:

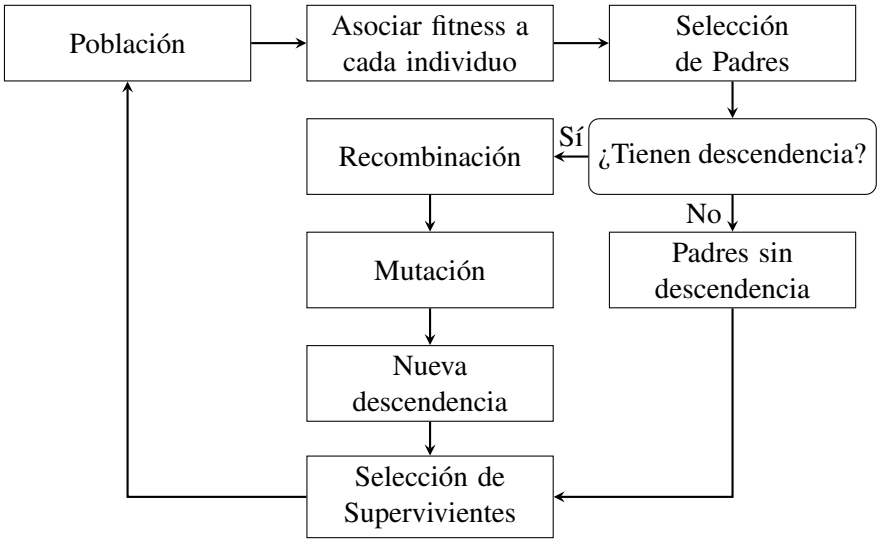

Figura 3.1: Esquema del funcionamiento de un algoritmo genético

[1] El esquema que se presenta es uno de los posibles planteamientos existentes.

Sea **P** una población de m individuos:

$$\begin{cases} \mathbf{p^1} = \left(p_1^1, \cdots, p_k^1\right) \\ \vdots \\ \mathbf{p^m} = \left(p_1^m, \cdots, p_k^m\right) \end{cases},$$

donde los individuos de la población son posibles soluciones del problema y la población es un conjunto de soluciones. Cabe notar que existen diversas formas de representación de los individuos, entre las más comunes se encuentran: strings $\{0,1\}$, vectores $\mathbf{x} \in \mathbb{R}^n$ y permutaciones.

Los pasos que se deben seguir cuando se aplica un algoritmo genético son los siguientes:

- **Paso 0**: Construir una población inicial y asociar a cada individuo $\mathbf{p}^i$ de la población un fitness f_i con $i = 1, \cdots, m$.

- **Paso 1**: Elegir de entre todos los individuos $\mathbf{p}^i$ con $i = 1, \cdots, m$ aquellos que puedan producir descendencia en la población (denominados *padres*). Para dicha elección existen distintos métodos, entre los más conocidos se encuentran:

  1. **Selección por Ruleta**: Se utiliza para la selección de los individuos. La probabilidad p_i de que el individuo $\mathbf{p}^i$ sea elegido es proporcional a la diferencia entre su fitness, f_i, y la del resto de individuos.
  $$p_i = \frac{f_i}{\sum_{i=1}^{m} f_i}.$$
  Cabe notar que un mismo individuo puede ser seleccionado más de una vez.

  2. **Selección por Torneo**: Se eligen subgrupos de individuos de la población, y los miembros de cada subgrupo compiten entre ellos. De cada subgrupo únicamente un individuo es elegido como ganador.
  Si se tiene el subgrupo formado por lo individuos $x_{j_1}, \cdots, x_{j_s}$ seleccionados aleatoriamente. El individuo elegido como ganador es:
  $$x_{j_k} = \underset{j=j_1,\cdots,j_s}{argmax}\{f_j\}.$$
  Cabe notar que, nuevamente, un mismo individuo puede ser seleccionado más de una vez.

  3. **Selección por ranking**: Cada individuo de la población tiene asignado un rango numérico basado en su fitness, y la selección de los individuos se basa en el rango numérico asignado en vez de en el fitness. Una ventaja que presenta este método, es que evita que individuos que a priori tienen un fitness asociado de mucho valor, no obtengan mucha más importancia con respecto a los individuos que presentan un menor fitness.

  El proceso de elección de padres se debe repetir hasta que se consiga llenar el número de padres que se quiere obtener.

- **Paso 2**: Realizar las recombinaciones y cruces correspondientes. Se juntan los padres por parejas aleatoriamente, cada pareja tiene asociada una probabilidad de reproducción (generalmente se suele tomar valores grandes, por ejemplo $p = 0.8$). En el caso de que no se reproduzcan los padres se conservan como están para la siguiente iteración, en el caso de que se reproduzcan se generan 2 *hijos* (en general).

  Una vez generados los hijos, estos pueden sufrir pequeñas mutaciones. Al igual que una mutación en los seres vivos produce una modificación de la información genética, una mutación en un algoritmo genético también causa pequeñas modificaciones en la codificación de los individuos.

Una de las posibilidades más sencillas para la reproducción de los padres, se denomina *recombinación de punto de corte*. En esta ocasión se intercambian segmentos del código genético de los padres, de manera que los individuos descendientes, hijos, son combinaciones de sus padres. Sean los padres:

$$\text{Padre1: } 00010|01000, \text{ Padre2: } 10010|11000$$

Una posible recombinación para la generación de sus dos hijos es:

$$\text{Hijo1: } 00010|11000, \text{ Hijo2: } 10010|01000$$

donde se han cortado a los progenitores en el gen 5 y se han intercambiado las colas de los cromosomas.

A su vez, los hijos sufren pequeñas modificaciones. Por ejemplo:

$$\text{Hijo1: } 00\mathbf{0}1011000 \rightarrow 00\mathbf{1}1011000$$

$$\text{Hijo2: } 1001001000 \rightarrow 1001000000$$

donde con una probabilidad muy pequeña se ha alterado el valor de alguno de los genes.

- **Paso 3**: Elegir supervivientes. Se juntan los padres originales, los padres que no se han recombinado y los hijos para proceder al proceso de selección de supervivientes. La elección de supervivientes se realiza favoreciendo aquellos que tenga un mejor valor de fitness, aunque esta también se puede realizar de manera completamente elitista; es decir, quedándonos con los de mejor fitness. Tras esta elección se debe volver al paso 1.

Los pasos anteriores se repiten hasta cumplir un cierto criterio de parada, número máximo de iteraciones, tiempo máximo de ejecución, un cierto número de iteraciones sin mejora, etc.

En el caso particular de nuestro problema, el algoritmo genético utilizado se ejecuta para cada equipo y construye un conjunto de trips que responden a un patrón dado e intenta minimizar la suma de kilómetros recorridos por el equipo. Por ejemplo, si tenemos 12 ciudades, un posible patrón es buscar 3 trips de 3 equipos y 1 trip de 2 equipos. Otro podría ser, buscar 1 trip de 4 equipos, 1 trip de 3 equipos y 2 trips de 2 equipos.

En nuestro problema, los individuos se codifican mediante permutaciones de n equipos y un patrón previamente definido. El fitness asociado a un individuo es la suma total de kilómetros recorridos al realizar los trips indicados por las permutaciones encontradas y el patrón. El número de padres que se generan por iteración es 2 y se generan 2 hijos. Para la elección de los padres, se aplica el método de *selección por torneo*, donde se eligen 5 individuos y se obtienen 2 ganadores. En este caso, la recombinación de los padres y generación de 2 hijos es obligatoria. Veamos como se realiza a través de un ejemplo. Supongamos que el algoritmo se está aplicando para la ciudad 12 (de un conjunto de 12 ciudades) y sean los siguientes dos padres de la población:

$$\text{Padre1: } 1|4|5||6|8|9||7|2|10||3|11, \text{ Padre2: } 1|5|8||4|3|10||6|9|11||2|7$$

donde las líneas azules indican el patrón. Para el Padre1 se han creado los siguientes trips de tamaño 3: $\{1,4,5\}$, $\{6,8,9\}$, $\{7,2,10\}$ y el trip de tamaño 2: $\{3,11\}$. Análogamente, se pueden ver los trips correspondientes para el Padre2. Estos trips están ordenados, es decir para el trip 1 de la ciudad 12 se debe ir a la ciudad 1, posteriormente a la 4, después a la 5 y finalmente regresar a la ciudad 12.

Para la recombinación se elige un número al azar entre 2 y el número de ciudades, en nuestro caso 11. Suponemos que el número elegido es 3.

$$\text{Padre1: } 1|4|5||6|8|9|7|2|10|3|11, \text{ Padre2: } 1|5|8||4|3|10|6|9|11|2|7$$

Los hijos que se obtienen son:

$$\text{Hijo 1: } 1|4|5||3|10|6|9|11|2|7|8, \text{ Hijo 2: } 1|5|8||6|9|7|2|10|3|11|4$$

donde el hijo i conserva el primer segmento del padre i, con $i = 1,2$. La cola de cada hijo se completa siguiendo el orden de las ciudades de la cola del padre del que no proviene el primer segmento, ignorando las ciudades que ya estén escritas. De esta manera nos aseguramos conservar la estructura de permutación.

Por último, para la mutación de los hijos se intercambian dos ciudades elegidas aleatoriamente y de esta forma también se mantiene la estructura de permutación.

$$\text{Hijo 1: } 1|4|5|3|10|6|9|11|2|7|8 \rightarrow 1|4|5|3|10|7|9|11|2|6|8$$

$$\text{Hijo 2: } 1|5|8|6|9|7|2|10|3|11|4 \rightarrow 1|5|8|6|4|7|2|10|3|11|9$$

Los dos hijos generados reemplazarán a los dos peores padres de la población.

Este proceso se repite 30000 iteraciones, donde el individuo elegido para formar los trips es aquel que tiene un menor valor en la función fitness, ya que se quieren minimizar los kilómetros recorridos.

El algoritmo genético propuesto es una adaptación del expuesto en [7] para el problema de las 8 reinas. Su código se puede ver en el Anexo C.3.

Capítulo 4

Desarrollo, modelización e implementación del problema

En este capítulo se va a explicar el planteamiento del problema. Como he mencionado en el Capítulo 1, el objetivo propuesto es minimizar la distancia que recorren todos los equipos de la NBA en la fase regular de la competición. Debido a la complejidad del problema este tiene que ser resuelto en dos fases: una primera fase en la que se determinan los trips y la secuencia de los partidos de todos los equipos y una segunda fase en la que se determina el día exacto en el que se va a jugar cada partido, teniendo en cuenta la secuencia de los partidos que proporciona la primera fase, obteniendo de esta manera un calendario optimizado de partidos.

4.1. 1ª Fase: Determinación de trips y secuencia de partidos

En primer lugar, cabe notar que en este primer modelo, ronda tiene el mismo significado que día, aunque no será hasta el segundo modelo cuando tomen importancia los días para poder construir el calendario de partidos.

4.1.1. Notación

La notación a emplear en el planteamiento de la 1ª fase del problema es la siguiente:

- N: número total de equipos.
- $E = \{E_1, \cdots, E_N\}$: conjunto de todos los equipos.
- $K = \{1, \cdots, D\}$: conjunto total de rondas en las que se pueden disputar los partidos.
- $S(i)$: conjunto de trips del equipo E_i con $i = 1, \cdots, N$.
- $S = \bigcup_{i \in E} S(i)$: conjunto de todos los posibles trips.
- $c(s)$: coste en kilómetros del trip $s \in S$.

En el problema que tenemos, el número de equipos que disputan la NBA es $N = 30$.
Definimos dos variables binarias:

- $Z_{sk} = \begin{cases} 1 \text{ si el trip } s \text{ se comienza a jugar en la ronda } k \in K, \text{ siendo } k \leq D - |s| + 1 \\ 0 \text{ en otro caso} \end{cases}$
- $X_{ijk} = \begin{cases} 1 \text{ si el equipo } i \text{ juega contra el equipo } j \text{ en el campo del equipo } i \text{ en la ronda } k, \text{ con } k \in K \\ 0 \text{ en otro caso} \end{cases}$

4.1.2. Restricciones

Antes de ver que restricciones se deben tener en cuenta y por qué, cabe notar que un equipo no va a jugar contra sí mismo, por ello la necesidad de indicar $\forall i, j \in E$ con $i \neq j$ en todas las restricciones. Las restricciones que se imponen se estructuran en seis grupos:

1. Es necesario que todos los partidos queden programados, es decir que todos los equipos jueguen contra todos.

$$\sum_{k \in K} X_{ijk} = 1 \quad \forall i, j \in E, i \neq j. \tag{4.1}$$

   Además con esta restricción, de manera implícita, obtengo que un partido no puede jugarse en dos rondas distintas ya que queda programado para una única ronda.

2. En una ronda un equipo juega a lo sumo un partido.

$$\sum_{\substack{j \in E \\ j \neq i}} \left(X_{ijk} + X_{jik}\right) \leq 1 \quad \forall i \in E, \forall k \in K. \tag{4.2}$$

3. Los partidos jugados deben respetar la secuencia definida por los trips de los equipos.

$$X_{ijk} = \sum_{s \in S(j)} Z_{s,(k-pos(s,i))} \quad \forall i, j \in E, i \neq j, \forall k \in K, \tag{4.3}$$

   donde $pos(s,i) \in \{0, \cdots, |s|-1\}$ es la posición del equipo i en la secuencia de partidos del trip s.

4. Después de un trip todos los equipos tienen que jugar al menos un partido en casa en las siguientes T rondas. De esta manera, en función del valor que tome T, se puede establecer que un equipo no pueda jugar dos trips seguidos.

$$\sum_{\substack{j \in E \\ j \neq i}} \sum_{t=1}^{T} X_{ij,k+|s|+t-1} \geq Z_{sk} \quad \forall i \in E, \forall k \in K, \forall s \in S(i) \text{ y } k + |s| - 1 < D. \tag{4.4}$$

5. Debido al gran número de kilómetros que recorren los equipos, es necesario que después de finalizar un trip descansen una ronda.

$$\sum_{j \in E} \left(X_{i,j,k+|s|} + X_{j,i,k+|s|}\right) \leq 2\left(1 - Z_{sk}\right) \quad \forall i \in E, \forall s \in S(i), \forall k \in K \text{ y } 1 < k \leq D - |s|. \tag{4.5}$$

6. Para que ningún equipo empiece muy tarde la competición imponemos esta restricción, que nos proporciona la seguridad de que todos los equipos jugaran al menos un partido en las H primeras rondas.

$$\sum_{\substack{j \in E \\ j \neq i}} \sum_{k=1}^{H} \left(X_{ijk} + X_{jik}\right) \geq 1 \quad \forall i \in E. \tag{4.6}$$

4.1.3. Función objetivo

El objetivo de esta primera fase, es minimizar el número total de kilómetros totales recorridos por todos los equipos.

$$\min \quad \sum_{i \in E} \sum_{s \in S(i)} \sum_{k=1}^{D} c(s) Z_{sk}. \tag{4.7}$$

4.1.4. Planteamiento

Cabe notar que la solución obtenida se utilizará para resolver la segunda fase del problema. De manera que el planteamiento de la 1ª fase del problema es el siguiente:

$$
\begin{cases}
\min & \displaystyle\sum_{i\in E}\sum_{s\in S(i)}\sum_{k=1}^{D} c(s)Z_{sk} \\
\text{s.a.} & \displaystyle\sum_{k\in K} X_{ijk} = 1 \quad \forall i,j \in E, i \neq j \\
& \displaystyle\sum_{\substack{j\in E \\ j\neq i}} \left(X_{ijk} + X_{jik}\right) \leq 1 \quad \forall i \in E, \forall k \in K \\
& \displaystyle X_{ijk} = \sum_{s\in S(j)} Z_{s,(k-pos(s,i))} \quad \forall i,j \in E, i \neq j, \forall k \in K \\
& \displaystyle\sum_{\substack{j\in E \\ j\neq i}}\sum_{t=1}^{T} X_{ij,k+|s|+t-1} \geq Z_{sk} \quad \forall i \in E, \forall k \in K, \forall s \in S(i) \text{ y } k+|s|-1 < D \\
& \displaystyle\sum_{j\in E} \left(X_{i,j,k+|s|} + X_{j,i,k+|s|}\right) \leq 2\left(1 - Z_{sk}\right) \quad \forall i \in E, \forall s \in S(i), \forall k \in K \text{ y } 1 < k \leq D - |s| \\
& \displaystyle\sum_{\substack{j\in E \\ j\neq i}}\sum_{k=1}^{H} \left(X_{ijk} + X_{jik}\right) \geq 1 \quad \forall i \in E \\
& X_{ijk} \in \{0,1\} \\
& Z_{sk} \in \{0,1\} \quad \text{con} \quad k \leq D - |s| + 1
\end{cases}
$$

La resolución de este problema nos definirá el orden en el que cada equipo debe jugar sus partidos. Aunque se ha planteado la existencia de rondas/días estos se ignoran y nos quedamos únicamente con la secuencia ordenada de partidos.

4.2. 2ª Fase: Creación del calendario de partidos

Para la resolución de esta segunda fase, partimos de la secuencia ordenada de partidos obtenida en la primera fase y se asigna cada uno de ellos a un día "real" de juego. Se podrá tener en cuenta las distintas necesidades de los equipos, como puede ser la disponibilidad de los pabellones, festividades, preferencias por jugar ciertos días, etc.

4.2.1. Notación

Además de la notación definida en la 1ª fase, hay que añadir:

- $T = \{1, \cdots, H\}$: conjunto total real de días en los que se pueden disputar los partidos.
- $G_i = \{1, \cdots, g_i\}$: conjunto ordenado de todos los partidos que tiene que jugar el equipo i, $\forall i \in E$, obtenido en la 1ª fase del problema.

Por otra parte se deben definir dos nuevas variables binarias:

- $Y_{ipt} = \begin{cases} 1 \text{ si el equipo } i \text{ juega el partido } p \in G_i \text{ en el día } t, \text{ con } t \in T \\ 0 \text{ en otro caso} \end{cases}$
- $da_{ip} = \begin{cases} 0 \text{ si el equipo } i \text{ tiene un día de descanso después de finalizar el trip } s \text{ al que} \\ \quad \text{pertenece el partido } p \in G_i \\ 1 \text{ en otro caso} \end{cases}$

4.2.2. Restricciones

1. Todos los partidos deben ser asignados a un día en concreto.

$$\sum_{t\in T} Y_{ipt} = 1 \quad \forall i \in E, \forall p \in G_i. \tag{4.8}$$

2. Cada día, un equipo puede jugar a lo sumo un partido.

$$\sum_{p\in G_i} Y_{ipt} \leq 1 \quad \forall i \in E, \forall t \in T. \tag{4.9}$$

3. Para que se respete el orden en el que se tienen que jugar los partidos, obtenidos en la primera fase del problema, se deben añadir las siguientes dos restricciones. Notar que $\sum_{s=1}^{D} Y_{i,p+1,t+s}$ obliga a que el partido siguiente se juegue dentro de los próximos D días; es decir no se permite estar más de D días sin jugar.

$$Y_{ipt} \leq \sum_{s=1}^{D} Y_{i,p+1,t+s} \quad \forall i \in E, \forall t \in T, t < H, t+s < H. \tag{4.10}$$

$$Y_{ipt} \leq \sum_{s=1}^{D} Y_{i,p-1,t-s} \quad \forall i \in E, \forall t \in T, t > 1, t-s > 0. \tag{4.11}$$

4. Si hay equipos que comparten pabellón (esto solo ocurre con los Lakers y los Clippers) estos no pueden jugar en casa en el mismo día; es por ello que hay que imponer la siguiente restricción.

$$Y_{ipt} + Y_{jqt} \leq 1 \quad \text{siendo } i = \text{Lakers, } j = \text{Clippers}, \forall p \in G_i, q \in G_j, \forall t \in T. \tag{4.12}$$

5. Se deben programar días de descanso entre partidos fuera de casa en un mismo trip. Se va a distinguir en función de la longitud del trip:

   - Si el trip está formado por un único equipo, no existe ningún tipo de descanso.
   - Si el trip está formado por dos equipos, siendo p el primer partido del trip, existirá un día de descanso entre los dos partidos o bien se jugarán en dos días consecutivos.

$$Y_{ipt} \leq \sum_{m=1}^{2} Y_{i,p+1,t+m} \quad \forall i \in E, \forall p \in G_i, p < g_i, \forall t \in T, t < H-1, |s| = 2. \tag{4.13}$$

   - Si el trip esta formado por tres equipos, siendo p el primer partido del trip, será necesario que exista un día de descanso, ya sea entre el primer y el segundo partido o entre el segundo y el tercero. De esta forma se juegan como máximo dos partidos seguidos.

$$Y_{ipt} \leq Y_{i,p+2,t+4} \quad \forall i \in E, \forall p \in G_i, p < g_i, \forall t \in T, t < H-3, |s| = 3. \tag{4.14}$$

6. Después de un trip se debe dejar un día de descanso para los jugadores; siendo p el último partido del trip, si bien puede incumplirse en cuyo caso da_{ip} tomará valor 1.

$$Y_{ipt} + Y_{i,p+1,t+2} \leq 1 + da_{ip} \quad \forall i \in E, \forall p \in G_i, p < g_i, \forall t \in T, t < H-1. \tag{4.15}$$

7. Un partido entre dos equipos se tiene que asignar en el mismo día.

$$Y_{ipt} = Y_{jpt} \quad \forall i, j \in E, i \neq j, \forall t \in T, \forall p \in G_i \cap G_j. \tag{4.16}$$

8. Para que un equipo no empiece excesivamente tarde la competición con respecto al resto, cada equipo deberá jugar al menos un partido en los primeros J días de juego.

$$\sum_{t=1}^{J} Y_{ipt} \geq 1 \quad \forall i \in E, \tag{4.17}$$

siendo p el primer partido de G_i.

Destacar que en EEUU, los pabellones se utilizan tanto para partidos de cualquier deporte como para eventos de otros tipos, es por ello que los equipos tienen la obligación de decir antes de que se diseñe el calendario de la siguiente temporada, que días tienen disponible el pabellón. En el problema que se esta tratando se ha supuesto que los pabellones están disponibles todos los días al no disponer de dicha información, si bien en caso de disponer de ella bastaría con imponer:

$$Y_{ipt} = 0,$$

para aquellos días t en los que el pabellón del equipo E_i no esté disponible, con $i = 1, \cdots, N$.

4.2.3. Función objetivo

En esta segunda fase del problema, adquiere importancia que se respeten los descansos después de un trip, así como que el número de días que dura la competición no sea excesivamente largo. Es por ello, que se plantea la siguiente función objetivo ponderada, en la que se le otorga mucha más importancia a los descansos, y dentro de las soluciones con el mismo nivel de descanso se intenta minimizar el número de días utilizados.

$$\min \quad 1000 \sum_{p \in G_i} da_{ip} + 0.01 \sum_{\substack{p \in G_i \\ t \in T}} t Y_{ipt}. \tag{4.18}$$

4.2.4. Planteamiento

Con todo lo anterior, el planteamiento de la 2ª fase del problema es el siguiente:

$$\begin{cases}
\min & 1000 \sum_{p \in G_i} da_{ip} + 0.01 \sum_{\substack{p \in G_i \\ t \in T}} t Y_{ipt} \\
\text{s.a.} & \sum_{t \in T} Y_{ipt} = 1 \quad \forall i \in E, \forall p \in G_i \\
& \sum_{p \in G_i} Y_{ipt} \leq 1 \quad \forall i \in E, \forall t \in T \\
& Y_{ipt} \leq \sum_{s=1}^{D} Y_{i,p+1,t+s} \quad \forall i \in E, \forall t \in T, t < H, t+s < H \\
& Y_{ipt} \leq \sum_{s=1}^{D} Y_{i,p-1,t-s} \quad \forall i \in E, \forall t \in T, t > 1, t-s > 0 \\
& Y_{ipt} + Y_{jqt} \leq 1 \quad \text{siendo } i = \text{Lakers}, \ j = \text{Clippers}, \forall p \in G_i, q \in G_j, \forall t \in T \\
& Y_{ipt} \leq \sum_{m=1}^{2} Y_{i,p+1,t+m} \quad \forall i \in E, \forall p \in G_i, p < g_i, \forall t \in T, t < H-1, |s| = 2 \\
& Y_{ipt} \leq Y_{i,p+2,t+4} \quad \forall i \in E, \forall p \in G_i, p < g_i, \forall t \in T, t < H-3, |s| = 3 \\
& Y_{ipt} + Y_{i,p+1,t+2} \leq 1 + da_{ip} \quad \forall i \in E, \forall p \in G_i, p < g_i, \forall t \in T, t < H-1 \\
& Y_{ipt} = Y_{jpt} \quad \forall i, j \in E, i \neq j, \forall t \in T, p \in G_i \cap G_j \\
& \sum_{t=1}^{J} Y_{ipt} \geq 1 \quad \forall i \in E \\
& Y_{ipt} \in \{0,1\} \\
& da_{ip} \in \{0,1\}
\end{cases}$$

4.3. Modelización

Como se ha comentado en el Capítulo 1, el TTP solo es resoluble para un tamaño de equipos pequeños (hasta el momento solo se ha resuelto para 12 equipos [3]). Para un número de equipos superior se buscará una solución aproximada.

Los planteamientos anteriores de la 1ª y 2ª fase se tratan de planteamientos ideales. Por ello, en función del número de equipos de los que se disponen puede no ser posible resolver la 1ª fase y hay que plantear aproximaciones. Se ha considerado dividir la 1ª fase en diferentes subproblemas, obteniendo de esta manera un sub-óptimo y no un óptimo si se resolviera de una sola vez. En nuestro caso, como se ha mencionado en ocasiones anteriores, el número de equipos es $N = 30$, y se ha dividido el problema en 4 subproblemas.

A la hora de resolver el problema, dividiéndolo en distintos subproblemas, la restricción (4.1) se debe suprimir y escribir en su lugar las siguientes restricciones:

1. Leer la solución del subproblema resuelto anteriormente. Se deben leer los partidos que ya han sido programados (XPrev), así como los trips que ya han sido utilizados (ZPrev). Estas variables deberán tomar valor 1.
$$X_{ijk} = 1 \quad \forall ijk \in XPrev \tag{4.19}$$
$$Z_{sk} = 1 \quad \forall sk \in ZPrev \tag{4.20}$$

2. No es necesario que todos los partidos queden programados.
$$\sum_{k \in K} X_{ijk} \leq 1 \quad \forall i, j \in E, i \neq j. \tag{4.21}$$

3. Se deben jugar al menos un cierto número de partidos P, en las D rondas disponibles.
$$\sum_{\substack{j \in E \\ j \neq i}} X_{ijk} \geq P \quad \forall i \in E, \forall k \in K, \tag{4.22}$$

4. Cada equipo debe jugar en las D rondas disponibles, un número de partidos comprendidos entre U y L.
$$U \leq \sum_{k \in K} \sum_{\substack{j \in E \\ j \neq i}} \left(X_{ijk} + X_{jik} \right) \leq L \quad \forall i \in E, \tag{4.23}$$
donde U es el número mínimo y L el número máximo de partidos que debe jugar cada equipo en las D rondas disponibles.

5. Cada equipo debe jugar en las D rondas disponibles, un número de partidos como local, comprendidos entre U_{loc} y L_{loc}.
$$U_{loc} \leq \sum_{k \in K} \sum_{\substack{j \in E \\ j \neq i}} X_{ijk} \leq L_{loc} \quad \forall i \in E, \tag{4.24}$$
donde U_{loc} es el número mínimo y L_{loc} el número máximo de partidos que debe jugar cada equipo como local en las D rondas disponibles.

6. Cada equipo debe jugar en las D rondas disponibles, un número de partidos como visitante, comprendidos entre U_{vis} y L_{vis}.
$$U_{vis} \leq \sum_{k \in K} \sum_{\substack{j \in E \\ j \neq i}} X_{jik} \leq L_{vis} \quad \forall i \in E, \tag{4.25}$$
donde U_{vis} es el número mínimo y L_{vis} el número máximo de partidos que debe jugar cada equipo como visitante en las D rondas disponibles.

Observar, que las restricciones (4.23) y (4.24) implican indirectamente la restricción (4.25) debido a que al establecer el número mínimo y máximo total de partidos que tiene que jugar cada equipo en cada subproblema y a su vez el intervalo de partidos que debe jugar como local, el resto de partidos necesariamente los deberá jugar como visitante.

Por otra parte, cabe notar las siguientes observaciones sobre algunas de las restricciones que se han tenido que añadir. La restricción (4.23) es necesaria para que no se decidan programar la mayoría de partidos de un equipo y ningún partido de algún equipo. Esto podría ocurrir debido a que el objetivo sigue siendo minimizar los kilómetros recorridos por todos los equipos, como consecuencia se programarían primero los partidos que implicasen recorrer una distancia menor. Por otra parte, las restricciones (4.24) y (4.25) son necesarias para que el número de partidos como local y visitante sean lo más similares posible entre equipos. En caso contrario podría ocurrir que en las D rondas disponibles tuviera que desplazarse en la mayoría de sus partidos y jugara como local muy pocos de ellos. Además, con las tres restricciones mencionadas se trata de que la solución que se obtiene en cada parte sea lo más realista posible.

4.3.1. Planteamiento subproblemas

Por lo tanto, el planteamiento del subproblema-r de la 1ª fase del problema, en caso de que este se deba resolver diviendolo en r subproblemas, es el siguiente:

$$
\begin{cases}
\min & \displaystyle\sum_{i\in E}\sum_{s\in S(i)}\sum_{k=1}^{D_r} c(s)Z_{sk} \\
\text{s.a.} & X_{ijk}=1 \quad \forall ijk\in XPrev \\
& Z_{sk}=1 \quad \forall sk\in ZPrev \\
& \displaystyle\sum_{k\in K} X_{ijk}\le 1 \quad \forall i,j\in E, i\neq j \\
& \displaystyle\sum_{\substack{j\in E\\ j\neq i}} X_{ijk}\ge P \quad \forall i\in E,\forall k\in K \\
& \displaystyle U\le \sum_{k\in K}\sum_{\substack{j\in E\\ j\neq i}}\left(X_{ijk}+X_{jik}\right)\le L \quad \forall i\in E \\
& \displaystyle U_{loc}\le \sum_{k\in K}\sum_{\substack{j\in E\\ j\neq i}} X_{ijk}\le L_{loc} \quad \forall i\in E \\
& \displaystyle\sum_{\substack{j\in E\\ j\neq i}}\left(X_{ijk}+X_{jik}\right)\le 1 \quad \forall i\in E,\forall k\in K \\
& \displaystyle X_{ijk}=\sum_{s\in S(j)} Z_{s,(k-pos(s,i))} \quad \forall i,j\in E, i\neq j,\forall k\in K \\
& \displaystyle\sum_{\substack{j\in E\\ j\neq i}}\sum_{t=1}^{T} X_{ij,k+|s|+t-1}\ge Z_{sk} \quad \forall i\in E,\forall k\in K,\forall s\in S(i) \text{ y } k+|s|-1<D_r \\
& \displaystyle\sum_{j\in E}\left(X_{i,j,k+|s|}+X_{j,i,k+|s|}\right)\le 2\left(1-Z_{sk}\right) \quad \forall i\in E,\forall s\in S(i),\forall k\in K \text{ y } 1<k\le D_r-|s| \\
& \displaystyle\sum_{\substack{j\in E\\ j\neq i}}\sum_{k=1}^{3}\left(X_{ijk}+X_{jik}\right)\ge 1 \quad \forall i\in E \\
& X_{ijk}\in\{0,1\} \\
& Z_{sk}\in\{0,1\} \quad \text{con} \quad k\le D_r-|s|+1
\end{cases}
$$

Observar, que para el caso del primer subproblema las restricciones 4.19 y 4.20 no son necesarias ya que no se disponen datos previos. Para los subproblemas intermedios, se debe leer la solución obtenida

en el subproblema anterior. Para el último subproblema, se deben leer las soluciones obtenidas de los subproblemas resueltos, pero el planteamiento es el comentado en un primer momento, con la restricción (4.1).

4.4. Implementación y resultados

En primer lugar, los problemas han sido resueltos con un ordenador con un sistema operativo LINUX XUBUNTU 20.04, con procesador AMD RYZ 7 3700X y con 32GB de RAM.

Como se ha mencionado en ocasiones anteriores, la modelización del problema se ha realizado mediante el programa informático CPLEX. Para resolver cada problema, ha sido necesario crear dos archivos, el primero de ellos donde se escribe la propia modelización y el segundo donde se leen los datos. Estos archivos tienen extensión .mod y .dat respectivamente y se pueden ver en el Anexo C.

Para la implementación y modelización del problema, lo primero que hay que hacer es generar un conjunto de posibles trips. Se ha considerado adecuado que los trips tengan como mucho tamaño 3, evitando de esta manera grandes periodos de los equipos fuera de casa.

Se presentan los resultados para un caso sencillo, formado por solo 12 equipos, y posteriormente para los 30 equipos de la NBA.

4.4.1. Estudio 12 equipos

Para la creación de los trips a través de técnicas cluste se ha aplicado el método *K-Means Constrained* con $N = n$ siendo $n = 2, 3$. Para el caso de algoritmos genéticos se han utilizado dos patrones distintos, el primero de ellos busca 3 trips de tamaño 3 y 1 trip de tamaño 2. Mientras, que el segundo busca 5 trips de tamaño 2. Además, hay que añadir los trips de tamaño 1. Estos han sido calculados con R usando la tabla de distancias.

En este caso, no es necesario crear subproblemas y se puede modelar y resolver directamente los problemas planteados en 4.1.4 y 4.2.4. Los programas de CPLEX se pueden ver en el Anexo C.4.1 (el archivo .dat es el correspondiente a trips creados a través de algoritmos genéticos). Comentar que este será el único caso en el que mostrare el archivo .dat debido a su extensión y la similitud que existe con el resto.

Para la resolución de la primera fase se han utilizado los siguientes parámetros:

$$D = 50 \qquad T = 3 \qquad H = 3$$

donde D indica el número de rondas disponibles para planificar los partidos, T el número de rondas siguientes en las que se tiene que jugar un partido en casa después de finalizar un trip y por último en las H primeras rondas de la competición todos los equipos han tenido que jugar al menos un partido.

En este caso los problemas han sido resueltos hasta la optimalidad. Los tiempos de ejecución, así como el VFO, el numero de restricciones y de variables que tiene cada problema, en cada caso han sido:

- **Utilizando únicamente trips formados con técnicas cluster**. Se obtiene un VFO de: 243447.804 km. El problema presenta 30336 restricciones y 18456 variables binarias. El tiempo de ejecución ha sido de, aproximadamente 15 minutos.
- **Utilizando trips creados a través de algoritmos genéticos**. Se obtiene un VFO de: 208929.912 km, es decir un 14 % menos que en el caso anterior. El problema presenta 30336 restricciones y 18456 variables binarias. El tiempo de ejecución ha sido de, aproximadamente 8 minutos.
- **Juntando tips formados con técnicas cluster y algoritmos genéticos**. Se obtiene un VFO de: 208929.912 km. La solución es similar a la anterior, difiere únicamente en el orden en el que se recorren las ciudades de un trip (aparece invertido). El problema presenta 40524 restricciones y 23712 variables binarias. El tiempo de ejecución ha sido de, aproximadamente 30 minutos. Cabe notar, que como se ha obtenido el mismo VFO que utilizando algoritmos genéticos, esto nos puede indicar que dicha idea ha sido acertada.

A la vista de los resultados obtenidos, para resolver la segunda fase, se ha utilizado la solución obtenida al utilizar trips creados a través de algoritmos genéticos. Los programas de CPLEX se pueden ver en el Anexo C.4.5.

Los parámetros utilizados para la resolución de la segunda fase del problema han sido:

$$H = 65 \qquad R = 21 \qquad D = 9 \qquad J = 4$$

donde H indica los días disponibles para jugar los partidos, R los partidos que tiene que jugar cada equipo (empezando a contar en 0), D el número máximo de días que un equipo puede estar sin jugar ningún partido y, en los primeros J días de juego todos los equipos han tenido que jugar al menos un partido.

El VFO ha sido: 3047.92, es decir se producen 3 incumplimientos de descansos y se han programado todos los partidos en 64 días. El problema presenta 78861 restricciones y 18912 variables binarias. El tiempo de ejecución ha sido de, aproximadamente 1 minuto.

Los kilómetros que tendrá que recorrer cada equipo son, aproximadamente:

- Hawks: 15271 km
- Bulls: 14498 km
- Pistons: 14829 km
- Celtics: 19310 km
- Cavaliers: 14816 km
- Warriors: 28935 km
- Nets: 17826 km
- Mavericks: 16756 km
- Rockets: 17625 km
- Hornets: 15331 km
- Nuggets: 19737 km
- Pacers: 13988 km

Observamos que el equipo que más kilómetros tiene que recorrer es Warriors mientras que el que menos es Pacers. Esto es debido a la ubicación de cada uno de ellos, Pacers se encuentra próximo a la gran mayoria de los equipos (en el centro de Estados Unidos), lo contrario ocurre con Warriors (en la costa Oeste).

Si comparamos la distancia obtenida que tiene que recorrer cada equipo, con la distancia obtenida en los algoritmos genéticos para trips de tamaño 3, que podría considerarse como una buena cota inferior de los kilómetros a recorrer por cada equipo, se puede observar que esta es exactamente la misma. Esto se debe a que el programa, los únicos trips que ha escogido han sido los creados para trips de tamaño 3 en el algoritmo genético. Se puede ver el calendario creado en el Anexo D.

En la solución, se puede ver que no todo es perfecto debido a que en ocasiones aparecen ciertos periodos de descanso entre partidos demasiados largos. Esto es debido a que no se ha incorporado en el modelo propuesto una restricción del siguiente tipo:

$$\sum_{\substack{j \in E \\ j \neq i}} \sum_{m=0}^{M} \left(X_{ij,k+m} + X_{ji,k+m} \right) \geq 1 \quad \forall i \in E, \forall k \in K. \tag{4.26}$$

Esta garantizaría que en los M días siguientes a disputar un partido, se tendría que jugar el siguiente partido. Esta restricción se incorporó inicialmente pero con ella no se conseguía resolver el problema en un tiempo razonable incluso para 12 equipos.

4.4.2. Estudio NBA

Para la creación de los trips a través de técnicas cluster se ha aplicado el método *K-Means Constrained* con $N = n$ siendo $n = 2, 3$. Para el caso de algoritmos genéticos se han utilizado dos patrones distintos. El primero de ellos busca 9 trips de tamaño 3 y 1 trip de tamaño 2, mientras, que el segundo busca 14 trips de tamaño 2. Además, hay que añadir los trips de tamaño 1. Estos han sido calculados en R usando la tabla de distancias.

En este caso, ha sido necesario dividir la primera fase del problema en 4 subproblemas, modelando y resolviendo el problema planteado en 4.3.1 para el caso de los 3 primeros subproblemas. Para el

subproblema 1-4 se debe hacer uso de la solución obtenida hasta el momento en el subproblema 1-3 y proceder a resolver el problema 4.1.4. Los programas de CPLEX se pueden ver en los Anexos C.4.2, C.4.3 y C.4.4. El programa del subproblema 1-3 no se muestra en el anexo debido a que es análogo al subproblema 1-2.

Para la resolución de la primera fase se han utilizado los siguientes parámetros:

- **Subproblema1-1**
  - $D = 30$
  - $L = 17$
  - $T = 3$
  - $Uloc = 6$
  - $H = 3$
  - $Lloc = 9$
  - $P = 217$
  - $Uvis = 6$
  - $U = 13$
  - $Lvis = 9$
- **Subproblema1-2**
  - $D = 60$
  - $L = 34$
  - $T = 3$
  - $Uloc = 12$
  - $H = 3$
  - $Lloc = 18$
  - $P = 434$
  - $Uvis = 12$
  - $U = 26$
  - $Lvis = 18$
- **Subproblema1-3**
  - $D = 90$
  - $L = 51$
  - $T = 3$
  - $Uloc = 18$
  - $H = 3$
  - $Lloc = 27$
  - $P = 651$
  - $Uvis = 18$
  - $U = 39$
  - $Lvis = 27$

Además, para el subproblema 1-4 los parámetros que se han utilizado son:

$$D = 120 \qquad T = 3 \qquad H = 3$$

A diferencia del caso anterior, los subproblemas se han resuelto con un gap[1] inferior al 5 % en vez de hasta la optimalidad. Los tiempos de ejecución, así como el VFO, el número de restricciones y de variables que tiene cada subproblema, en cada caso han sido:

- **Utilizando únicamente trips formados con técnicas cluster**. El subproblema 1-1 presenta 116641 restricciones y 72810 variables binarias. El tiempo de ejecución ha sido de, aproximadamente 8 horas y 30 minutos. El subproblema 1-2 tiene 239365 restricciones y 146610 variables binarias. Su tiempo de ejecución ha sido de, aproximadamente 1 hora. El subproblema 1-3 presenta 362067 restricciones y 220410 variables binarias, y su tiempo de ejecución ha sido de, aproximadamente 16 minutos. Por último en el subproblema 1-4 se obtiene un VFO de: 1620860.383 km, y tiene 484675 restricciones y 294210 variables binarias. Su tiempo de ejecución ha sido de, aproximadamente 19 minutos.

- **Utilizando trips creados a través de algoritmos genéticos**. El subproblema 1-1 presenta 116641 restricciones y 72810 variables binarias. El tiempo de ejecución ha sido de, aproximadamente 7 horas y 30 minutos. El subproblema 1-2 tiene 239348 restricciones y 146610 variables binarias. Su tiempo de ejecución ha sido de, aproximadamente 1 hora y 40 minutos. El subproblema 1-3 presenta 362045 restricciones y 220410 variables binarias, y su tiempo de ejecución ha sido de, aproximadamente 36 minutos. Por último en el subproblema 1-4 se obtiene un VFO de: 1466098.956 km, es decir un 10 % menos que en el caso anterior. EL problema tiene 484653 restricciones y 294210 variables binarias. Su tiempo de ejecución ha sido de, aproximadamente 20 minutos.

- **Juntando tips formados con técnicas cluster y algoritmos genéticos**. El subproblema 1-1 presenta 155.701 restricciones y 93420 variables binarias. El tiempo de ejecución ha sido de, aproximadamente 17 horas y 30 minutos. El subproblema 1-2 tiene 321608 restricciones y 188820 variables binarias. Su tiempo de ejecución ha sido de, aproximadamente 10 horas y 30 minutos. El subproblema 1-3 presenta 487527 restricciones y 284220 variables binarias, y su tiempo de ejecución ha sido de, aproximadamente 2 horas y 30 minutos. Por último en el subproblema 1-4 se obtiene un VFO de: 1454301.528 km, es decir un 1 % menos que en el caso anterior. El problema tiene 653311 restricciones y 379620 variables binarias. Su tiempo de ejecución ha sido de, aproximadamente 30 minutos.

[1] $\text{Gap} = \dfrac{|\text{Mejor solución} - \text{Mejor solucion entera}|}{e^{-10} + |\text{Mejor solución entera}|}$.

Se puede dar como mejor solución la obtenida utilizando trips formados con técnicas cluster y algoritmos genéticos. Examinando los resultados obtenidos, se ha observado que los trips seleccionados son en su mayoría provenientes del algoritmo genético, aunque a diferencia de lo que ocurre con el modelo de 12 equipos, esta vez sí que han sido seleccionados trips creados a través de técnicas cluster. Además se hacen uso de trips de tamaño 1. Esto nos índica, que la construcción de trips a través del diseño de un algoritmo genético es indispensable, pero que convenientemente debería ir acompañada de trips creados a través de otras técnicas como pueden ser las técnicas cluster.

Para la resolución de la segunda fase, se debería utilizar la solución obtenida al utilizar tanto los trips creados con técnicas cluster como los trips creados a través de algoritmos genéticos. Sin embargo, esta ha resultado imposible de resolver. El modelo con el que se ha probado la resolución es el mismo que para el caso de 12 equipos, que se puede ver en el Anexo C.4.5. Dependiendo de los parámetros elegidos se han tenido problemas de factibilidad o de no finalizar por excesivo tiempo de computación.

Los kilómetros que tendrá que recorrer cada equipo son, aproximadamente:

- Hawks: 45002 km
- Celtics: 57267 km
- Nets: 54487 km
- Hornets: 38582 km
- Bulls: 38061 km
- Cavaliers: 50524 km
- Mavericks: 42953 km
- Nuggets: 42895 km
- Pistons: 39376 km
- Warriors: 62925 km
- Rockets: 45078 km
- Pacers: 34973 km
- Clippers: 74927 km
- Lakers: 64693 km
- Grizzlies: 35704 km
- Heat: 54749 km
- Bucks: 36260 km
- Timberwolves: 36771 km
- Pelicans: 42682 km
- Knicks: 44709 km
- Thunder: 42729 km
- Magic: 57602 km
- 76ers: 46293 km
- Suns: 56058 km
- TrailBlazers: 68058 km
- Kings: 65215 km
- Spurs: 45349 km
- Raptors: 40075 km
- Jazz: 48960 km
- Wizards: 41331 km

Observamos que el equipo que más kilómetros tiene que recorrer es Clippers mientras que el que menos es Pacers. Esto es debido a la ubicación de cada uno de ellos, Pacers se encuentra próximo a la gran mayoria de los equipos, lo contrario ocurre con Clippers.

Si comparamos la distancia obtenida que tiene que recorrer cada equipo, con la distancia obtenida en los algoritmos genéticos para trips de tamaño 3 (esta distancia es una de las mejores para cada equipo de visitar a los otros 29 equipos, sin tenerlos en cuenta), se puede observar que esta es, en termino medio, un 10 % peor con respecto a los genéticos (el valor mínimo que se obtiene es de un 0 %, mientras que el valor máximo es: 24 %). Esto se debe a que el programa, aparte de trips creados con algoritmos genéticos, ha escogido alguno creado con técnicas cluster para poder compatibilizar las planificaciones de los equipos de la mejor manera.

Capítulo 5

Conclusiones

Como bien se ha comentado en el Capítulo 1, el problema al que nos enfrentábamos era muy complejo, entre otros factores por el gran número de equipos $N = 30$. Por ello, el problema ha sido resuelto de manera completa para 12 equipos, y para la actual estructura de la NBA, 30 equipos, se ha proporcionado una solución parcial en la que se ha minimizado en la medida de lo posible el número de kilómetros recorridos.

El trabajo realizado me ha permitido ampliar mis conocimientos en la programación lineal entera, aprender técnicas cluster e iniciar el estudio y diseño de algoritmos genéticos, además de enfrentarme a un problema real y ver las complicaciones que surgen a la hora de modelar estos problemas.

En el desarrollo del trabajo he estudiado dos herramientas para construir los trips, siendo el diseño de un algoritmo genético la que mejor resultado ha proporcionado. Además, he creado dos modelos de programación lineal entera para la resolución del problema. El primero de ellos ha proporcionado el orden en el que se deben de jugar los partidos, minimizando los kilómetros totales que recorre cada equipo. Mientras que el segundo ha asignado el día exacto de juego de cada partido, minimizando el número de incumplimientos de descansos después de jugar un trip y los días de juego.

Se ha observado que el diseño de un algoritmo genético ha proporcionado trips de gran calidad debido a que el modelo ha seleccionado una gran mayoría de estos para la resolución del problema.

Como se ha mencionado a lo largo de la memoria, el problema ha podido ser resuelto de manera completa para 12 equipos, llegando hasta la optimalidad de los resultados. Sin embargo, para 30 equipos se ha logrado resolver el problema 1, teniéndolo que dividir en 4 subproblemas, con una solución de un 5 % de gap.

Como punto negativo para este caso, señalar la imposibilidad de encontrar soluciones para el segundo problema, bien por problemas de infactibilidad provocadas por las excesivas separaciones entre partidos que genera el primer problema, o bien porque el modelo resulta demasiado complejo y la máquina no ha podido resolverlo. Señalar que en este caso se trató de modelar un proceso de división en subproblemas, pero no se logró un modelado adecuado. Por este motivo, algunas futuras líneas de mejora podrían ser:

- Incluir elementos adicionales en el primer problema para tratar de reducir el tamaño de los periodos de inactividad.
- En el segundo problema, permitir alterar el orden de algún trip (adelantarlo o retrasarlo) con objeto de facilitar la distribución de los partidos y sus descansos. A priori, esta idea sería muy complicada de implementar pero de conseguirla seguro que mejoraría la calidad de las soluciones.
- También en el segundo problema se podría permitir que los trips se recorrieran en sentido inverso, esto es posible ya que los kilómetros seguirían siendo los mismos y el VFO no cambiaría.
- Permitir diferentes estructuras de descansos dentro de los trips, permitiendo que un trip se jugara en más días. De esta manera se flexibilizaría la restricción actual.

Bibliografía

[1] Colaborativo. National Basketball Association. https://es.wikipedia.org/wiki/National_Basketball_Association#Sistema_de_competici{ó}n, 2021.

[2] Mundo Deportivo. Revolución NBA: los cambios en su sistema de competición. https://www.mundodeportivo.com/baloncesto/nba/20191225/472474477797/revolucion-nba-los-cambios-en-su-sistema-de-competicion.html, 2019.

[3] Guillermo Durán. Sports scheduling and other topics in sports analytics: a survey with special reference to Latin America. *TOP*, 29(1):125–155, 2021.

[4] Guillermo Durán, Santiago Durán, Javier Marenco, Federico Mascialino, and Pablo A. Rey. Scheduling Argentina's professional basketball leagues: A variation on the Travelling Tournament Problem. *European Journal of Operational Research*, 275(3):1126–1138, jun 2019.

[5] Kelly Easton, George Nemhauser, and Michael Trick. The traveling tournament problem description and benchmarks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 2239, pages 580–584. Springer Verlag, 2001.

[6] Kelly Easton, George Nemhauser, and Michael Trick. Solving the Travelling Tournament Problem: A combined integer programming and constraint programming approach. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2740:100–109, 2003.

[7] Eiben and J.E. Smith. *Introduction to Evolutionary Computing*, volume 33. Springer Verlag, 2010.

[8] F S Hillier and Lieberman G H. *Introducción a la investigación de operaciones*. McGraw-Hill Ltd. Interamericana, México DF, 10 edition, 2015.

[9] Tomás Malón. Manual de uso IBM ILOG CPLEX. Departamento de Métodos Estadísticos. Universidad de Zaragoza. 2020.

[10] Jean Monlog. Clustering into same size clusters. https://jmonlong.github.io/Hippocamplus/2018/06/09/cluster-same-size/, 2018.

[11] Jean Monlog. Clustering into same size clusters. Codes. https://github.com/jmonlong/Hippocamplus/blob/master/content/post/2018-06-09-ClusterEqualSize.Rmd, 2018.

[12] José Ignacio Pinilla. Los equipos recorrerán 2.222.641 kilómetros: 56 vueltas al mundo. https://as.com/baloncesto/2017/08/17/nba/1502994836_523703.html, 2017.

Anexos

Anexo A

NBA

Como bien se ha comentado la NBA está formada por 30 equipos (o franquicias). Estos son los siguientes:

1. Atlanta Hawks (Hawks)
2. Boston Celtics (Celtics)
3. Brooklyn Nets (Nets)
4. Charlotte Hornets (Hornets)
5. Chicago Bulls (Bulls)
6. Cleveland Cavaliers (Cavaliers)
7. Dallas Mavericks (Mavericks)
8. Denver Nuggets (Nuggets)
9. Detroit Pistons (Pistons)
10. Golden State Warriors (Warriors)
11. Houston Rockets (Rockets)
12. Indiana Pacers (Pacers)
13. Los Angeles Clippers (Clippers)
14. Los Angeles Lakers (Lakers)
15. Memphies Grizzlies (Grizzlies)
16. Miami Heat (Heat)
17. Milwaukee Bucks (Bucks)
18. Minnesota Timberwolves (Timberwolves)
19. New Orleans Pelicans (Pelicans)
20. New York Knicks (Knicks)
21. Oklahoma City Thunder (Thunder)
22. Orlango Magic (Magic)
23. Philadelphia 76ers (76ers)
24. Phoenix Suns (Suns)
25. Portland Trail Blazers (Trail Blazers)
26. Sacramento Kings (Kings)
27. San Antonio Spurs (Spurs)
28. Toronto Raptors (Raptors)
29. Utah Jazz (Jazz)
30. Washington Wizards (Wizards)

Debido a la extensión del nombre de los equipos, siempre me referiré a ellos con el nombre escrito entre paréntesis.

En la Figura A.1 (obtenida de [1]) se muestra la localización de cada uno de los equipos. Notar que Lakers y Clippers están situados en la misma ciudad. Un caso análogo ocurre con Nets y Knicks que están situados a una distancia inferior de 10 km. Además, al querer minimizar las distancias que recorren los equipos, es de gran importancia saber cuál es la distancia entre todas las ciudades. Por ello, sabiendo la latitud y longitud de cada ciudad se ha podido calcular la distancia entre todas ellas haciendo uso de *Excel* y de la siguiente función:

$$\begin{aligned}\textbf{d(C1,C2)} &= 6371 \cdot \arccos\left(\cos\left(90 - \text{ Latitud1 }\right)\right) \cdot \cos\left(90 - \text{ Latitud2 }\right) \\ &+ \sin\left(90 - \text{ Latitud1 }\right) \cdot \sin\left(90 - \text{ Latitud2 }\right) \cdot \cos\left(\text{ Longitud1} - \text{ Longitud2 }\right)\end{aligned} \tag{A.1}$$

siendo $C1$ y $C2$ las ciudades entre las que se calcula la distancia, Latitud1 y Longitud1 los datos correspondientes a $C1$ y Latitud2 y Longitud2 a $C2$. Estas distancias se pueden consultar en la Tabla A.1 y A.2.

Una vez vista la localización de los equipos, es momento de decir qué equipos forman parte de la Conferencia Este y cuáles la Conferencia Oeste. Además, dentro de cada conferencia voy a indicar el nombre de cada división y los equipos que la forman.

Conferencia Este

- División Sureste
  - Hawks
  - Hornets
  - Heat
  - Wizards
  - Magic
- División Central
  - Bulls
  - Cavaliers
  - Pistons
  - Pacers
  - Bucks
- División Atlántico
  - Celtics
  - Nets
  - Knicks
  - 76ers
  - Raptors

Conferencia Oeste

- División Noroeste
  - Nuggets
  - Timberwolves
  - Thunder
  - Trail Blazers
  - Jazz
- División Pacífico
  - Warriors
  - Clippers
  - Lakers
  - Suns
  - kings
- División Sureste
  - Mavericks
  - Rockets
  - Grizzlies
  - Pelicans
  - Spurs

En el Capítulo 1 ya he explicado el funcionamiento de la fase regular, a continuación voy a explicar como funciona la fase final. En el presente trabajo únicamente he creado un nuevo diseño para la fase regular, por lo tanto, la fase final se debería seguir disputándose de la misma manera que hasta ahora pero con los emparejamientos correspondientes.

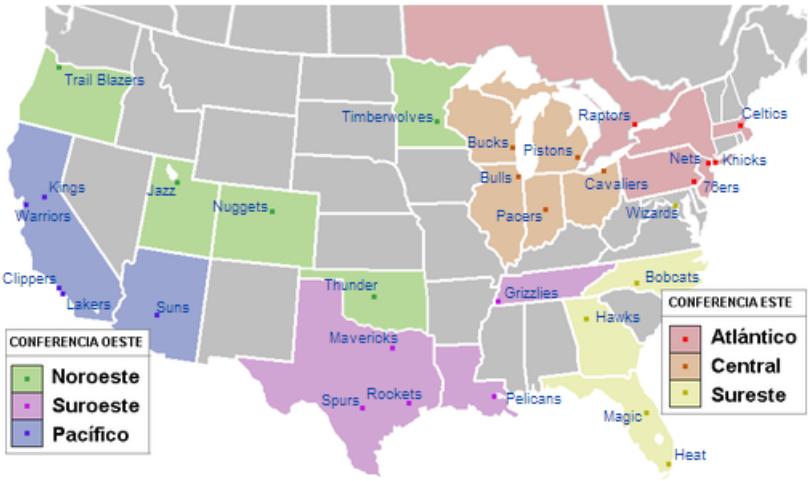

Figura A.1: Localización de los equipos de la NBA.

Una vez establecida una clasificación del 1 al 15 en cada conferencia, al finalizar la fase regular, los 8 mejores equipos de cada conferencia serán los que disputen la fase final. Otro nombre para esta fase es *playoff*. Los playoff constan de las siguientes rondas: 1ª ronda, semifinales de conferencia, finales de conferencia y final de la NBA. El funcionamiento del playoff es el siguiente:

- La primera ronda de playoff consta de los emparejamientos que se muestran a continuación:

$$1\begin{cases}1^{o}\text{Conferencia Oeste}\\ \text{vs}\\ 8^{o}\text{Conferencia Oeste}\end{cases} \qquad 5\begin{cases}1^{o}\text{Conferencia Este}\\ \text{vs}\\ 8^{o}\text{Conferencia Este}\end{cases}$$

$$2\begin{cases}4^{o}\text{Conferencia Oeste}\\ \text{vs}\\ 5^{o}\text{Conferencia Oeste}\end{cases} \qquad 6\begin{cases}4^{o}\text{Conferencia Este}\\ \text{vs}\\ 5^{o}\text{Conferencia Este}\end{cases}$$

$$3\begin{cases}3^{o}\text{Conferencia Oeste}\\ \text{vs}\\ 6^{o}\text{Conferencia Oeste}\end{cases} \qquad 7\begin{cases}3^{o}\text{Conferencia Este}\\ \text{vs}\\ 6^{o}\text{Conferencia Este}\end{cases}$$

$$4\begin{cases}2^{o}\text{Conferencia Oeste}\\ \text{vs}\\ 7^{o}\text{Conferencia Oeste}\end{cases} \qquad 8\begin{cases}2^{o}\text{Conferencia Este}\\ \text{vs}\\ 7^{o}\text{Conferencia Este}\end{cases}$$

- Los emparejamientos que se dan en las semifinales de conferencia son los siguientes:

$$9\begin{cases}\text{Ganador de 1}\\ \text{vs}\\ \text{Ganador de 2}\end{cases} \qquad 11\begin{cases}\text{Ganador de 5}\\ \text{vs}\\ \text{Ganador de 6}\end{cases}$$

$$10\begin{cases}\text{Ganador de 3}\\ \text{vs}\\ \text{Ganador de 4}\end{cases} \qquad 12\begin{cases}\text{Ganador de 7}\\ \text{vs}\\ \text{Ganador de 8}\end{cases}$$

- En la final de cada conferencia se juega el siguiente partido:

$$13\begin{cases}\text{Ganador de 9}\\ \text{vs}\\ \text{Ganador de 10}\end{cases} \qquad 14\begin{cases}\text{Ganador de 11}\\ \text{vs}\\ \text{Ganador de 12}\end{cases}$$

  De la eliminatoria 13 saldrá el ganador de la Conferencia Oeste y de la eliminatoria 14 el ganador de la Conferencia Este.

- La final de la NBA, la juegan los ganadores de cada conferencia:

$$\begin{cases}\text{Ganador de 13}\\ \text{vs}\\ \text{Ganador de 14}\end{cases}$$

Todas las eliminatorias de playoff se juegan con el formato al mejor de 7 partidos; esto es, una vez que un equipo gana 4 partidos, pasa a la siguiente eliminatoria sin necesidad de disputar los partidos restantes. En todas las eliminatorias los partidos 1,2,5,7 se juegan en el campo del equipo local mientras que los partidos 3,4,6 se harán en el campo del equipo visitante, siendo de los dos equipo que vayan a disputar la eliminatoria el que actúe como equipo local aquel que haya conseguido una mejor clasificación durante la fase regular.

Latidud	33,7629	42,332	40,6501	35,22709	41,8379	41,4995	32,7942	39,7618	42,383	37,7272	29,7805	39,76838	34,0194	34,0194	35,14953
Longitud	-84,4227	-71,0202	-73,94958	-80,84313	-87,6828	-81,69541	-96,7655	-104,881	-83,1022	-123,032	-95,3863	-86,15804	-118,411	-118,411	-90,04898
EQUIPOS	HAWKS	CELTICS	NETS	HORNETS	BULLS	CAVALIERS	MAVERICKS	NUGGETS	PISTONS	WARRIORS	ROCKETS	PACERS	CLIPPERS	LAKERS	GRIZZLIES
HAWKS	0	1508	1201	366	942	893	1152	1935	965	3487	1126	685	3123	3123	538
CELTICS	1508	0	307	1160	1374	888	2492	2835	992	4389	2583	1299	4188	4188	1828
NETS	1201	307	0	853	1155	656	2205	2615	786	4188	2283	1041	3957	3957	1536
HORNETS	366	1160	853	0	945	701	1491	2173	819	3751	1490	689	3420	3420	836
BULLS	942	1374	1155	945	0	499	1285	1463	383	3033	1508	263	2814	2814	772
CAVALIERS	893	888	656	701	499	0	1646	1960	152	3532	1792	423	3306	3306	1014
MAVERICKS	1152	2492	2205	1491	1285	1646	0	1062	1604	2438	360	1225	2010	2010	672
NUGGETS	1935	2835	2615	2173	1463	1960	1062	0	1844	1588	1407	1597	1360	1360	1404
PISTONS	965	992	786	819	383	152	1604	1844	0	3406	1780	387	3197	3197	1004
WARRIORS	3487	4389	4188	3751	3033	3532	2438	1588	3406	0	2692	3183	586	586	2949
ROCKETS	1126	2583	2283	1490	1508	1792	360	1407	1780	2692	0	1392	2219	2219	779
PACERS	685	1299	1041	689	263	423	1225	1597	387	3183	1392	0	2922	2922	618
CLIPPERS	3123	4188	3957	3420	2814	3306	2010	1360	3197	586	2219	2922	0	0	2591
LAKERS	3123	4188	3957	3420	2814	3306	2010	1360	3197	586	2219	2922	0	0	2591
GRIZZLIES	538	1828	1536	836	772	1014	672	1404	1004	2949	779	618	2591	2591	0
HEAT	977	2023	1752	1053	1913	1754	1783	2768	1866	4221	1557	1652	3772	3772	1405
BUCKS	1075	1380	1185	1060	135	539	1377	1459	399	3010	1617	392	2818	2818	896
TIMBERWOLVES	1311	1687	1508	1355	415	869	1329	1186	725	2704	1623	667	2545	2545	1005
PELICANS	662	2168	1861	1025	1324	1470	716	1734	1499	3154	527	1132	2711	2711	565
KNICKS	1203	306	2	855	1155	657	2206	2616	786	4188	2284	1041	3957	3957	1538
THUNDER	1212	2405	2135	1511	1108	1529	305	805	1462	2285	663	1108	1912	1912	678
MAGIC	665	1803	1516	759	1600	1455	1556	2500	1562	3989	1376	1339	3564	3564	1115
76ERS	1078	430	123	732	1073	577	2089	2526	717	4104	2161	940	3861	3861	1419
SUNS	2553	3697	3448	2863	2331	2809	1426	951	2712	1098	1630	2408	589	589	2028
TRAILBLAZERS	3485	4078	3930	3675	2819	3296	2625	1584	3153	869	2947	3025	1330	1330	2969
KINGS	3347	4228	4029	3604	2875	3374	2316	1436	3247	165	2582	3029	578	578	2809
SPURS	1416	2842	2547	1776	1688	2022	409	1289	1993	2444	303	1609	1948	1948	1017
RAPTORS	1187	699	564	950	705	308	1934	2149	333	3692	2095	710	3507	3507	1316
JAZZ	2540	3373	3173	2774	2019	2517	1609	605	2390	1016	1927	2178	945	945	2008
WIZARS	873	635	328	529	959	490	1898	2387	642	3972	1961	790	3708	3708	1227

Tabla A.1: Distancia entre los equipos de la NBA

Latidud	25,7751	43,0389	44,0923	30,0675	40,6643	35,46756	28,4159	40,0094	33,44838	45,5369	38,58157	29,42412	43,70011	40,76078	38,89511
Longitud	-80,2011	-87,9065	-91,7499	-89,9272	-73,9385	-97,51643	-81,2988	-75,1333	-112,074	-122,65	-121,494	-98,49363	-79,4163	-111,891	-77,03637
EQUIPOS	HEAT	BUCKS	TIMBERWOLVES	PELICANS	KNICKS	THUNDER	MAGIC	76ERS	SUNS	TRAILBLAZERS	KINGS	SPURS	RAPTORS	JAZZ	WIZARS
HAWKS	977	1075	1311	662	1203	1212	665	1078	2553	3485	3347	1416	1187	2540	873
CELTICS	2023	1380	1687	2168	306	2405	1803	430	3697	4078	4228	2842	699	3373	635
NETS	1752	1185	1508	1861	2	2135	1516	123	3448	3930	4029	2547	564	3173	328
HORNETS	1053	1060	1355	1025	855	1511	759	732	2863	3675	3604	1776	950	2774	529
BULLS	1913	135	415	1324	1155	1108	1600	1073	2331	2819	2875	1688	705	2019	959
CAVALIERS	1754	539	869	1470	657	1529	1455	577	2809	3296	3374	2022	308	2517	490
MAVERICKS	1783	1377	1329	716	2206	305	1556	2089	1426	2625	2316	409	1934	1609	1898
NUGGETS	2768	1459	1186	1734	2616	805	2500	2526	951	1584	1436	1289	2149	605	2387
PISTONS	1866	399	725	1499	786	1462	1562	717	2712	3153	3247	1993	333	2390	642
WARRIORS	4221	3010	2704	3154	4188	2285	3989	4104	1098	869	165	2444	3692	1016	3972
ROCKETS	1557	1617	1623	527	2284	663	1376	2161	1630	2947	2582	303	2095	1927	1961
PACERS	1652	392	667	1132	1041	1108	1339	940	2408	3025	3029	1609	710	2178	790
CLIPPERS	3772	2818	2545	2711	3957	1912	3564	3861	589	1330	578	1948	3507	945	3708
LAKERS	3772	2818	2545	2711	3957	1912	3564	3861	589	1330	578	1948	3507	945	3708
GRIZZLIES	1405	896	1005	565	1538	678	1115	1419	2028	2969	2809	1017	1316	2008	1227
HEAT	0	2043	2287	1068	1754	1972	313	1651	3183	4351	4099	1846	1994	3358	1489
BUCKS	2043	0	331	1453	1185	1179	1730	1114	2353	2758	2850	1784	690	1994	1021
TIMBERWOLVES	2287	331	0	1568	1509	1078	1975	1442	2113	2428	2543	1737	988	1690	1352
PELICANS	1068	1453	1568	0	1863	929	857	1738	2123	3316	3032	830	1778	2309	1533
KNICKS	1754	1185	1509	1863	0	2136	1518	125	3449	3930	4030	2548	563	3173	330
THUNDER	1972	1179	1078	929	2136	0	1716	2026	1352	2388	2150	678	1795	1386	1850
MAGIC	313	1730	1975	857	1518	1716	0	1407	2977	4078	3861	1676	1708	3097	1230
76ERS	1651	1114	1442	1738	125	2026	1407	0	3347	3869	3947	2427	542	3091	205
SUNS	3183	2353	2113	2123	3449	1352	2977	3347	0	1619	1021	1363	3035	813	3183
TRAILBLAZERS	4351	2758	2428	3316	3930	2388	4078	3869	1619	0	779	2765	3386	1020	3776
KINGS	4099	2850	2543	3032	4030	2150	3861	3947	1021	779	0	2343	3531	856	3817
SPURS	1846	1784	1737	830	2548	678	1676	2427	1363	2765	2343	0	2318	1749	2230
RAPTORS	1994	690	988	1778	563	1795	1708	542	3035	3386	3531	2318	0	2676	570
JAZZ	3358	1994	1690	2309	3173	1386	3097	3091	813	1020	856	1749	2676	0	2964
WIZARS	1489	1021	1352	1533	330	1850	1230	205	3183	3776	3817	2230	570	2964	0

Tabla A.2: Distancia entre los equipos de la NBA

Anexo B

Ejemplos

B.1. Resolución de un problema de PLE aplicando técnicas de ramificación y acotación

Sea el problema PLE:

$$\begin{cases} \max & 3x_1 + 2x_2 \\ \text{s.a.} & x_1 \leq 2 \quad x_2 \leq 2 \\ & x_1 + x_2 \leq 3.5 \\ & x_j \geq 0 \text{ y enteras } \forall j = 1,2 \end{cases}.$$

El problema relajado (PR1) correspondiente es:

$$\begin{cases} \max & 3x_1 + 2x_2 \\ \text{s.a.} & x_1 \leq 2 \quad x_2 \leq 2 \\ & x_1 + x_2 \leq 3.5 \\ & x_j \geq 0 \quad \forall j = 1,2 \end{cases}.$$

Se trata de un problema de dos dimensiones que se puede resolver de manera gráfica. Para resolver un problema de PLE de forma gráfica se debe encontrar la región factible, que en este caso podemos observar que esta acotada, por lo tanto la solución global se encontrará en uno de los puntos extremos.

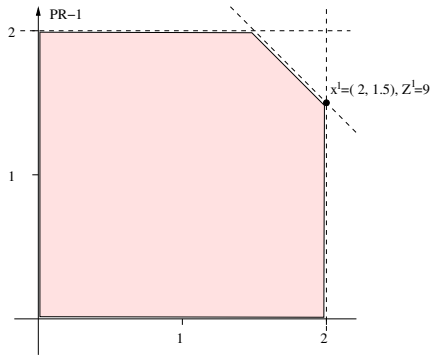

Figura B.1: Resolución gráfica del problema relajado (PR1).

Observamos que la solución obtenida es: $\mathbf{Z^1} = 9$ y $\mathbf{X^1} = (2, 1.5)$. Es claro que $x_2^1 \notin \mathbb{Z}$, por lo tanto no se ha obtenido una solución factible al resolver el problema relajado y se debe aplicar el algoritmo de ramificación y acotación. Inicializamos una cota inferior $Z^* = -\infty$.

Se procede a plantear dos nuevos subproblemas relajados (PR2) y (PR3) añadiendo la siguiente restricción:

$$\begin{cases} x_2 \leq 1 \text{ en (PR2)} \\ x_2 \geq 2 \text{ en (PR3)} \end{cases} .$$

Los subproblemas (PR2) y (PR3) son respectivamente:

$$\begin{cases} \max & 3x_1 + 2x_2 \\ \text{s.a.} & x_1 \leq 2 \quad x_2 \leq 2 \\ & \mathbf{x_2 \leq 1} \\ & x_1 + x_2 \leq 3.5 \\ & x_j \geq 0 \quad \forall j = 1,2 \end{cases} .$$

$$\begin{cases} \max & 3x_1 + 2x_2 \\ \text{s.a.} & x_1 \leq 2 \quad x_2 \leq 2 \\ & \mathbf{x_2 \geq 2} \\ & x_1 + x_2 \leq 3.5 \\ & x_j \geq 0 \quad \forall j = 1,2 \end{cases} .$$

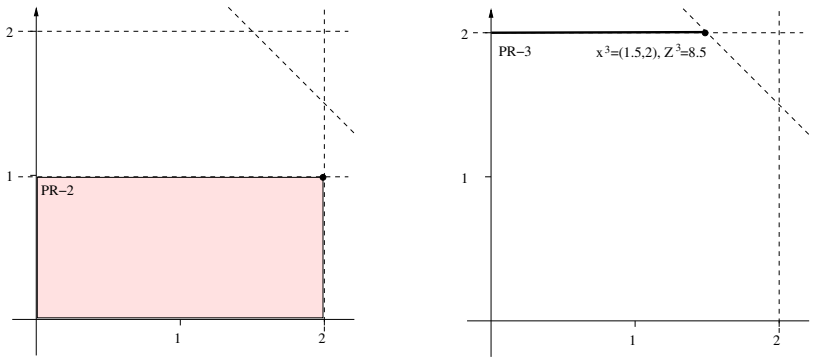

Figura B.2: Solución gráfica de los subproblemas relajados (PR2) y (PR3) respectivamente.

Observamos que las soluciones obtenidas son: $\mathbf{Z^2} = 8$, $\mathbf{X^2} = (2,1)$ y $\mathbf{Z^3} = 8.5$, $\mathbf{X^3} = (1.5,2)$. Al resolver (PR2) hemos obtenido una solución factible del subproblema de PLE y procedemos a actualizar la cota, así $\mathbf{Z}^* = 8$, y el problema se cierra. Al resolver (PR3) se puede observar que $x_1^3 \notin \mathbb{Z}$, por lo tanto no se ha obtenido una solución factible al resolver el subproblema relajado y se debe aplicar nuevamente el algoritmo de ramificación y acotación.

Se procede a plantear dos nuevos subproblemas relajados (PR4) y (PR5) añadiendo la siguiente restricción:

$$\begin{cases} x_1 \leq 1 \text{ en (PR4)} \\ x_1 \geq 2 \text{ en (PR5)} \end{cases} .$$

Los subproblemas (PR4) y (PR5) son respectivamente:

$$\begin{cases} \max & 3x_1 + 2x_2 \\ \text{s.a.} & x_1 \leq 2 \quad x_2 \leq 2 \\ & x_2 \geq 1 \quad \mathbf{x_1 \leq 1} \\ & x_1 + x_2 \leq 3.5 \\ & x_j \geq 0 \quad \forall j = 1,2 \end{cases} .$$

$$\begin{cases} \max & 3x_1 + 2x_2 \\ \text{s.a.} & x_1 \leq 2 \quad x_2 \leq 2 \\ & x_2 \geq 2 \quad \mathbf{x_1 \geq 2} \\ & x_1 + x_2 \leq 3.5 \\ & x_j \geq 0 \quad \forall j = 1,2 \end{cases} .$$

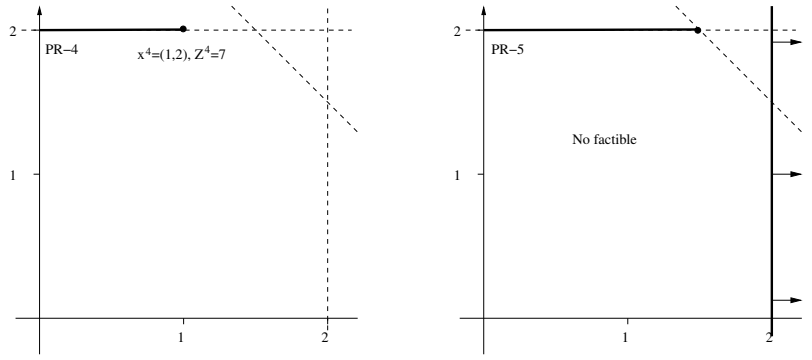

Figura B.3: Solución gráfica de los subproblemas relajados (PR4) y (PR5) respectivamente.

Las soluciones obtenidas son: $\mathbf{Z^4} = 7$, $\mathbf{X^2} = (1,2)$ y (PR5) resulta ser un subproblema no factible. En consecuencia se cierra el problema debido a la no factibilidad. Al resolver (PR4) hemos obtenido una solución factible del subproblema de PLE. En esta ocasión no se procede a actualizar la cota ya que la que se tiene hasta el momento es mejor que la obtenida y el problema se cierra por acotación.

Como se puede observar, el problema se ha resuelto aplicando el método gráfico de resolución. A continuación, se muestran algunas de las tablas de cada problema si se decide utilizar el algoritmo del *Simplex*.

$$\begin{cases} \max & 3x_1+2x_2 \\ \text{s.a.} & x_1 \leq 2 \\ & x_2 \leq 2 \\ & x_1+x_2 \leq 3.5 \\ & x_j \geq 0 \quad \forall j=1,2 \end{cases} \Rightarrow \begin{cases} \max & 3x_1+2x_2+0x_3+0x_4+0x_5 \\ \text{s.a.} & x_1+x_3=2 \\ & x_2+x_4=2 \\ & 2x_1+2x_2+x_5=7 \\ & x_j \geq 0 \quad \forall j=1,\cdots,5 \end{cases}$$

max	x_1	x_2	x_3	x_4	x_5	$\bar{\mathbf{b}}$
x_1	1	0	1	0	0	2
x_4	0	0	1	1	-1/2	1/2
x_2	0	1	-1	0	1/2	3/2
	0	0	-1	0	-1	

Tabla B.1: Tabla final del *Simplex* correspondiente al problema (PR1).

max	x_1	x_2	x_3	x_4	x_5	x_6	$\bar{\mathbf{b}}$
x_1	1	0	1	0	0	0	2
x_4	0	0	-1	1	0	-1	0
x_2	0	1	0	0	0	1	1
x_5	0	0	-2	0	1	-2	1
	0	0	-3	0	0	-2	

Tabla B.2: Tabla final del *Simplex* correspondiente al problema (PR2).

max	x_1	x_2	x_3	x_4	x_5	x_6	$\bar{\mathbf{b}}$
x_1	1	0	0	0	1/2	1	3/2
x_4	0	0	0	1	0	1	0
x_2	0	1	0	0	0	-1	1
x_3	0	0	1	0	-1/2	-1	1/2
	0	0	0	0	-3/2	-1	

Tabla B.3: Tabla final del *Simplex* correspondiente al problema (PR3).

max	x_1	x_2	x_3	x_4	x_5	x_6	x_7	$\bar{\mathbf{b}}$
x_1	1	0	0	0	1/2	1	0	3/2
x_4	0	0	0	1	0	1	0	0
x_2	0	1	0	0	0	-1	0	1
x_3	0	0	1	0	-1/2	-1	0	1/2
x_7	0	0	0	0	-1/2	-1	1	-1/2
	0	0	0	0	-3/2	-1	0	

Tabla B.4: Tabla inicial del *Simplex* correspondiente al problema (PR4).

max	x_1	x_2	x_3	x_4	x_5	x_6	x_7	$\bar{\mathbf{b}}$
x_1	1	0	0	0	1/2	1	0	3/2
x_4	0	0	0	1	0	1	0	0
x_2	0	1	0	0	0	-1	0	1
x_3	0	0	1	0	-1/2	-1	0	1/2
x_7	0	0	0	0	1/2	1	1	-1/2
	0	0	0	0	-3/2	-1	0	

Tabla B.5: Tabla inicial del *Simplex* del problema (PR5).

Al resolver (PR5) se obtiene que el problema no es factible.

La figura B.4 muestra un esquema de la resolución de un problema con técnicas de ramificación y acotación, donde se puede observar el VFO y el valor de las variables de decisión.

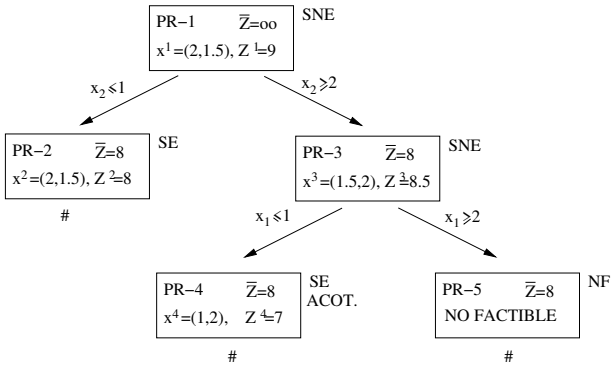

Figura B.4: Esquema de la resolución de un problema con técnicas de ramificación y acotación.

B.2. Aplicación del método *K-Means* en un ejemplo con 6 datos

Disponemos de un conjunto de datos de dimensión 6 y se quieren calcular 2 clusters. Los datos que se han elegido para realizar el ejemplo son 6 ciudades de la NBA, cuyos nombres y coordenadas son las siguientes:

Equipo	Coordenada X	Coordenada Y
Hawks	-84.4227	33.7629
Celtics	-71.0202	42.332
Nets	-73.94958	40.6501
Hornets	-80.84313	35.22709
Bulls	-87.6828	41.8379
Cavaliers	-81.69541	41.4995

Tabla B.6: Conjunto de datos con sus respectivas coordenadas

Inicializamos los centroides de manera completamente aleatoria, escogiendo las siguientes coordendas:

Centroide	Coordenada X	Coordenada Y
C1	-85	35
C2	-80	40

Tabla B.7: Coordenadas de los centroides iniciales

Se procede a calcular qué datos pertenecen a cada centroide. En este caso no se van a calcular distancias euclídeas como se ha explicado en el Capítulo 3 porque estamos con coordenadas en el espacio. Se utiliza la fórmula A.1.

Equipo	C1	C2
Hawks	147.41	797.05
Celtics	1459.15	794.67
Nets	1154.76	517.88
Hornets	378.92	535.89
Bulls	795.33	676.82
Cavaliers	778.02	219.53

Tabla B.8: Distancia entre cada dato inicial y los dos centroides iniciales

Los clusters que hemos obtenido son {Hawks, Hornets} y {Celtics , Nets , Bulls, Cavaliers}; con centroides C1 y C2 respectivamente. A continuación se procede a actualizar los centroides:

$$C1 = \left(\frac{-84.4227 - 80.84313}{2}, \frac{33.7629 + 35.22709}{2}\right) = (-82.6329, 34.4950).$$

$$C2 = \left(\frac{-71.0202 - 73.94958 - 87.6828 - 81.69541}{4}, \frac{42.3320 + 40.6501 + 41.8379 + 41.4995}{4}\right)$$
$$= (-78.5870, 41.5799).$$

Centroide	Coordenada X	Coordenada Y
C1	- 82.6329	34.4950
C2	-78.5870	41.5799

Tabla B.9: Coordenadas de los centroides actualizados tras la primera iteración

Comienza la segunda iteración del algoritmo

Equipo	C1	C2
Hawks	183.75	1009.01
Celtics	1333,08	631.05
Nets	1025.71	401.96
Hornets	182.46	733.17
Bulls	927.77	755.24
Cavaliers	783.17	258.85

Tabla B.10: Distancia de los datos iniciales a los centroides actualizados

Los clusters que hemos obtenido son {Hawks, Hornets} y {Celtics , Nets , Bulls, Cavaliers}, con centroides C1 y C2 respectivamente. Observar, que los clusters obtenidos son los mismos que teníamos anteriormente, por lo tanto se puede decir que el algoritmo ha convergido.

Anexo C

Códigos

C.1. Kmvar

A continuación se muestra la función *Kmvar* que implementa el método *K-Means Constrained*.

```
kmvar <- function(mat, clsize=10, method=c('random','maxd', 'mind', 'elki')){
  k = ceiling(nrow(mat)/clsize)
  km.o = kmeans(mat, k)
  labs = rep(NA, nrow(mat))
  centd = lapply(1:k, function(kk){
    euc = t(mat)-km.o$centers[kk,]
    sqrt(apply(euc, 2, function(x) sum(x^2)))
  })
  centd = matrix(unlist(centd), ncol=k)
  clsizes = rep(0, k)
  if(method[1]=='random'){
    ptord = sample.int(nrow(mat))
  } else if(method[1]=='elki'){
    ptord = order(apply(centd, 1, min) - apply(centd, 1, max))
  } else if(method[1]=='maxd'){
    ptord = order(-apply(centd, 1, max))
  } else if(method[1]=='mind'){
    ptord = order(apply(centd, 1, min))
  } else {
    stop('unknown method')
  }
  for(ii in ptord){
    bestcl = which.max(centd[ii,])
    labs[ii] = bestcl
    clsizes[bestcl] = clsizes[bestcl] + 1
    if(clsizes[bestcl] >= clsize){
      centd[,bestcl] = NA
    }
  }
  return(labs)
}
```

C.2. Generación de trips mediante clusters

A continuación se muestra el código que he creado para la generación de trips utilizando la técnica cluster: *K-Means Constrained*.

```
library(readxl)
library(ggplot2)
library(Rcmdr)

#Definimos un vector que contiene el nombre de los 30 equipos
Equipo<- c("Hawks", "Celtics", "Nets", "Hornets", "Bulls", "Cavaliers", "
    Mavericks", "Nuggets", "Pistons", "Warriors", "Rockets", "Pacers", "Clippers
    ", "Lakers", "Grizzlies", "Heat", "Bucks", "Timberwolves", "Pelicans", "
    Knicks", "Thunder", "Magic", "76ers", "Suns", "TrailBlazers", "Kings", "
    Spurs", "Raptors", "Jazz", "Wizards")

#Leemos de un archivo excel la latitud y longuitud de las ciudades de los 30
    equipos
excel<- read_excel("C:/Users/PORTATIL/Desktop/TFG/DistanciaEntreCiudades.xlsx",
                   sheet = "Hoja1", range = "A3:B33")

#Definimos dos vectores con las coordenadas de latitud y longitud de las
    ciudades de los diferentes equipos
lon<-c(excel$Longitud)
lat<-c(excel$Latitud)

#Se define un data frame que contiene el nombre cada equipo con su
    correspondiente latitud y longitud
coordenadas<-data.frame(Equipo,lon,lat)

#Ubicación de los equipos de la NBA
ggplot() + geom_point(aes(x = lon, y = lat), data = coordenadas, alpha = 0.5) +
  geom_text(aes(x = lon, y = lat),label = coordenadas$Equipo, data =
    coordenadas) + ggtitle('Equipos de la NBA')

#Leemos la distancia entre las diferentes ciudades de la tabla de excel
distancia <- readXL("C:/Users/PORTATIL/Desktop/TFG/DistanciaEntreCiudades.xlsx"
    ,
                    rownames=TRUE, header=TRUE, na="", sheet="Hoja2",
    stringsAsFactors=TRUE)
rownames(distancia)<- Equipo
colnames(distancia)<- Equipo

#convertimos a matriz el dataframe que contiene la distancia entre todas las
    ciudades, esto lo hacemos para poder calcular los clusters
my_dist <- as.matrix(distancia)

#variante de KMeans que crea cluster de tamaño fijo dada la distancia existente
     entre los puntos
...
FUNCIÓN KMVAR
...

#Hallamos clusters de tamaño 3
cluster3<- kmvar(distancia,3,method = 'maxd')
coordenadas$C3 <- cluster3

ggplot() + geom_point(aes(x = lon, y = lat, color = cluster3), data =
    coordenadas, alpha = 0.5, size = 8) +
  geom_text(aes(x = lon, y = lat),label = coordenadas$Equipo, data =
    coordenadas)+ scale_colour_gradientn(colours=rainbow(10)) + ggtitle('Cluster
     de tamaño 3')

#creamos una matriz que contenga los clusters de tamaño 3
```

```
tam3 <- matrix(NA, 3, 10)
for(i in 1:10) {
  for (j in 1:3) {
    tam3[j,i] <- subset(coordenadas, subset=C3 == i, select=c(Equipo))[j,1]
  }}
colnames(tam3)<- c(1:10)

#Distancia minima de los clusters de tamaño 3
mindist3 <- matrix(NA,10,3)
for(i in 1:10){
  mindist3[i,1] <- min(distancia[tam3[1,i],tam3[2,i]] + distancia[tam3[2,i],
    tam3[3,i]], distancia[tam3[1,i],tam3[3,i]] + distancia[tam3[3,i], tam3[2,i
    ]])
  mindist3[i,2] <- min(distancia[tam3[2,i],tam3[1,i]] + distancia[tam3[1,i],
    tam3[3,i]], distancia[tam3[2,i],tam3[3,i]] + distancia[tam3[3,i], tam3[1,i
    ]])
  mindist3[i,3] <- min(distancia[tam3[3,i],tam3[1,i]] + distancia[tam3[1,i],
    tam3[2,i]], distancia[tam3[3,i],tam3[2,i]] + distancia[tam3[2,i], tam3[1,i
    ]])
}
rownames(mindist3)<- c(1:10)

#Orden de los clusters de tamaño 3 empezando en la ciudad 1
Orden1 <- matrix(NA,3,10)
for (i in 1:10){
  if(distancia[tam3[1,i],tam3[2,i]] + distancia[tam3[2,i], tam3[3,i]] ==
    mindist3[i,1]){
    Orden1[1,i] <- tam3[1,i]
    Orden1[2,i] <- tam3[2,i]
    Orden1[3,i] <- tam3[3,i]
  }
  else {Orden1[1,i] <- tam3[1,i]
  Orden1[2,i] <- tam3[3,i]
  Orden1[3,i] <- tam3[2,i]
  }}
colnames(Orden1)<- c(1:10)

#Orden de los clusters de tamaño 3 empezando en la ciudad 2
Orden2 <- matrix(NA,3,10)
for (i in 1:10){
  if(distancia[tam3[2,i],tam3[1,i]] + distancia[tam3[1,i], tam3[3,i]] ==
    mindist3[i,2]){
    Orden2[1,i] <- tam3[2,i]
    Orden2[2,i] <- tam3[1,i]
    Orden2[3,i] <- tam3[3,i]}
  else {Orden2[1,i] <- tam3[2,i]
  Orden2[2,i] <- tam3[3,i]
  Orden2[3,i] <- tam3[1,i]
  }}
colnames(Orden2)<- c(1:10)

#Orden de los clusters de tamaño 3 empezando en la ciudad 3
Orden3 <- matrix(NA,3,10)
for (i in 1:10){
  if(distancia[tam3[3,i],tam3[1,i]] + distancia[tam3[1,i], tam3[2,i]] ==
    mindist3[i,3]){
    Orden3[1,i] <- tam3[3,i]
    Orden3[2,i] <- tam3[1,i]
    Orden3[3,i] <- tam3[2,i]}
  else {Orden3[1,i] <- tam3[3,i]
  Orden3[2,i] <- tam3[2,i]
  Orden3[3,i] <- tam3[1,i]
  }}
```

```
colnames(Orden3)<- c(1:10)

#Ahora tenemos que mirar para todo equipo cuando vaya a un cluster, que ciudad visitara primera, es decir que ciudad esta a menor distancia
#Cluster de tamaño 3
visita3<- matrix(NA,10,30)
for (i in 1:30){
  for (j in 1:10){
    visita3[j,i]<- min(distancia[i,Orden1[1,j]]+ mindist3[j,1]+ distancia[Orden1[3,j],i],
                       distancia[i,Orden2[1,j]]+ mindist3[j,2]+ distancia[Orden2[3,j],i],
                       distancia[i,Orden3[1,j]]+ mindist3[j,3]+ distancia[Orden3[3,j],i])
  }}
colnames(visita3)<-Equipo
rownames(visita3)<-c(1:10)

#En cada columna tenemos el nombre de los 30 equipo y en cada fila tenemos a la ciudad
#que tiene que ir en primer lugar si decide "escoger" el cluster
minimo3<- matrix(NA,10,30)
for (i in 1:30) {
  for (j in 1:10) {
    if (distancia[i,Orden1[1,j]]+ mindist3[j,1]+ distancia[Orden1[3,j],i] == visita3[j,i]){
      minimo3[j,i] <- Orden1[1,j]
    }
    else if (distancia[i,Orden2[1,j]]+ mindist3[j,2]+ distancia[Orden2[3,j],i] == visita3[j,i]){
      minimo3[j,i] <- Orden2[1,j]
    }
    else (minimo3[j,i]<- Orden3[1,j])
  }
}
colnames(minimo3)<-Equipo

#Creamos todos los vectores posibles de la forma <EquipoLocal(i), nºTrip, EquipoVisitante(j), posicion(j)DentroTrip>
trips3<- matrix(NA,300,6)
for (i in 1:30){
  for (k in 1:10){
    if (minimo3[k,i] == Orden1[1,k]){
      if(coordenadas$Equipo[i] == Orden1[1,k]){
        trips3[10*(i-1)+k,1]<-coordenadas$Equipo[i]
        trips3[10*(i-1)+k,2]<-colnames((Orden1))[k]
        trips3[10*(i-1)+k,3]<-Orden1[2,k]
        trips3[10*(i-1)+k,4]<- Orden1[3,k]
        trips3[10*(i-1)+k,6]<- visita3[k,i]
      }
      else {
        trips3[10*(i-1)+k,1]<-coordenadas$Equipo[i]
        trips3[10*(i-1)+k,2]<-colnames((Orden1))[k]
        trips3[10*(i-1)+k,3]<-Orden1[1,k]
        trips3[10*(i-1)+k,4]<- Orden1[2,k]
        trips3[10*(i-1)+k,5]<- Orden1[3,k]
        trips3[10*(i-1)+k,6]<- visita3[k,i]
      }
    }
    else if (minimo3[k,i] == Orden2[1,k]){
      if(coordenadas$Equipo[i] == Orden2[1,k]){
        trips3[10*(i-1)+k,1]<-coordenadas$Equipo[i]
        trips3[10*(i-1)+k,2]<-colnames((Orden2))[k]
```

```
154         trips3[10*(i-1)+k,3]<-Orden2[2,k]
155         trips3[10*(i-1)+k,4]<- Orden2[3,k]
156         trips3[10*(i-1)+k,6]<- visita3[k,i]
157       }
158       else{
159         trips3[10*(i-1)+k,1]<-coordenadas$Equipo[i]
160         trips3[10*(i-1)+k,2]<-colnames((Orden2))[k]
161         trips3[10*(i-1)+k,3]<-Orden2[1,k]
162         trips3[10*(i-1)+k,4]<- Orden2[2,k]
163         trips3[10*(i-1)+k,5]<- Orden2[3,k]
164         trips3[10*(i-1)+k,6]<- visita3[k,i]
165       }
166     }
167     else {
168       if(coordenadas$Equipo[i] == Orden3[1,k]){
169         trips3[10*(i-1)+k,1]<-coordenadas$Equipo[i]
170         trips3[10*(i-1)+k,2]<-colnames((Orden3))[k]
171         trips3[10*(i-1)+k,3]<-Orden3[2,k]
172         trips3[10*(i-1)+k,4]<- Orden3[3,k]
173         trips3[10*(i-1)+k,6]<- visita3[k,i]
174       }
175       else{
176         trips3[10*(i-1)+k,1]<-coordenadas$Equipo[i]
177         trips3[10*(i-1)+k,2]<-colnames((Orden3))[k]
178         trips3[10*(i-1)+k,3]<-Orden3[1,k]
179         trips3[10*(i-1)+k,4]<- Orden3[2,k]
180         trips3[10*(i-1)+k,5]<- Orden3[3,k]
181         trips3[10*(i-1)+k,6]<- visita3[k,i]
182 
183       }
184     }
185   }}
186 
187 
188 tripsfile<- file("Trips3.txt", open = 'at')
189 for (i in 1:300){
190   writeLines(sprintf("< ''%s'', %s>: < %s,{''%s'', ''%s'',''%s''}, %s >,",
      trips3[i,1], trips3[i,2], 3 - sum(is.na(trips3[i,3:5])), trips3[i,3], trips3
      [i,4], trips3[i,5], trips3[i,6] ), tripsfile)
191 }
192 close(tripsfile)
```

El caso de tamaño 2 y tamaño 1 no se muestra ya que es simplemente una suma de distancias.

C.3. Generación de trips mediante algoritmos genéticos

A continuación se muestra el código que he utilizado para generar *trips*, empleando algoritmos genéticos.

C.3.1. Evaluación de trips

Función que proporciona el fitness de cada individuo.

```
evaluaTrips <- function(trip, patern,city)
{
  noCities <- length(trip)-1
  fit=0

  for(j in 1:(length(patern)-1)){
    rg=trip[patern[j]:(patern[j+1]-1)]
    fit=fit+D[city,rg[1]]+D[rg[length(rg)],city]
    for(i in 1:(length(rg)-1)){
      fit=fit+D[rg[i],rg[i+1]]
    }
  }
  return(fit)
}
```

C.3.2. Recombinación

Función que realiza la recombinación de los individuos de la población, conservando la estructura de permutación y el patrón definido.

```
recombina <- function(individuos){

  nReinas <- length(individuos[1,])

  hijos <- matrix(0,nrow=2,ncol=nReinas)
  pos <- sample(1:(nReinas-1),1)
  hijos[,1:pos]  <- individuos[,1:pos]

  for(j in 1:2){
    k <- 1
    for(i in 1:nReinas){
      posicion <- 1+((pos+i-1)%%nReinas)
      saltar <- TRUE
      for(h in 1:pos){
        if(individuos[-j+3,posicion]==hijos[j,h]){
          saltar=FALSE
        }
      }
      if(saltar){
        hijos[j,pos+k] <- individuos[-j+3,posicion]
        k <- k +1
      }
    }
  }

  return(hijos)
}
```

C.3.3. Mutación

Función que realiza la mutación de los hijos, conservando la estructura de permutación y el patrón definido.

```
mutacion <- function(hijos,pm){
  d<-dim(hijos)
  numHijos <- d[1]
  nReinas <- d[2]
  for(i in 1:numHijos){
    if(runif(1)<pm){
      pos <- sample(1:nReinas,2)
      k1 <- hijos[i,pos[1]]
      hijos[i,pos[1]] <- hijos[i,pos[2]]
      hijos[i,pos[2]] <- k1
    }
  }
  return(hijos)
```

C.3.4. Selección de supervivientes

Función que elimina los dos peores padres de la población y los reemplaza por los dos hijos generados.

```
supervivientes <- function(poblacion,fitness,hijos,fitnessHijos)
{
  orden <- order(fitness[,1])
  poblacion[1:(tamagnoPoblacion-2),] <- poblacion[orden[1:(tamagnoPoblacion-2)
    ],]
  poblacion[(tamagnoPoblacion-1):tamagnoPoblacion,] <- hijos[1:2,]

  fitness[1:(tamagnoPoblacion-2),] <- fitness[orden[1:(tamagnoPoblacion-2)],]
  fitness[(tamagnoPoblacion-1):tamagnoPoblacion,] <- fitnessHijos[1:2,]


  return(list("pob"=poblacion,"fit"=fitness))

}
```

C.3.5. Programa principal - BuildTrips

```
source("recombina.R")
source("mutacion.R")
source("evaluaTrips.R")
source("supervivientes.R")
source("pintaTrips.R")

load("DistanciaEntreCiudades.RData")
D = data.matrix(DistanciaEntreCiudades)
load("coordenadas.RData")

tamagnoPoblacion <- 100
noTeams <- 30
Equipo<- c("Hawks", "Celtics", "Nets", "Hornets", "Bulls", "Cavaliers", "
    Mavericks", "Nuggets", "Pistons", "Warriors", "Rockets", "Pacers", "Clippers
    ", "Lakers", "Grizzlies", "Heat", "Bucks", "Timberwolves", "Pelicans", "
    Knicks", "Thunder", "Magic", "76ers", "Suns", "TrailBlazers", "Kings", "
    Spurs", "Raptors", "Jazz", "Wizards")
names(DistanciaEntreCiudades) <- Equipo
rownames(DistanciaEntreCiudades) <- Equipo
colnames(D)<- Equipo
rownames(D)<- Equipo

numeroIteraciones <- 30000
patron=cumsum(c(1,3,3,3,3,3,3,3,3,3,2)) #para trips de tamaño 3
#patron=cumsum(c(1,2,2,2,2,2,2,2,2,2,2,2,2,2,2)) #para trips de tamaño 2
```

```
22
23
24 for (city in 1:30){
25 poblacion <- matrix(0,nrow=tamagnoPoblacion,ncol=noTeams-1)
26 fitness <- matrix(0,nrow=tamagnoPoblacion,ncol=1)
27 hijos <- matrix(0,2,noTeams)
28 fitnessHijos <- matrix(0,nrow=2,ncol=1)
29
30 #Poblacion inicial
31 for(i in 1:tamagnoPoblacion){
32   poblacion[i,] <- setdiff(sample(1:noTeams,noTeams),city)
33   fitness[i] <- evaluaTrips(poblacion[i,],patron,city)
34 }
35
36 #bucle principal
37 for(jj in 1:numeroIteraciones){
38   #Seleccion de padres
39   quienes=sample(1:tamagnoPoblacion,5);
40   candidatos <- quienes[order(fitness[quienes])[1:2]]
41   #Crossover
42   hijos <- recombina(poblacion[candidatos,])
43   #mutacion
44   hijos <- mutacion(hijos,0.8)
45   #Evaluacion de hijos
46   for(i in 1:2)    fitnessHijos[i] <- evaluaTrips(hijos[i,],patron,city)
47   #Seleccion supervivientes
48   res <- supervivientes(poblacion,fitness,hijos,fitnessHijos)
49   poblacion <- res$pob
50   fitness <- res$fit
51   #imprime info
52   print(sprintf("%2.0f) min= %3.0f max =%3.0f, media=%3.1f",jj,min(fitness),max
       (fitness),mean(fitness)))
53   if(min(fitness)==0) break
54 }
55
56
57 #Resultado final
58 i <- which.min(fitness)
59
60 for(j in 1:(length(patron)-1)){
61   rg=poblacion[i,patron[j]:(patron[j+1]-1)]
62   fit=D[city,rg[1]]+D[rg[length(rg)],city]
63   for(aa in 1:(length(rg)-1)){
64     fit=fit+D[rg[aa],rg[aa+1]]
65   }
66   print(rg)
67   numero<- rep(c(1:10),30)
68   tripsgeneticos<- file("GeneticosTrips_Tamaño3.txt", open = 'at')
69   writeLines(sprintf("< ''%s'', %s>: < %s,{''%s'', ''%s'', ''%s''}, %s >,",
       Equipo[city], numero[j], length(rg) , Equipo[rg[1]], Equipo[rg[2]], Equipo[
       rg[3]], fit), tripsgeneticos)
70   close(tripsgeneticos)
71 }
```

C.4. Programa principal - CPLEX

C.4.1. Problema 1

Problema1.mod

```
int D = ...; //Número de rondas disponibles para jugar partidos

int T = ...; //Ronda en el que se tiene jugar después de un Trip un partido en
    casa
int H = ...; //En las primeras H rondas de juego se debe jugar un partido

range K = 1..D;

{string} equipos = ...; //Nombre de todos los posibles equipos
int NumeroTrips[equipos]=...; //Número de posibles trips de cada equipo

//Tupla que contiene el nombre de un equipo y el número de trip
tuple IndexEquipoTrip{
  string equipo;
  int num;
}

/*Generamos el posible conjunto de indices para la variable Z_{sk} donde
s indica el trip entre todos los posibles, que se escoge del conjunto de
    indices */
{IndexEquipoTrip} IndexTrip = {<i,n> | i in equipos , n in 1..NumeroTrips[i]};

//Tupla que contiene los elementos que hay en los trips; excluyendo el índice
tuple Trip{
  int longitud;
  {string} visitantes;
  float coste;
  }

//Lectura de todos los posibles trips
Trip S[IndexTrip] = ...;

//Tupla que contiene dos equipos
tuple Cruces{
string i;
string j;
};

//Cruces hace referencia al conjunto de todos los enfrentamientos posibles que
    debe haber
{Cruces} cruces={ <i,j> | i in equipos ,j in equipos: i!=j};

tuple indexz{
   string equipo;
   int numTrip;
   int ti;
};

//Restriccion necesaria para que la variable Z_sk este bien definida
{indexz} IndexZ = {<i,n,k> | <i,n> in IndexTrip, k in K:  k<= D-S[<i,n>].
    longitud +1 };

int posicion[IndexTrip][equipos];
int estaEnTrip[IndexTrip][equipos];
execute{
var cont=0;
for(var i in IndexTrip){
```

```
for(var j in equipos)
estaEnTrip[i][j]=0;
}
for(var i in IndexTrip){
  for(var vis in S[i].visitantes){
   posicion[i][vis]=cont;
   estaEnTrip[i][vis]=1;
   cont=cont+1;
   }
   cont=0;
}
};

//Variables de decisión
dvar boolean X[cruces][K]; //Corresponde a la variable X_{ijk}
dvar boolean Z[IndexZ] ; //Corresponde a la variable Z_{sk}

//Función Objetivo
dexpr float CosteTotal =  sum (s in IndexZ) Z[s]*S[<s.equipo, s.numTrip>].coste
    ;
minimize
 CosteTotal;

//Restricciones
subject to{
  //1. Para que se jueguen todos los partidos
  forall (c in cruces)
    RestriccionJugar:
    sum (k in K)(X[c][k]) == 1;

  //2. Para que en un día cada equipo juegue a lo sumo un partido
  forall (i in equipos, k in K)
    RestriccionCapacidad:
    sum (j in equipos: j !=i)(X[<i,j>][k] + X[<j,i>][k]) <=1;

  //3. Para que los partidos se juegen entre los posibles trips
  forall (c in cruces, k in K)
    RestriccionTrip:
    X[c][k] == sum(s in IndexZ: s.equipo==c.j && estaEnTrip[<s.equipo,s.numTrip
    >][c.i]!=0 && s.ti==k-posicion[<s.equipo,s.numTrip>][c.i]) Z[s];

  //4. Para que se juege un partido en casa en los T días siguientes al
    finalizar un trip
  forall (s in IndexZ: s.ti+S[<s.equipo,s.numTrip>].longitud-1 < D)
      RestriccionDespuesTrip:
      sum (c in cruces: c.i ==s.equipo)
        sum(t in 1..T: s.ti+S[<s.equipo,s.numTrip>].longitud+t -1<=D)
        (X[c][s.ti+S[<s.equipo,s.numTrip>].longitud+t-1])>=Z[s];

  //5. Asignar descansos después de un trip
  forall (s in IndexZ: 1<s.ti<= D -S[<s.equipo,s.numTrip>].longitud)
    RestriccionDescanso:
    sum(c in cruces: c.i ==s.equipo)
      (X[c][s.ti+S[<s.equipo,s.numTrip>].longitud] + X[<c.j,c.i>][s.ti+S[<s.
    equipo,s.numTrip>].longitud])<= 2*(1-Z[s]);

  //6. Para que cada equipo juegue al menos un partido en las H primeras rondas
     posibles de juego
  forall (i in equipos)
    RestriccionEmpezar:
    sum(j in equipos: j!=i)sum (k in 1..H)(X[<i,j>][k] + X[<j,i>][k])>=1;
};

```

```
/*Se crea un archivo .txt en el que se guarda el orden en el que cada
equipo tiene que jugar sus partidos.Esta información se utilizará para resolver
el segundo problema.*/
execute{
  var ofile = new IloOplOutputFile("Partidos_12Equipos.txt");
    for (var i in equipos){
      ofile.write('"',  i , '"', ':', '{');
    for(var k in K){
      for (var c in cruces){
        if (X[c][k] !=0 && i == c.i | i ==c.j){
          ofile.write( c,',');
          }}}
          ofile.writeln('},')}
  ofile.close();
};

/*Se crea un archivo .txt en el que guardar los trips que han sido elegidos.
    Esta información se utilizará para resolver el segundo problema.*/
execute{
  var ofile = new IloOplOutputFile("TripsUtilizados_12Equipos.txt");
  for(var i in IndexZ){
    if(Z[i]==1){
        ofile.writeln('<',i.equipo,",",i.numTrip , '>:', '<' , S[IndexTrip.get(
    i.equipo,i.numTrip)].longitud,",",
        S[IndexTrip.get(i.equipo,i.numTrip)].visitantes , '>,');
     }
}
ofile.close();
};

//Ver los kilómetros recorridos por cada equipo
execute{
  var ofile = new IloOplOutputFile("KilometrosRecorridos_12Equipos.txt");
  for(var j in equipos){
  km = 0
  ofile.write('"',  j , '"', ':');
  for(var i in IndexZ){
    if(Z[i]==1 && i.equipo == j){
        km = km + S[IndexTrip.get(i.equipo,i.numTrip)].coste
      }}
  ofile.writeln(km)};
ofile.close();
};


//Para ver por pantalla rapidamente el resultado de la variable X
execute{
  for(var c in cruces){
    for (var k in K){
      if (X[c][k] !=0){
        writeln (c,k);
        }}}};
```

Problema1.dat

```
equipos = {"Hawks", "Celtics", "Nets", "Hornets", "Bulls", "Cavaliers", "
    Mavericks", "Nuggets", "Pistons", "Warriors", "Rockets", "Pacers"};

D = 50; //Rondas disponibles para jugar los partidos
T = 3; //Ronda en la que se tiene que jugar un partido en casa después de un
    trip
H = 3; //En las primeras H rondas de juego debe jugar un partido cada equipo
```

```
6
7  NumeroTrips = [20,20,20,20,20,20,
8  20,20,20,20,20,20];
9
10 S =
11 #[
12 < "Hawks", 1>: < 3,{"Hornets", "Celtics", "Nets"}, 3034.92545588967 >,
13 < "Hawks", 2>: < 3,{"Cavaliers", "Pistons", "Pacers"}, 2118.12023507257 >,
14 < "Hawks", 3>: < 3,{"Bulls", "Nuggets", "Warriors"}, 7480.64832883331 >,
15 < "Hawks", 4>: < 2,{"Rockets", "Mavericks"}, 2637.78917555498 >,
16 < "Celtics", 1>: < 3,{"Hawks", "Rockets", "Mavericks"}, 5486.26477208506 >,
17 < "Celtics", 2>: < 3,{"Bulls", "Nuggets", "Warriors"}, 8813.76294140505 >,
18 < "Celtics", 3>: < 3,{"Cavaliers", "Pacers", "Pistons"}, 2689.74333479534 >,
19 < "Celtics", 4>: < 2,{"Nets", "Hornets"}, 2320.76906596672 >,
20  .
21  .
22  .
23 < "Pacers", 10>: < 1,{"Hawks"}, 1370.78692645367 >,
24 < "Pacers", 11>: < 1,{"Celtics"}, 2598.22694728358 >,
25 < "Pacers", 12>: < 1,{"Nets"}, 2081.00800761898 >,
26 < "Pacers", 13>: < 1,{"Hornets"}, 1377.62637089136 >,
27 < "Pacers", 14>: < 1,{"Bulls"}, 526.95344863904 >,
28 < "Pacers", 15>: < 1,{"Cavaliers"}, 845.688475527946 >,
29 < "Pacers", 16>: < 1,{"Mavericks"}, 2450.80642302308 >,
30 < "Pacers", 17>: < 1,{"Nuggets"}, 3194.73972815034 >,
31 < "Pacers", 18>: < 1,{"Pistons"}, 774.841019253492 >,
32 < "Pacers", 19>: < 1,{"Warriors"}, 6366.5064262247 >,
33 < "Pacers", 20>: < 1,{"Rockets"}, 2784.92966834791 >
34 ]#;
```

C.4.2. Subproblema1-1

Subproblema1-1.mod

```
1  int D = ...; //Número de rondas disponibles para jugar partidos
2
3  int T = ...; //Ronda en el que se tiene jugar después de un Trip un partido en
       casa
4  int H = ...; //En las primeras H rondas de juego se debe jugar un partido
5
6  int P = ...; //Número minimo de partidos que se tienen que programar
7
8  int U = ...; //Número minimo de partidos que tiene que jugar cada equipo
9  int L = ...; //Número máximo de partidos que tiene que jugar cada equipo
10
11 int Uloc = ...; //Número minimo de partidos que tiene que jugar cada equipo
       como local
12 int Lloc = ...; //Número máximo de partidos que tiene que jugar cada equipo
       como local
13
14 int Uvis = ...; //Número minimo de partidos que tiene que jugar cada equipo
       como visitante
15 int Lvis = ...; //Número mñaximo de partidos que tiene que jugar cada equipo
       como visitante
16
17 range K = 1..D;
18
19 {string} equipos = ...; //Nombre de todos los posibles equipos
20 int NumeroTrips[equipos]=...; //Número de posibles trips de cada equipo
21
22 //Tupla que contiene el nombre de un equipo y el número de trip
23 tuple IndexEquipoTrip{
```

```
  string equipo;
  int num;
}

/*Generamos el posible conjunto de indices para la variable Z_{sk} donde
s indica el trip entre todos los posibles, que se escoge del conjunto de
    indices */
{IndexEquipoTrip} IndexTrip = {<i,n> | i in equipos , n in 1..NumeroTrips[i]};

//Tupla que contiene los elementos que hay en los trips; excluyendo el índice
tuple Trip{
  int longitud;
  {string} visitantes;
  float coste;
  }

// Lectura de todos los posibles trips
Trip S[IndexTrip] = ...;

//Tupla que contiene dos equipos
tuple Cruces{
string i;
string j;
};

//Cruces hace referencia al conjunto de todos los enfrentamientos posibles que
    debe haber
{Cruces} cruces={ <i,j> | i in equipos ,j in equipos: i!=j};

tuple indexz{
   string equipo;
   int numTrip;
   int ti;
};

//Restriccion necesaria para que la variable Z_sk este bien definida
{indexz} IndexZ = {<i,n,k> | <i,n> in IndexTrip, k in K:  k<= D-S[<i,n>].
    longitud +1 };


int posicion[IndexTrip][equipos];
int estaEnTrip[IndexTrip][equipos];
execute{
var cont=0;
for(var i in IndexTrip){
for(var j in equipos)
estaEnTrip[i][j]=0;
}
for(var i in IndexTrip){
  for(var vis in S[i].visitantes){
   posicion[i][vis]=cont;
   estaEnTrip[i][vis]=1;
   cont=cont+1;
   }
   cont=0;
}};

//Variables de decisión
dvar boolean X[cruces][K]; //Corresponde a la variable X_{ijk}
dvar boolean Z[IndexZ] ; //Corresponde a la variable Z_{sk}

//Función Objetivo
dexpr float CosteTotal =  sum (s in IndexZ) Z[s]*S[<s.equipo, s.numTrip>].coste
```

```
    ;
minimize
 CosteTotal;

//Restricciones
subject to{
  //1. No es necesario que se juegen todos los partidos
  forall (c in cruces)
    RestriccionJugar:
    sum (k in K)(X[c][k]) <= 1;

  //1.1 Como es un subproblema, se deben jugar al menos (globalmente) P
    partidos en las D rondas disponibles
  partidos_minimos:sum(c in cruces, k in K)(X[c][k])>= P ;

  //1.2 Para que cada equipo juege un número de partidos, comprendidos en el
    intervalo [U,L]
  forall (i in equipos)
    min_max_partidos:
    U <= sum(k in K)(sum (c in cruces: c.i == i)X[c][k] + sum(c in cruces: c.j
    ==i)X[c][k]) <= L;

  //1.3 Para que cada equipo juegue un número de partidos como local,
    comprendidos en el intervalo [Umin , Lmin]
  forall (i in equipos)
    min_local_partidos:
    Uloc <= sum(k in K)(sum(c in cruces:c.i ==i)X[c][k]) <= Lloc;

  //1.4 Para que cada equipo juege un número de partidos como visitante,
    comprendidos en el intervalo [Umax, Lmax]
  forall (i in equipos)
    min_visitante_partidos:
    Uvis <= sum(k in K)(sum(c in cruces:c.j ==i)X[c][k]) <= Lvis;

  //2. Para que en un día cada equipo juegue a lo sumo un partido
  forall (i in equipos, k in K)
    RestriccionCapacidad:
    sum (j in equipos: j !=i)(X[<i,j>][k] + X[<j,i>][k]) <=1;

  //3. Para que los partidos se juegen entre los posibles trips
  forall (c in cruces, k in K)
    RestriccionTrip:
    X[c][k] == sum(s in IndexZ: s.equipo==c.j && estaEnTrip[<s.equipo,s.numTrip
    >][c.i]!=0 && s.ti==k-posicion[<s.equipo,s.numTrip>][c.i]) Z[s];

  //4. Para que se juege un partido en casa en los T días siguientes al
    finalizar un trip
  forall (s in IndexZ: s.ti+S[<s.equipo,s.numTrip>].longitud-1 < D)
      RestriccionDespuesTrip:
      sum (c in cruces: c.i ==s.equipo)
        sum(t in 1..T: s.ti+S[<s.equipo,s.numTrip>].longitud+t -1<=D)
        (X[c][s.ti+S[<s.equipo,s.numTrip>].longitud+t-1])>=Z[s];

  //5. Asignar un día de descanso después de finalizar un trip
  forall (s in IndexZ: 1<s.ti<= D -S[<s.equipo,s.numTrip>].longitud)
    RestriccionDescanso:
    sum(c in cruces: c.i ==s.equipo)
      (X[c][s.ti+S[<s.equipo,s.numTrip>].longitud] + X[<c.j,c.i>][s.ti+S[<s.
    equipo,s.numTrip>].longitud])<= 2*(1-Z[s]);

  //6. Para que cada equipo juegue al menos un partido en los H primeras rondas
     posibles de juego
  forall (i in equipos)
```

```
    RestriccionEmpezar:
    sum(j in equipos: j!=i)sum (k in 1..H)(X[<i,j>][k] + X[<j,i>][k])>=1;
};

/*Se crea un archivo .txt en el que se guardan los trips utilizados en el
subproblema1.1. Esta información es necesaria para resolver el
subproblema1-2*/

execute{
  var ofile = new IloOplOutputFile("TripsUtilizados_Fase1_Geneticos.txt");
  for (var i in IndexZ){
    if(Z[i] ==1){
      ofile.writeln(i.equipo,",",i.numTrip, "," , i.ti);
    }
  }
};

/*Se crea un archivo .txt en el que se guardan los partidos que ya han sido
programados en el subproblema1-1. Esta información es necesaria para resolver
el subproblema1-2*/
execute{
  var ofile = new IloOplOutputFile("Partidos_Fase1_Geneticos.txt");
  for (var k in K){
    for (var c in cruces){
      if (X[c][k] !=0){
        ofile.writeln(c.i,',',c.j,',',k);
      }
    }
  }
};
```

C.4.3. Subproblema1-2

```
int D = ...; //Número de rondas disponibles para jugar partidos

int T = ...; //Ronda en el que se tiene jugar después de un Trip un partido en
    casa
int H = ...; //En las primeras H rondas de juego se debe jugar un partido

int P = ...; //Número minimo de partidos que se tienen que programar

int U = ...; //Número minimo de partidos que tiene que jugar cada equipo
int L = ...; //Número máximo de partidos que tiene que jugar cada equipo

int Uloc = ...; //Número minimo de partidos que tiene que jugar cada equipo
    como local
int Lloc = ...; //Número máximo de partidos que tiene que jugar cada equipo
    como local

int Uvis = ...; //Número minimo de partidos que tiene que jugar cada equipo
    como visitante
int Lvis = ...; //Número mñaximo de partidos que tiene que jugar cada equipo
    como visitante

range K = 1..D;

{string} equipos = ...; //Nombre de todos los posibles equipos
int NumeroTrips[equipos]=...; //Número de posibles trips de cada equipo

//Tupla que contiene el nombre de un equipo y el número de trip
tuple IndexEquipoTrip{
  string equipo;
```

```
  int num;
}

/*Generamos el posible conjunto de indices para la variable Z_{sk} donde
s indica el trip entre todos los posibles, que se escoge del conjunto de
    indices */
{IndexEquipoTrip} IndexTrip = {<i,n> | i in equipos , n in 1..NumeroTrips[i]};

//Tupla que contiene los elementos que hay en los trips; excluyendo el índice
tuple Trip{
  int longitud;
  {string} visitantes;
  float coste;
  }

// Lectura de todos los posibles trips
Trip S[IndexTrip] = ...;

//Tupla que contiene dos equipos
tuple Cruces{
string i;
string j;
};

//Cruces hace referencia al conjunto de todos los enfrentamientos posibles que
    debe haber
{Cruces} cruces={ <i,j> | i in equipos ,j in equipos: i!=j};

tuple indexz{
   string equipo;
   int numTrip;
   int ti;
};

tuple indexx{
  string i;
  string j;
  int t;
};

//Restriccion necesaria para que la variable Z_sk este bien definida
{indexz} IndexZ = {<i,n,k> | <i,n> in IndexTrip, k in K:  k<= D-S[<i,n>].
    longitud +1 };

{indexz} ZPrev ;
{indexx} XPrev ;

//Lectura de los resultados obtenidos en el problema 1-1

//XPrev
execute{
var f = new IloOplInputFile("Partidos_Fase1_Geneticos.txt");
  while (!f.eof) {
   var data = f.readline().split(",");
   if (data.length == 3)
     XPrev.add(data[0],data[1], Opl.intValue(data[2]));
  }
  writeln(XPrev);
}

//ZPrev
execute{
var f = new IloOplInputFile("TripsUtilizados_Fase1_Geneticos.txt");
```

```
  while (!f.eof) {
   var data = f.readline().split(",");
   if (data.length == 3)
     ZPrev.add(data[0],Opl.intValue(data[1]), Opl.intValue(data[2]));
  }
  writeln(ZPrev);
}


int posicion[IndexTrip][equipos];
int estaEnTrip[IndexTrip][equipos];
execute{
var cont=0;
for(var i in IndexTrip){
for(var j in equipos)
estaEnTrip[i][j]=0;
}
for(var i in IndexTrip){
  for(var vis in S[i].visitantes){
   posicion[i][vis]=cont;
   estaEnTrip[i][vis]=1;
   cont=cont+1;
   }
   cont=0;
}
};

//Variables de decisión
dvar boolean X[cruces][K]; //Corresponde a la variable X_{ijk}
dvar boolean Z[IndexZ] ; //Corresponde a la variable Z_{sk}

//Función Objetivo
dexpr float CosteTotal =  sum (s in IndexZ) Z[s]*S[<s.equipo, s.numTrip>].coste
    ;
minimize
 CosteTotal;

//Restricciones
subject to{
  // Lectura de las variables que han tomado valor en el subproblema1-1
    forall (i in ZPrev)
      ObligacionTrips:
      Z[i] == 1;

     forall (c in XPrev)
       ObligacionJugar:
       X[<c.i,c.j>][c.t] == 1;

  //1. No es necesario que se juegen todos los partidos
  forall (c in cruces)
    RestriccionJugar:
    sum (k in K)(X[c][k]) <= 1;

  //1.1 Como es un subproblema, se deben jugar al menos (globalmente) P
    partidos en las D rondas disponibles
  partidos_minimos:sum(c in cruces, k in K)(X[c][k])>= P ;

  //1.2 Para que cada equipo juege un número de partidos, comprendidos en el
    intervalo [U,L]
  forall (i in equipos)
    min_max_partidos:
    U <= sum(k in K)(sum (c in cruces: c.i == i)X[c][k] + sum(c in cruces: c.j
    ==i)X[c][k]) <= L;
```

```
144
145   //1.3 Para que cada equipo juegue un número de partidos como local,
        comprendidos en el intervalo [Umin , Lmin]
146   forall (i in equipos)
147     min_local_partidos:
148     Uloc <= sum(k in K)(sum(c in cruces:c.i ==i)X[c][k]) <= Lloc;
149
150   //1.4 Para que cada equipo juege un número de partidos como visitante,
        comprendidos en el intervalo [Umax, Lmax]
151   forall (i in equipos)
152     min_visitante_partidos:
153     Uvis <= sum(k in K)(sum(c in cruces:c.j ==i)X[c][k]) <= Lvis;
154
155   //2. Para que en un día cada equipo juegue a lo sumo un partido
156   forall (i in equipos, k in K)
157     RestriccionCapacidad:
158     sum (j in equipos: j !=i)(X[<i,j>][k] + X[<j,i>][k]) <=1;
159
160   //3. Para que los partidos se juegen entre los posibles trips
161   forall (c in cruces, k in K)
162     RestriccionTrip:
163     X[c][k] == sum(s in IndexZ: s.equipo==c.j && estaEnTrip[<s.equipo,s.numTrip
        >][c.i]!=0 && s.ti==k-posicion[<s.equipo,s.numTrip>][c.i]) Z[s];
164
165   //4. Para que se juege un partido en casa en los T días siguientes al
        finalizar un trip
166   forall (s in IndexZ: s.ti+S[<s.equipo,s.numTrip>].longitud-1 < D)
167       RestriccionDespuesTrip:
168       sum (c in cruces: c.i ==s.equipo)
169         sum(t in 1..T: s.ti+S[<s.equipo,s.numTrip>].longitud+t -1<=D)
170         (X[c][s.ti+S[<s.equipo,s.numTrip>].longitud+t-1])>=Z[s];
171
172   //5. Asignar un día de descanso después de finalizar un trip
173   forall (s in IndexZ: 1<s.ti<= D -S[<s.equipo,s.numTrip>].longitud)
174     RestriccionDescanso:
175     sum(c in cruces: c.i ==s.equipo)
176       (X[c][s.ti+S[<s.equipo,s.numTrip>].longitud] + X[<c.j,c.i>][s.ti+S[<s.
      equipo,s.numTrip>].longitud])<= 2*(1-Z[s]);
177
178   //6. Para que cada equipo juegue al menos un partido en los H primeras rondas
       posibles de juego
179   forall (i in equipos)
180     RestriccionEmpezar:
181     sum(j in equipos: j!=i)sum (k in 1..H)(X[<i,j>][k] + X[<j,i>][k])>=1;
182 };
183
184 /*Se crea un archivo .txt en el que se guardan los trips utilizados en el
185 subproblema1-2. Esta información es necesaria para resolver el
186 subproblema1-3*/
187
188 execute{
189   var ofile = new IloOplOutputFile("TripsUtilizados_Fase2_Geneticos.txt");
190   for (var i in IndexZ){
191     if(Z[i] ==1){
192       ofile.writeln(i.equipo,",",i.numTrip, "," , i.ti);
193     }
194   }
195 };
196
197 /*Se crea un archivo .txt en el que se guardan los partidos que ya han sido
198 programados en el subproblema1-2. Esta información es necesaria para resolver
199 el subproblema1-3*/
200 execute{
```

```
  var ofile = new IloOplOutputFile("Partidos_Fase2_Geneticos.txt");
  for (var k in K){
    for (var c in cruces){
      if (X[c][k] !=0){
        ofile.writeln(c.i,',',c.j,',',k);
      }
    }
  }
};
```

C.4.4. Subproblema1-4

```
int D = ...; //Número de rondas disponibles para jugar partidos

int T = ...; //Ronda en el que se tiene jugar después de un Trip un partido en
    casa
int H = ...; //En las primeras H rondas de juego se debe jugar un partido

range K = 1..D;

{string} equipos = ...; //Nombre de todos los posibles equipos
int NumeroTrips[equipos]=...; //Número de posibles trips de cada equipo

//Tupla que contiene el nombre de un equipo y el número de trip
tuple IndexEquipoTrip{
  string equipo;
  int num;
}

/*Generamos el posible conjunto de indices para la variable Z_{sk} donde
s indica el trip entre todos los posibles, que se escoge del conjunto de
    indices */
{IndexEquipoTrip} IndexTrip = {<i,n> | i in equipos , n in 1..NumeroTrips[i]};

//Tupla que contiene los elementos que hay en los trips; excluyendo el índice
tuple Trip{
  int longitud;
  {string} visitantes;
  float coste;
  }

// Lectura de todos los posibles trips
Trip S[IndexTrip] = ...;

//Tupla que contiene dos equipos
tuple Cruces{
string i;
string j;
};

//Cruces hace referencia al conjunto de todos los enfrentamientos posibles que
    debe haber
{Cruces} cruces={ <i,j> | i in equipos ,j in equipos: i!=j};

tuple indexz{
   string equipo;
   int numTrip;
   int ti;
};

tuple indexx{
  string i;
```

```
  string j;
  int t;
};

//Restriccion necesaria para que la variable Z_sk este bien definida
{indexz} IndexZ = {<i,n,k> | <i,n> in IndexTrip, k in K:  k<= D-S[<i,n>].
    longitud +1 };

{indexz} ZPrev ;
{indexx} XPrev ;

//Lectura de los resultados obtenidos en el problema 1-3

//XPrev
execute{
var f = new IloOplInputFile("Partidos_Fase3_Geneticos.txt");
  while (!f.eof) {
   var data = f.readline().split(",");
   if (data.length == 3)
     XPrev.add(data[0],data[1], Opl.intValue(data[2]));
  }
  writeln(XPrev);
}

//ZPrev
execute{
var f = new IloOplInputFile("TripsUtilizados_Fase3_Geneticos.txt");
  while (!f.eof) {
   var data = f.readline().split(",");
   if (data.length == 3)
     ZPrev.add(data[0],Opl.intValue(data[1]), Opl.intValue(data[2]));
  }
  writeln(ZPrev);
}


int posicion[IndexTrip][equipos];
int estaEnTrip[IndexTrip][equipos];
execute{
var cont=0;
for(var i in IndexTrip){
for(var j in equipos)
estaEnTrip[i][j]=0;
}
for(var i in IndexTrip){
  for(var vis in S[i].visitantes){
   posicion[i][vis]=cont;
   estaEnTrip[i][vis]=1;
   cont=cont+1;
   }
   cont=0;
}
};

//Variables de decisión
dvar boolean X[cruces][K]; //Corresponde a la variable X_{ijk}
dvar boolean Z[IndexZ] ; //Corresponde a la variable Z_{sk}

//Función Objetivo
dexpr float CosteTotal =  sum (s in IndexZ) Z[s]*S[<s.equipo, s.numTrip>].coste
    ;
minimize
 CosteTotal;
```

```
109
110 //Restricciones
111 subject to{
112   // Lectura de las variables que han tomado valor en el subproblema1-3
113     forall (i in ZPrev)
114       ObligacionTrips:
115       Z[i] == 1;
116
117      forall (c in XPrev)
118        ObligacionJugar:
119        X[<c.i,c.j>][c.t] == 1;
120
121   //1. Para que se jueguen todos los partidos; es decir todos los equipos
       jueguen contra todos el partido de ida y el partido de vuelta
122   forall (c in cruces)
123     RestriccionJugar:
124     sum (k in K)(X[c][k]) == 1;
125
126   //2. Para que en un día cada equipo juegue a lo sumo un partido
127   forall (i in equipos, k in K)
128     RestriccionCapacidad:
129     sum (j in equipos: j !=i)(X[<i,j>][k] + X[<j,i>][k]) <=1;
130
131   //3. Para que los partidos se juegen entre los posibles trips
132   forall (c in cruces, k in K)
133     RestriccionTrip:
134     X[c][k] == sum(s in IndexZ: s.equipo==c.j && estaEnTrip[<s.equipo,s.numTrip
       >][c.i]!=0 && s.ti==k-posicion[<s.equipo,s.numTrip>][c.i]) Z[s];
135
136   //4. Para que se juege un partido en casa en los T días siguientes al
       finalizar un trip
137   forall (s in IndexZ: s.ti+S[<s.equipo,s.numTrip>].longitud-1 < D)
138       RestriccionDespuesTrip:
139       sum (c in cruces: c.i ==s.equipo)
140         sum(t in 1..T: s.ti+S[<s.equipo,s.numTrip>].longitud+t -1<=D)
141         (X[c][s.ti+S[<s.equipo,s.numTrip>].longitud+t-1])>=Z[s];
142
143   //5. Asignar un día de descanso después de finalizar un trip
144   forall (s in IndexZ: 1<s.ti<= D -S[<s.equipo,s.numTrip>].longitud)
145     RestriccionDescanso:
146     sum(c in cruces: c.i ==s.equipo)
147       (X[c][s.ti+S[<s.equipo,s.numTrip>].longitud] + X[<c.j,c.i>][s.ti+S[<s.
      equipo,s.numTrip>].longitud])<= 2*(1-Z[s]);
148
149   //6. Para que cada equipo juegue al menos un partido en los H primeras rondas
       posibles de juego
150   forall (i in equipos)
151     RestriccionEmpezar:
152     sum(j in equipos: j!=i)sum (k in 1..H)(X[<i,j>][k] + X[<j,i>][k])>=1;
153 };
154
155 /*Se crea un archivo .txt en el que se guarda el orden en el que cada equipo
      tiene
156 que jugar sus partidos. Esta información se utilizará para resolver el segundo
157 problema*/
158 execute{
159   var ofile = new IloOplOutputFile("Partidos_NBA_Geneticos.txt");
160     for (var i in equipos){
161       ofile.write('"',  i , '"', ':', '{');
162     for(var k in K){
163       for (var c in cruces){
164         if (X[c][k] !=0 && i == c.i | i ==c.j){
165           ofile.write( c,',');
```

```
          }}}
          ofile.writeln('},')}
  ofile.close();
};


/*Se crea un archivo .txt en el que guardar los trips que han sido elegidos. Esta
información se utilizará para resolver el segundo problema*/
execute{
  var ofile = new IloOplOutputFile("TripsUtilizados_NBA_Geneticos.txt");
  for(var i in IndexZ){
    if(Z[i]==1){
        ofile.writeln('<',i.equipo,",",i.numTrip , '>:', '<' , S[IndexTrip.get(i.equipo,i.numTrip)].longitud,",",
        S[IndexTrip.get(i.equipo,i.numTrip)].visitantes , '>,');
     }
}
ofile.close();
};

//Ver los kilómetros recorridos por cada equipo
execute{
  var ofile = new IloOplOutputFile("KilometrosRecorridos_NBA_Geneticos.txt");

  for(var j in equipos){
  km = 0
  ofile.write('"',  j , '"', ':');
  for(var i in IndexZ){
    if(Z[i]==1 && i.equipo == j){
        km = km + S[IndexTrip.get(i.equipo,i.numTrip)].coste
      }}
  ofile.writeln(km)};
ofile.close();
};
```

C.4.5. Problema 2

Problema2.mod

```
int H = ...; //Número de días reales en los que se pueden jugar los partidos
range T = 1..H;

int R = ...; //Número de partidos que tiene que jugar un equipo, comenzando en el 0

int D = ...; //Días siguientes en los que se debe jugar un partido después del anterior

int J = ...; //En los J primeros días de juego cada equipo debe jugar al menos un partido

{string} equipos = ...; //Nombre de todos los posibles equipos

int NumeroTrips[equipos]=...; //Numero de posibles trips de cada equipo

//Tupla que contiene el nombre de un equipo y el número de trip.
tuple IndexEquipoTrip{
  string equipo;
  int num;
}

```

```
//Conjunto de índices para la lectura de los trips utilizados
{IndexEquipoTrip} IndexTam = {<i,n> | i in equipos , n in 1..NumeroTrips[i]};

//Tupla que contiene el tamaño del trip y los rivales
tuple Trip{
  int tam;
  {string} oponentes;
  }

//Lectura de los trips utilizados
Trip TR[IndexTam] = ...;

//Tupla que contiene el equipo local y el equipo visitante
tuple Cruces{
string home;
string away;
};

//Lectura del orden de los partidos para cada equipo
ordered {Cruces} P[equipos] = ...;

//Tupla que contiene el nombre del equipo y el nombre de los partidos que tiene
    que jugar en orden
tuple indices{
  string equipos;
  Cruces p;
};

ordered {indices} ind;

execute{
  for (var i in equipos){
    for (var p in P[i]){
      ind.add(i,p);
    };
  };
};

//Declaración de las variables de decisión
dvar boolean Y[ind][T]; //Corresponde a la variable Y_{ipt}
dvar boolean da[ind]; //Corresponde a la variable da_{ip}

//Función objetivo
dexpr float descansos =  sum (p in ind) da[p];
dexpr float total = sum (p in ind, t in T: t>40)t*Y[p][t];
minimize
  //descansos;
 1000*descansos + 0.01*total;

//Restricciones

subject to{
  //1. Para que se asignen todos los partidos que se tienen que jugar a un día
    concreto
  forall (p in ind)
    RestriccionAsignar:
    sum(t in T)(Y[p][t]) == 1;

  //2. Para que en un día cada equipo juegue a lo sumo un partido
  forall (i in equipos, t in T)
    RestriccionCapacidad:
    sum(p in ind: p.equipos == i) Y[p][t]<=1;

```

```
81  //3. Para que se respete el orden en el que se tienen que jugar los partidos
82  forall (i in equipos)
83    forall (r in 0..R-1)
84      forall (t in T :t<H)
85      RestriccionOrden:
86      Y[<i,item(P[i],r)>][t] <= sum(s in 1..D: t+s<=H)Y[<i,item(P[i],r+1)>][t+s
    ];
87
88  //Restricción simétrica para el orden de los partidos
89  forall (i in equipos)
90    forall (r in 1..R)
91      forall (t in T :t>1)
92      RestriccionSimetrica:
93      Y[<i,item(P[i],r)>][t] <= sum(s in 1..D:t-s>0)Y[<i,item(P[i],r-1)>][t-s];
94
95  //4. Para que los equipos que comparten pabellón no juegen en casa el mismo
     dia
96
97  //5.
98
99  //Asignar dias de descanso dentro de un trip, trip de tamaño 2
100 forall (s in IndexTam: TR[s].tam ==2)
101   forall (t in T:t<H-2)
102   RestriccionDescansoTrip2:
103   Y[<s.equipo, <item(TR[s].oponentes,0),s.equipo>>][t]<=Y[<s.equipo, <item(TR
    [s].oponentes,1),s.equipo>>][t+1]+Y[<s.equipo, <item(TR[s].oponentes,1),s.
    equipo>>][t+2];
104
105
106 //Asignar días de descanso dentro de un trip, trip de tamaño 3
107 forall (s in IndexTam: TR[s].tam ==3)
108   forall (t in T:t<H-3)
109   RestriccionDescansoTrip3:
110    Y[<s.equipo, <item(TR[s].oponentes,0),s.equipo>>][t] <= Y[<s.equipo, <item
    (TR[s].oponentes,2),s.equipo>>][t+3];
111
112 //6. Asignar días de descanso después de un trip; siendo p el ultimo partido
     del trip
113 //Trips de tamaño 1
114 forall (s in IndexTam: TR[s].tam ==1)
115  forall (t in T:t<H && t>1)
116    RestriccionDescansoDespues1:
117    if (ord(P[s.equipo],<item(TR[s].oponentes,0),s.equipo>)<R ){
118    Y[<s.equipo, <item(TR[s].oponentes,0),s.equipo>>][t]
119    + Y[<s.equipo, item(P[s.equipo],ord(P[s.equipo],<item(TR[s].oponentes,0),s
    .equipo>) +1)>][t+1] <=1 + da[<s.equipo, <item(TR[s].oponentes,0),s.equipo
    >>];}
120
121 //Trips de tamaño 2
122 forall (s in IndexTam: TR[s].tam ==2 )
123  forall (t in T:t<H-1 && t>2)
124    RestriccionDescansoDespues2:
125    if (ord(P[s.equipo],<item(TR[s].oponentes,1),s.equipo>)<R){
126    Y[<s.equipo, <item(TR[s].oponentes,1),s.equipo>>][t]
127    + Y[<s.equipo, item(P[s.equipo],ord(P[s.equipo],<item(TR[s].oponentes,1),s
    .equipo>) +1)>][t+1]
128      + Y[<s.equipo, item(P[s.equipo],ord(P[s.equipo],<item(TR[s].oponentes,1),
    s.equipo>) +1)>][t+2]
129      <=1 + da[<s.equipo, <item(TR[s].oponentes,1),s.equipo>>];}
130
131 //Trips de tamaño 3
132 forall (s in IndexTam: TR[s].tam ==3)
133  forall (t in T:t<H-2 && t>1)
```

```
134      RestriccionDescansoDespues3:
135      if (ord(P[s.equipo],<item(TR[s].oponentes,2),s.equipo>)<R){
136      Y[<s.equipo, <item(TR[s].oponentes,2),s.equipo>>][t]
137      + Y[<s.equipo, item(P[s.equipo],ord(P[s.equipo],<item(TR[s].oponentes,2),s
     .equipo>) +1)>][t+1]
138      + Y[<s.equipo, item(P[s.equipo],ord(P[s.equipo],<item(TR[s].oponentes,2),s
     .equipo>) +1)>][t+2]
139        <=1 + da[<s.equipo, <item(TR[s].oponentes,2),s.equipo>>];}
140
141   //7. Para que se asigne el mismo día al partido entre dos equipos
142   forall(p in ind, q in ind , t in T : p.p == q.p)
143     RestriccionCoincidir:
144     Y[p][t] == Y[q][t];
145
146   //8. Para que todos los equipos jueguen un partido en los primeros J días de
       juego
147   forall (i in equipos)
148     RestriccionEmpezar:
149     sum(p in P[i])sum(t in 1..J) (Y[<i,p>][t]) >=1;
150
151   //Eliminamos variables que sabemos con certeza que son cero
152   //Para que el segundo partido de un equipo no pueda asignarse el primer día
153   forall(i in equipos)
154    forall(r in 1..R) //partidos
155      forall(t in 1..r)
156       Y[<i,item(P[i],r)>][t]==0;
157
158   //Para que el penúltimo partido de un equipo no pueda asignarse el último día
159   forall(i in equipos)
160    forall(r in 0..R-1) //partidos
161      forall(t in H-(R-r-1)..H)
162      Y[<i,item(P[i],r)>][t]==0;
163 };
164
165 //Ver el dia en el que queda asignado cada partido de cada equipo y si se
     cumplen los descansos necesarios
166
167 execute{
168     for(var c in ind){
169       for (var t in T){
170         if (Y[c][t] !=0){
171           writeln (c,t, ",",da[c]);
172         }}}}
173
174 //Crea el calendario de partidos buscado
175 execute{
176   var ofile = new IloOplOutputFile("calendario_NBA_genetico.xls");
177
178   ofile.write("Equipo,");
179
180   for(var t in T)
181   {
182     ofile.write(t ,',');
183   };
184   ofile.writeln(" ");
185   for(var e in equipos)
186   {
187       ofile.write( '"',  e , '",');
188       for(var t in T)
189       {
190        var pinta = true;
191         for (var c in ind)
192         {
```

```
193             if(c.equipos==e)
194             {
195              if (Y[c][t]!=0)
196              {
197                ofile.write('"',  c.p.home,  ",", c.p.away, '",');
198                pinta=false;
199              }
200             }
201           }
202           if(pinta==true)
203           {
204               ofile.write(", ");
205           }
206         }
207         ofile.writeln(" ")
208     }
209     ofile.close();
210 };
```

Problema2.dat

```
1  equipos = {"Hawks", "Celtics", "Nets", "Hornets", "Bulls", "Cavaliers", "
       Mavericks", "Nuggets", "Pistons", "Warriors", "Rockets", "Pacers"};
2
3  NumeroTrips = [20,20,20,20,20,20,
4  20,20,20,20,20,20];
5
6  H = 65; //Número de días en los que se pueden jugar partidos
7
8  R = 21; //Número de partidos que tiene que jugar un equipo, comenzando en el 0
9
10 D = 9; //Días siguientes en los que se debe jugar un partido después del
       anterior
11
12 J = 4; //En los J primeros días de juego cada equipo debe jugar al menos un
       partido
13
14 //Lectura del orden en el que cada equipo debe jugar sus partidos
15
16  P = #[
17 "Hawks":{ <"Hawks" "Hornets">, <"Bulls" "Hawks">, <"Nuggets" "Hawks">, <"
       Warriors" "Hawks">, <"Hawks" "Warriors">, <"Hawks" "Nets">, <"Hawks" "
       Cavaliers">, <"Hawks" "Rockets">, <"Hawks" "Celtics">, <"Rockets" "Hawks">,
       <"Mavericks" "Hawks">, <"Hawks" "Pistons">, <"Hornets" "Hawks">, <"Celtics"
       "Hawks">, <"Nets" "Hawks">, <"Hawks" "Pacers">, <"Cavaliers" "Hawks">, <"
       Pistons" "Hawks">, <"Pacers" "Hawks">, <"Hawks" "Mavericks">, <"Hawks" "
       Bulls">, <"Hawks" "Nuggets">,},
18  .
19  .
20  .
21 "Pacers":{ <"Pacers" "Celtics">, <"Celtics" "Pacers">, <"Nets" "Pacers">, <"
       Hornets" "Pacers">, <"Pacers" "Nets">, <"Pacers" "Bulls">, <"Pacers" "
       Mavericks">, <"Pacers" "Cavaliers">, <"Nuggets" "Pacers">, <"Warriors" "
       Pacers">, <"Bulls" "Pacers">, <"Pacers" "Warriors">, <"Pacers" "Hornets">, <
       "Hawks" "Pacers">, <"Rockets" "Pacers">, <"Mavericks" "Pacers">, <"Pacers" "
       Hawks">, <"Pacers" "Pistons">, <"Cavaliers" "Pacers">, <"Pistons" "Pacers">,
        <"Pacers" "Nuggets">, <"Pacers" "Rockets">,}
22 ]#;
23
24 //Lectura de los trips utilizados
25 TR =
26 #[
```

```
<Hawks,1>:<3, {"Hornets" "Celtics" "Nets"}>,
<Hawks,2>:<3, {"Cavaliers" "Pistons" "Pacers"}>,
<Hawks,3>:<3, {"Bulls" "Nuggets" "Warriors"}>,
<Hawks,4>:<2, {"Rockets" "Mavericks"}>,
 .
 .
 .
<Pacers,1>:<3, {"Hawks" "Rockets" "Mavericks"}>,
<Pacers,2>:<3, {"Nuggets" "Warriors" "Bulls"}>,
<Pacers,3>:<3, {"Celtics" "Nets" "Hornets"}>,
<Pacers,9>:<2, {"Cavaliers" "Pistons"}>
]#;
```

Anexo D

Calendario optimizado para 12 equipos

Equipo	1	2	3	4	5	6	7	8
Hawks	Hawks,Hornets							
Celtics	Cavaliers,Celtics	Pacers,Celtics		Pistons,Celtics	Celtics,Pacers			
Nets	Nuggets,Nets	Warriors,Nets		Bulls,Nets			Nets,Pacers	Rockets,Nets
Hornets	Hawks,Hornets	Rockets,Hornets		Mavericks,Hornets				Hornets,Pacers
Bulls				Bulls,Nets			Bulls,Pistons	
Cavaliers	Cavaliers,Celtics							
Mavericks		Mavericks,Nuggets		Mavericks,Hornets				
Nuggets	Nuggets,Nets	Mavericks,Nuggets	Rockets,Nuggets		Warriors,Nuggets			Nuggets,Pistons
Pistons				Pistons,Celtics			Bulls,Pistons	Nuggets,Pistons
Warriors		Warriors,Nets			Warriors,Nuggets			
Rockets		Rockets,Hornets	Rockets,Nuggets					Rockets,Nets
Pacers		Pacers,Celtics			Celtics,Pacers		Nets,Pacers	Hornets,Pacers

	9	10	11	12	13	14	15	16
Hawks	Bulls,Hawks		Nuggets,Hawks	Warriors,Hawks				
Celtics				Celtics,Mavericks				
Nets	Mavericks,Nets		Pacers,Nets		Nets,Mavericks			
Hornets	Nuggets,Hornets		Warriors,Hornets	Bulls,Hornets				
Bulls	Bulls,Hawks			Bulls,Hornets	Pistons,Bulls	Cavaliers,Bulls		Pacers,Bulls
Cavaliers		Cavaliers,Mavericks				Cavaliers,Bulls		Mavericks,Cavaliers
Mavericks	Mavericks,Nets	Cavaliers,Mavericks		Celtics,Mavericks	Nets,Mavericks			Mavericks,Cavaliers
Nuggets	Nuggets,Hornets		Nuggets,Hawks					
Pistons		Warriors,Pistons			Pistons,Bulls			
Warriors		Warriors,Pistons	Warriors,Hornets	Warriors,Hawks				
Rockets								
Pacers			Pacers,Nets					Pacers,Bulls

	17	18	19	20	21	22	23	24
Hawks	Hawks,Warriors	Hawks,Nets	Hawks,Cavaliers					
Celtics					Nets,Celtics		Hornets,Celtics	
Nets	Hornets,Nets	Hawks,Nets			Nets,Celtics			
Hornets	Hornets,Nets		Hornets,Warriors				Hornets,Celtics	
Bulls			Bulls,Nuggets			Nuggets,Bulls	Warriors,Bulls	
Cavaliers	Rockets,Cavaliers		Hawks,Cavaliers				Cavaliers,Nuggets	
Mavericks								Pacers,Mavericks
Nuggets		Pistons,Nuggets	Bulls,Nuggets			Nuggets,Bulls	Cavaliers,Nuggets	
Pistons		Pistons,Nuggets		Pistons,Warriors				
Warriors	Hawks,Warriors		Hornets,Warriors	Pistons,Warriors			Warriors,Bulls	
Rockets	Rockets,Cavaliers							
Pacers								Pacers,Mavericks

	25	26	27	28	29	30	31	32
Hawks	Hawks,Rockets		Hawks,Celtics			Rockets,Hawks	Mavericks,Hawks	
Celtics		Celtics,Nuggets	Hawks,Celtics		Rockets,Celtics	Mavericks,Celtics		
Nets	Nets,Nuggets							
Hornets		Hornets,Rockets						
Bulls			Bulls,Mavericks					Bulls,Pacers
Cavaliers			Pacers,Cavaliers	Pistons,Cavaliers			Cavaliers,Rockets	
Mavericks		Pistons,Mavericks	Bulls,Mavericks			Mavericks,Celtics	Mavericks,Hawks	
Nuggets	Nets,Nuggets	Celtics,Nuggets			Nuggets,Pacers			
Pistons		Pistons,Mavericks		Pistons,Cavaliers				
Warriors						Warriors,Pacers		
Rockets	Hawks,Rockets	Hornets,Rockets			Rockets,Celtics	Rockets,Hawks	Cavaliers,Rockets	
Pacers			Pacers,Cavaliers		Nuggets,Pacers	Warriors,Pacers		Bulls,Pacers

	33	34	35	36	37	38	39	40
Hawks		Hawks,Pistons	Hornets,Hawks	Celtics,Hawks		Nets,Hawks		
Celtics	Celtics,Rockets	Celtics,Cavaliers		Celtics,Hawks				
Nets		Nets,Rockets		Nets,Cavaliers		Nets,Hawks		
Hornets			Hornets,Hawks		Hornets,Cavaliers	Pacers,Hornets		Pistons,Hornets
Bulls				Bulls,Warriors				
Cavaliers		Celtics,Cavaliers		Nets,Cavaliers	Hornets,Cavaliers			
Mavericks					Mavericks,Pistons	Nuggets,Mavericks	Warriors,Mavericks	
Nuggets	Nuggets,Warriors					Nuggets,Mavericks		
Pistons		Hawks,Pistons	Rockets,Pistons		Mavericks,Pistons			Pistons,Hornets
Warriors	Nuggets,Warriors		Pacers,Warriors	Bulls,Warriors			Warriors,Mavericks	Rockets,Warriors
Rockets	Celtics,Rockets	Nets,Rockets	Rockets,Pistons					Rockets,Warriors
Pacers			Pacers,Warriors			Pacers,Hornets		

	41	42	43	44	45	46	47	48
Hawks	Hawks,Pacers				Cavaliers,Hawks		Pistons,Hawks	Pacers,Hawks
Celtics	Celtics,Pistons	Celtics,Bulls						
Nets		Nets,Pistons	Nets,Bulls				Nets,Warriors	
Hornets	Cavaliers,Hornets			Hornets,Pistons	Hornets,Bulls			
Bulls		Celtics,Bulls	Nets,Bulls		Hornets,Bulls			
Cavaliers	Cavaliers,Hornets				Cavaliers,Hawks	Cavaliers,Warriors		Cavaliers,Pistons
Mavericks		Mavericks,Warriors		Mavericks,Pacers				Mavericks,Rockets
Nuggets						Nuggets,Rockets		
Pistons	Celtics,Pistons	Nets,Pistons		Hornets,Pistons			Pistons,Hawks	Cavaliers,Pistons
Warriors		Mavericks,Warriors			Warriors,Rockets	Cavaliers,Warriors	Nets,Warriors	
Rockets		Rockets,Pacers			Warriors,Rockets	Nuggets,Rockets		Mavericks,Rockets
Pacers	Hawks,Pacers	Rockets,Pacers		Mavericks,Pacers				Pacers,Hawks

	49	50	51	52	53	54	55	56
Hawks			Hawks,Mavericks					
Celtics	Celtics,Warriors	Bulls,Celtics	Nuggets,Celtics		Warriors,Celtics			Celtics,Nets
Nets								Celtics,Nets
Hornets	Hornets,Mavericks							
Bulls		Bulls,Celtics					Mavericks,Bulls	Rockets,Bulls
Cavaliers		Cavaliers,Pacers						
Mavericks	Hornets,Mavericks		Hawks,Mavericks	Rockets,Mavericks			Mavericks,Bulls	
Nuggets			Nuggets,Celtics					Pacers,Nuggets
Pistons	Pacers,Pistons			Pistons,Pacers				
Warriors	Celtics,Warriors				Warriors,Celtics			
Rockets				Rockets,Mavericks				Rockets,Bulls
Pacers	Pacers,Pistons	Cavaliers,Pacers		Pistons,Pacers				Pacers,Nuggets

	57	58	59	60	61	62	63	64
Hawks		Hawks,Bulls	Hawks,Nuggets					
Celtics				Celtics,Hornets				
Nets	Pistons,Nets		Cavaliers,Nets			Nets,Hornets		
Hornets	Hornets,Nuggets			Celtics,Hornets		Nets,Hornets		
Bulls		Hawks,Bulls			Bulls,Cavaliers	Bulls,Rockets		
Cavaliers			Cavaliers,Nets		Bulls,Cavaliers	Warriors,Cavaliers		Nuggets,Cavaliers
Mavericks								
Nuggets	Hornets,Nuggets		Hawks,Nuggets					Nuggets,Cavaliers
Pistons	Pistons,Nets			Pistons,Rockets				
Warriors						Warriors,Cavaliers		
Rockets			Pacers,Rockets	Pistons,Rockets		Bulls,Rockets		
Pacers			Pacers,Rockets					